Universidad Nacional Autónoma de México

Facultad de Ingeniería

"Estrategias de programación para robustecer la Seguridad Informática al desarrollar aplicaciones en Visual Basic .NET"

T E S I S

Q U E P R E S E N T A:

ALBERTO AXCANA DE LA MORA PLIEGO

P A R A O B T E N E R E L T I T U L O D E:

I N G E N I E R O E N C O M P U T A C I O N

Directora de Tesis:
M.C. Ma. Jaquelina López Barrientos

México, D.F. **Noviembre 2004.**

Este trabajo está dedicado a mi familia.

A mi mamá Nieves:

Por su gran ejemplo, su amor incondicional, su comprención, por siempre confiar en mi y no dudar que lo lograría. Gracias mamí por hacerme el hombre que hoy soy.

A mi papá José:

Por ser mi amigo, por el gran apoyo en los momentos más difíciles y de dudas, siempre dándome ánimo y consejos, que sólo un "grande" como tú puede dar.

y a mi hermana Ritva.

Por ser mi guía para lograr todas las cosas que me propongo, por orientarme y por ser mi confidente, gracias hermanita por todo el cariño que me regalas

~:-)

Gracias por ayudarme a cumplir este sueño.
Los amo

Reconocimientos

Este trabajo ha sido posible gracias a los enormes apoyos, recomendaciones y aliento que he recibido. En primer término, agradezco profundamente a la *M.C. Ma. Jaquelina López Barrientos*, maestra y directora de esta tesis, por las incontables horas de discusión, orientación y dedicación pedagógica para la elaboración de este trabajo.

Agradezco al *Ing. Orlando Zaldívar Zamorategui*, a la *Ing. Laura Sandoval Montaño*, al *Ing. Jorge Valeriano Assem*, y a la *Ing. María del Rosario Barragán Paz*, Sinodales de esta tesis, la paciencia y dedicación en la lectura, revisión y observaciones a este trabajo.

Agradezco a la *Unidad de Servicios de Cómputo Académico* por haberme dado la oportunidad de formar parte del grupo de Becarios, donde compartimos muchos de los conocimientos de nuestra vida escolar y profesional. En especial agradezco al *Ing. Noé Cruz Marín*, por su experiencia, sus conocimientos y consejos que conmigo ha compartido.

Agradezco a mi *Facultad de Ingeniería*, a mi *Universidad* y a todos mis *profesores* y *profesoras* de quienes adquirí los conocimientos para llegar a este lugar.

Hago un reconocimiento y agradecimiento especial a mis amigas y amigos que a lo largo de la carrera me enseñan, apoyan, aconsejan y acompañan: *Claudia Ibeth, José de Jesús, Marian, Margarita, Evangelina, Marisol, Víctor Hugo, Guillermo, Jesús, Alejandra, Carolina, Tanya Itzel, Eugenia, Verónica, Arlenee, Patricia, Alfredo y Paola.*

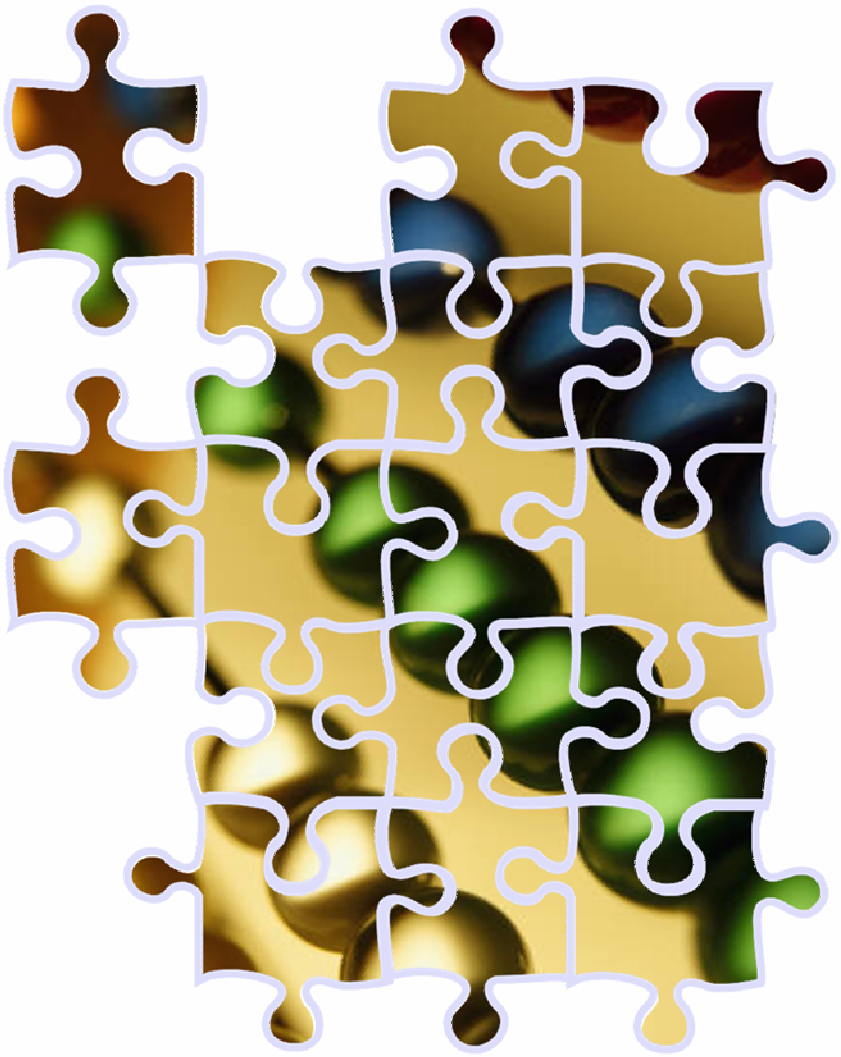
Hago una mención especial y respetuosa para agradecer a la *Mtra. Artemisa Pedrosa de De Gortari* su apoyo solidario y valioso tiempo dedicado para compartirme consejos y experiencias que me han permitido dar un sentido y trascendencia en las actividades que me desempeño.

Finalmente, dejo constancia de mi deuda con todas aquellas personas que de manera directa o indirecta han posibilitado mi superación personal con la conclusión de esta tesis.

ÍNDICE

Capítulo 1

Introducción



"ESTRATEGIAS DE PROGRAMACIÓN PARA ROBUSTECER LA SEGURIDAD INFORMÁTICA AL DESARROLLAR APLICACIONES EN VISUAL BASIC .NET"

1. INTRODUCCIÓN

Debido al incremento en las complejidades de las operaciones computacionales dentro de las organizaciones, a las necesidades de manejar mejores condiciones de seguridad en los programas, lo mismo para disminuir los tiempos de respuestas requeridas, la única forma de disponer del recurso de la información es teniendo aplicaciones y sistemas confiables que permitan a las Organizaciones e Instituciones estar a la vanguardia tecnológica y en responder oportunamente a las exigencias de un mercado cada vez más competitivo.

El aumento en las competencias de los programas computacionales, incrementa la necesidad de establecer políticas y procedimientos de seguridad efectivas que disminuyan los riesgos de que se produzcan escape, alteraciones o destrucciones de datos que sean de vital importancia para las organizaciones o usuarios que requieren de aplicaciones o sistemas.

La integridad, confidencialidad, confiabilidad y disponibilidad de la información sólo puede ser garantizada adoptando las estrategias de seguridad adecuadas al momento de diseñar e instrumentar las aplicaciones para dicha información.

La Seguridad es una condición importante de cualquier Servicio y Aplicación Informática, aunque a menudo ésta es postergada, basta tan sólo una brecha en la seguridad para crear graves daños. Por esto, el papel de la Seguridad Informática es cada vez más trascendente y no podrá ser ignorado, especialmente por el aumento de la exposición al riesgo que implica la cada vez mayor integración y globalización de los Sistemas y aplicaciones Informáticas.

Incluso, la industria del software en su creciente y decidida penetración en todos los aspectos de los negocios y procesos de las organizaciones modernas, es una de las realidades tangibles que la sociedad exige para generar mejores y seguras soluciones sistematizadas, que aumenten la efectividad de las acciones corporativas, den mayor valor agregado a sus clientes y se contrasten con las presiones e implicaciones que esto lleva en el desarrollo mismo de dichas soluciones.

En este sentido, al evaluar la necesidad de obtener un producto de software de calidad que permita a las organizaciones adelantar con oportunidad y alto contenido estratégico sus directrices de negocio, enfrentan un doble desafío para los dedicados a la programación y desarrollo de los sistemas de información de las organizaciones, así como para los clientes de los mencionados sistemas. Por un lado, todas las variables asociadas con la administración del proyecto en sí mismo, las que imprimen una importante complejidad al desarrollo de soluciones informáticas, y en segundo lugar, los altos estándares de aseguramiento en la calidad del desarrollo de software.

En la encrucijada de formalidad administrativa y técnica, la construcción de sistemas de información ofrece un desafío práctico para las nuevas generaciones de programadores y administradores de proyectos informáticos, y una especial atención de los experimentados

ingenieros de software para contextualizar sus aprendizajes en elementos conceptuales y formales que alimenten la práctica de la creación de software seguro.

Sólo los programadores perfectos son capaces de crear código perfecto desde el principio. El resto debe hacer frente a algunas imperfecciones que surgen durante el desarrollo de una aplicación que se desea que tenga éxito.

Por tanto y, considerando el gran reto de la construcción de software, la necesidad de soluciones de software intercomunicadas (orientadas al uso de redes de computadores), la utilización de modernos lenguajes de programación (*VISUAL BASIC .NET, C#, JAVA, PERL, PHP*, entre otros), la necesidad de soluciones eficientes y de alta portabilidad; el uso de metodologías de aseguramiento de calidad, se hace necesario revisar elementos relacionados con las prácticas y principios de programación segura como aspecto complementario del proceso de desarrollo de software.

Cuando se diseñan aplicaciones que interactúan con recursos del sistema o servidor, existen algunos problemas de seguridad que se deben solucionar para poder crear aplicaciones sencillas y prácticas para el usuario. La seguridad de los recursos conlleva un proceso que abarca el ciclo de desarrollo completo y en el que entran en juego diferentes tecnologías.

Aunque Microsoft Visual Studio .NET ofrece mayor control sobre la seguridad de aplicaciones en ejecución de Visual Basic, que el proporcionado en versiones anteriores, se requiere mayor precaución y responsabilidad por parte del programador y administrador durante las tareas de diseño, prueba e implementación sobre las aplicaciones que se desarrollen para lograr conseguir un nivel de seguridad elevado que permita aprovechar al máximo las tecnologías de seguridad.

Cuando se habla de conseguir un nivel de seguridad elevado se deben contemplar estrategias que permitan establecer los puntos de partida para la programación segura de aplicaciones y mecanismos, que además logren cubrir los aspectos de seguridad con respecto a la protección del servidor de los ataques de código malintencionado y del daño de datos. Si se establece el adecuado conjunto de reglas de programación que permita diseñar, desarrollar e instrumentar los mecanismos de seguridad, en el entorno de desarrollo, pueden aprovecharse para garantizar la seguridad de las aplicaciones y los servidores de desarrollo.

La seguridad dentro de la informática es un campo que abarca muchísimos aspectos. Un ingeniero en computación esta, quizás, mas acostumbrados a hablar de seguridad desde el punto de vista de protección de un sistema y es correcto, ya que eso es seguridad. Pero además el punto del que parten la gran mayoría de los fallos de la seguridad informática es en la programación. Al comenzar a establecer la seguridad debe partirse de la suposición que un programa falla o es susceptible de fallar, lo cual permite ingeniárselas para que no solo falle sino que además lo haga como uno quiere. Por eso desde el punto de vista del programador, se tiene que diseñar el programa para que sea lo más seguro posible.

Evidentemente debe tenerse claro que "No existe el programa perfecto", los programadores somos humanos y cometemos errores.

Sin embargo, existe una serie de aspectos básicos de diseño que todo programador debería seguir para evitar problemas de seguridad:

- Privilegios. Los programas deben funcionar con el menor número de privilegios posible. De esta manera no se evita el error, pero si se limita el daño que puede causar. Esta misma idea puede aplicarse no solo a un programa completo sino también a determinadas partes de un programa.
- Comprobaciones de acceso. Se deben comprobar todo tipo de accesos e interactividad de los programas.
- Máxima seguridad por defecto. Es la forma predeterminada o por defecto de funcionamiento de un programa, debe ser el de no permitir nada, se debe establecer claramente bajo que condiciones se permiten las operaciones.
- Recursos compartidos. Se debe minimizar en la medida de lo posible el acceso a recursos compartidos o recursos hardware.
- Transparencia. Todos estos mecanismos no deben interferir en ningún momento con el uso de la aplicación. Deben ser mecanismos transparentes al usuario, que no hagan que la aplicación sea mas complicada de usar o menos intuitiva.

En muchas ocasiones descuidar este tipo de aspectos resultan en efectos de mayores consecuencias. El presente trabajo de tesis es el análisis a una serie de cuestionamientos básicos, para no descuidar los anteriores y, probablemente, nuevos aspectos más avanzados, partiendo de cuestionamientos básicos como por ejemplo: ¿Por qué se programa código inseguro?

La idea de realizar estos cuestionamientos no es solo resolver el problema que plantea la pregunta anterior, sino también la de dar pautas para hacer otros cuestionamientos y de ser posible también resolverlos. Para poder responder al cuestionamiento anterior, observo que algunas de las variadas razones por las que los programadores escriben código inseguro están fundamentadas en los siguientes terminos:

- *No es un tema que se suela explicar en la Universidad*. Durante la carrera se cursan muchas asignaturas involucradas con la programación, en las que se conocen distintos lenguajes. En determinadas asignaturas se ven temas de seguridad, criptografía, protocolos, etc. pero no se enseña a programar de modo seguro: como evitar *buffer overflows*, comprobaciones en las operaciones de entrada/salida, etc. Este es uno de los principales problemas.
- *Algo parecido pasa con los libros*, existen muchísimos libros de programación, pero muy pocos hablan acerca de escribir programas seguros.
- *El lenguaje o plataforma de programación, C*, por ejemplo, es de por si un lenguaje inseguro, las funciones de manejo de caracteres de la librería estándar de C no son seguras. Independientemente del aspecto de seguridad es un lenguaje poderoso para realizar los motores de grandes aplicaciones y hoy en día muy utilizado para realizar pruebas y ataques de seguridad.

- Muchos programadores *se conforman con que su programa funcione* y si algo falla ya se corregirá en el futuro. El pensamiento tiene que ser el opuesto, hacer un buen programa, permitirá que el número de fallos que se tengan que corregir en el futuro sea menor.
- Programar de forma segura *conlleva un mayor tiempo de desarrollo*, esto hace que determinadas empresas publiquen código inseguro por cumplir unos plazos de mercado.
- Los *programadores somos humanos*, es prácticamente imposible no equivocarse cuando se escriben miles de líneas de código.

Existen muchas otras razones, que en este trabajo se muestran y dan soluciones para mitigarlos o disminuirlos. Como se puede observar, se resolvió el cuestionamiento de manera básica, pero se abrieron nuevos cuestionamientos que permiten enriquecer este tema.

Este trabajo de tesis está dirigido a los programadores de Visual Basic .NET que necesitan apoyo y experiencia en seguridad, y no para expertos en seguridad que no conocen Visual Basic .NET.

Lo elaboré con la metodología de análisis que la propia lógica humana, que dentro de los procesos de programación, permite ir diseñando y construyendo. Partí de la razón inductiva para ir infiriendo una secuencia de proyecciones y análisis que lleven a la construcción de una serie de escenarios de riesgos en materia de seguridad, a los que se enfrentarían las propuestas de aplicaciones realizadas en Visual Basic .NET. La metodología fue pensada para conseguir los objetivos al establecer estrategias, mecanismos y estructuras de seguridad en aplicaciones que se desarrollan en Visual Basic .NET contemplando etapas como:

La Investigación: Con esta etapa logré examinar y descubrir el entorno físico y lógico para el desarrollo de este proyecto, esto implicó conocer las tecnologías y el lenguaje donde se desarrollaron los estudios de los mecanismos de seguridad que se aplicaron.

El Análisis: La revisión de toda la información recopilada de los lenguajes de programación me permitió comprender la problemática de la seguridad informática durante el ciclo de desarrollo completo de las aplicaciones encontradas, que nos permitieron determinar las técnicas y requerimientos necesarios para cubrir las demandas y mecanismos de control que se desarrollaron en este proyecto.

El Desarrollo: A partir de la revisión de la serie de mecanismos que muestra el programa Visual Basic .NET, de las dudas que me fue mostrando el propio programa y de las analogías que se tienen de otros programas que se encuentran en proceso de revisión y de criticas. Considero que esta etapa debió estar constituida por dos acciones principales: el diseño y la prueba. Reglas que permiten adecuar y desarrollar aplicaciones y estrategias de seguridad.

El diseño y la prueba se presentan como un ciclo que permiten valorar la calidad de la programación.

El Diseño: Durante el diseño de las aplicaciones como de las estrategias de seguridad pretendí apoyarme en el análisis de la problemática para identificar plenamente el punto de seguridad que

se desea robustecer, valiéndome de las técnicas más propicias para solucionar el problema y sin descartar las modificaciones que pueden resultar de las pruebas para mejorar el diseño.

Las Pruebas: El experimentar y probar los diseños durante la etapa de desarrollo permitió obtener mejores resultados de las estrategias y los mecanismos de seguridad que se desarrollan. Pues se tiene claro que los errores que se presentaron, fueron de mucha ayuda las pruebas que se realizaron y así poder disminuir al mínimo los errores de programación que se pudiesen presentar.

La Aplicación: Una vez aprobados los mecanismos y estrategias de seguridad en las aplicaciones fue necesario hacer uso de estructuras, técnicas y herramientas al ponerlas en práctica con los procedimientos adecuados para asegurar que se consiguió el objetivo por el cual fueron desarrolladas.

La Documentación: Para mantener un control de la información fue necesario, durante esta etapa documentar, los procedimientos de instalación, aplicación, uso y mantenimiento así como describir el funcionamiento de las herramientas y estrategias que se hayan diseñado y probado. Esta última etapa permitió establecer las estrategias que se hubieron de seguir para construir las aplicaciones.

Las estrategias son comunes a los estilos de pensamiento de genios creativos en ciencia, ingeniería, arte e incluso industria. Para lograr obtener dichas estrategias a lo largo de la tesis se utilizaron una serie de reglas que son muy útiles al momento de realizar estrategias en cualquier campo y que yo las aplique cuando ví su utilidad como técnicas para desarrollar las estrategias, tal y como se describen:

- En principio es necesario *mirar los problemas* que se quieren solucionar de modos muy diferentes, hallando nuevas perspectivas que nadie antes ha adoptado.
- *¡Visualizar!* todo acerca del problema, de ser necesario debe formularse la problemática de tantos modos diferentes como sea posible, incluyendo el uso de diagramas y si se quiere visualizando las soluciones.
- *¡Producir!* características distintivas de las estrategias relacionadas con la productividad que proporcionan. Esta productividad brinda, para si y los colaboradores, ideas valiosas para enriquecer las estrategias.
- *Realizar nuevas combinaciones*. Combinando y mezclando ideas, imágenes y pensamientos en diferentes modos, no importa cuan incongruentes o insólitos sean.
- *Relacionar conexiones o analogías* con temas diferentes o ajenos al tema que se estudia para realizar la estrategia.
- *Pensar metafóricamente*. La metáfora es un signo de genialidad y capacidad de percibir parecidos entre dos áreas separadas de la realidad y relacionarlas conjuntamente, con la finalidad de lograr una abstracción de las características y necesidades de la estrategia o problemática.

- *Pensar en los opuestos y las reacciones*. Cuando se construyen estrategias, mientras se plantean soluciones deben pensarse a la par los problemas que causaran estas soluciones.
- *Probar*. Este ultimo punto forma parte de las pruebas e implantación de una estrategia, pues para decidir si la estrategia es la adecuada, debió previamente haber funcionado o fallado. Al prepararse uno mismo para una oportunidad, toda vez que se intenta hacer algo y falla, termina siendo un aprendizaje más. Este es el "principio del accidente creativo". El fracaso puede ser productivo sólo si no se considera como un resultado improductivo y si se analizan los procesos, sus partes, y cómo puede cambiarse o modificarse para obtener otros resultados. No hay que preguntarse "¿Por qué falló?" sino, "¿Qué se realizó?".

Los anteriores puntos permitieron tener una guía cuando se estaban realizando las estrategias de seguridad en el desarrollo de la tesis.

A lo largo del desarrollo de este trabajo se cumplió con una jerarquía en los conceptos y técnica necesaria con el objetivo de facilitar la comprensión, fiabilidad y la implementación de la seguridad. Con el término de "jerarquía" quiero referirme a que se comienza con conceptos básicos y particulares para aplicarlos y ocuparlos en los siguientes capítulos, sin dejar de aprender nuevos conceptos y términos conforme se avanza en los temas de la tesis.

Este análisis detallado de seguridad es lo necesario para algunos desarrolladores, mientras que otros, después de conocer y aplicar conceptos de seguridad, querrán saber más. Esta tesis no es la última palabra en seguridad; por el contrario, es el primer paso para realizar estrategias al momento de diseñar y programar con seguridad.

Con base en los procesos y estrategias de análisis, desarrollo y evaluación de las estructuras, métodos y sistemas que ofrecen los programas de Microsoft Visual Studio .NET, se propone mostrar técnicas, herramientas y mecanismos que conformen estrategias de seguridad para lograr mayores niveles de seguridad en sus usos y aplicaciones.

Para lograr este fin se estableció el siguiente orden en los temas de tesis:

En el capítulo 2 se presenta la "**PLATAFORMA MICROSOFT .NET**" donde se explican conceptos y términos que fueron de gran utilidad para comprender cómo funciona esta tecnología como base de la programación propuesta, en ella se habló acerca de los componentes de Microsoft .NET, y su aplicación de una forma muy general. Solo explicando los conceptos que considero son los más importantes para comprender los temas siguientes.

Una vez que se presentaron los componentes de esta plataforma fue el momento de aprender, dentro del capítulo 3, los "**PRINCIPALES CONCEPTOS DE SEGURIDAD EN .NET"**. Con este tema pretendí ser amplio y explicativo en muchos de los conceptos que involucran la seguridad en esta plataforma, debo entenderse que este capítulo presenta el concepto, no la herramienta, puesto que estas se trataron más adelante, en capitulo separado.

Hasta ese momento presenté la plataforma .NET de manera general, por lo que en el capítulo 4, comienzo a aterrizar en el leguaje de programación que se utiliza para ejemplificar y facilitar la comprensión de los conceptos y las necesidades de seguridad. En él se presentaron algunas

"**CARACTERÍSTICAS Y ELEMENTOS DEL ENTORNO DE DESARROLLO CON VISUAL BASIC .NET",** quiero aclarar que esta tesis no es un manual para aprender la programación de VB.NET, es una propuesta que recopila, descubre y muestra mucha de la información que existe acerca de la programación segura apoyándose de una herramienta como lo es VB.NET

Al llegar al capítulo 5 de este trabajo conocí conceptos, términos, algunas herramientas, métodos e incluso estrategias "implícitas" de programación que me permitió tener un panorama amplio acerca de la plataforma y del lenguaje en que se propone aprender la programación segura para las aplicaciones, por lo que puedo decir que se ha planteado el escenario, de la obra. Como en toda obra, ya sea teatral o literaria, una vez establecido el escenario debe comenzar a construirse la "trama". Los "**PROBLEMAS DE SEGURIDAD PARA PROGRAMADORES DE VISUAL BASIC .NET"** resultan ser una trama perfecta para la obra, puesto que se deben comprender los errores que se comenten al desarrollar aplicaciones y las habilidades que otros pueden llegar a tener para afectarlas, ya sea por la falta de conciencia o el exceso de conocimientos (mal aplicados) de seguridad informática, respectivamente. En la trama de la tesis, en este capítulo, se comienzan a plantear cuestionamientos que más adelante nos permitieron resolverlos, pero otros a lo largo de los capítulos restantes y otros más que aun cuando se resuelvan al momento se complementaran con herramientas o técnicas en temas y trabajos posteriores, fuera de este trabajo de tesis.

Continuando con el orden de una obra literaria, se presenta el "clímax" en el capítulo 6 con "**ESTRATEGIAS PARA ROBUSTECER LA SEGURIDAD EN APLICACIONES *VB.NET",*** este es el plato fuerte de la tesis, donde se define una estrategia de seguridad cuando se programan aplicaciones contemplando conceptos y técnicas de seguridad. Es necesario definir en este momento que en las estrategias dependerá de cada programador como las construya y como las utilice. A lo largo de la tesis se definen estrategias de una forma "implícita" y en este capítulo en particular se presentan de una forma "explícita". La importancia de construir una estrategia de seguridad permite establecer las necesidades y habilidades con las que se cuenta al momento de programar y responder los sistemas con una conciencia de seguridad informática. Dichas necesidades y habilidades exigirán mecanismos, herramientas y estructuras de programación que permitan un diseño completo de una estrategia de seguridad. Quiero también dejar en claro que este trabajo es para programadores pero no se descartan análisis físicos y humanos propios de un administrador de sistemas o servidores.

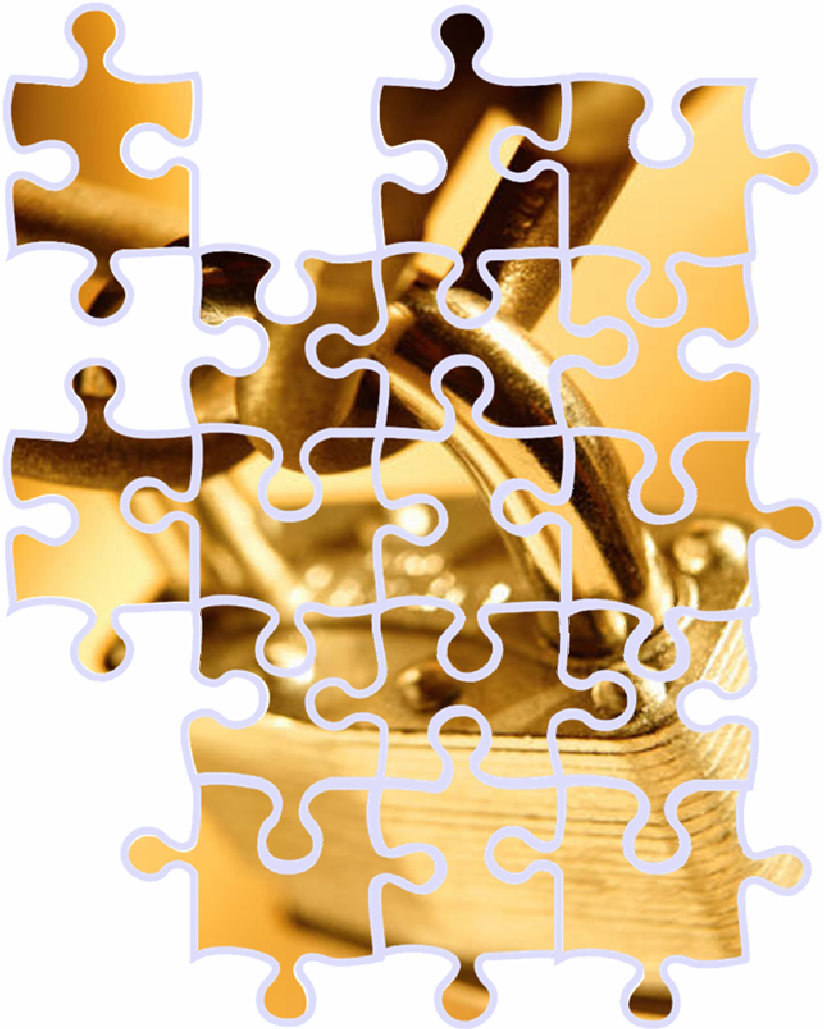
Por último en el capítulo 7 se presentan a manera de "desenlace" las "**HERRAMIENTAS Y TÉCNICAS PARA ASEGURAR PROGRAMAS Y SISTEMAS DE *VISUAL BASIC .NET"*** y evidentemente se servirán guiar para conocerlas y aplicarlas dentro de los sistemas o aplicaciones con que contemplan estrategia de seguridad. En este ultimo capítulo se describe instrucciones de control de errores de Visual Basic .NET

OBJETIVOS

- ☑ Revisar las herramientas y sistemas de trabajo que ofrece Microsoft Visual Studio .NET como parte de los procesos de modernización y actualización de los paquetes de programación.
- ☑ Analizar las herramientas de control sobre seguridad que ofrece Microsoft Visual Studio .NET.
- ☑ Explicar los principales conceptos de seguridad informática existentes en .NET.
- ☑ Describir la estructura básica de un proyecto realizado con Visual Basic .NET y el entorno de desarrollo.
- ☑ Identificar las principales problemáticas de seguridad que deben solucionar los programadores cuando desarrollan aplicaciones en Visual Basic .NET.
- ☑ Diseñar e instrumentar mecanismos, estructuras y herramientas idóneas, a partir de un conjunto de reglas, que permitan robustecer la seguridad en aplicaciones con Visual Basic .NET a partir de una estrategia de seguridad.
- ☑ Proponer herramientas que permitan robustecer la seguridad de las aplicaciones en Visual Basic .NET.

Capítulo 2

Plataforma Microsoft .NET



2. PLATAFORMA MICROSOFT .NET

El mundo de la comunicación ha cambiado rápidamente en los últimos años debido a que se ha convertido en una necesidad, no un lujo, el poder mantenerse en contacto con clientes, compañeros y colegas e incluso cuando se viaja. Son cambios significativos en materia de comunicación como en los últimos años se ha presentado el uso del Internet.

La tecnología también avanza rápidamente, ahora resulta de un menor costo el hardware de cómputo y la conectividad a Internet. Y a la vez nos permiten tener la información al alcance, se presentan: vía los teléfonos celulares, los Asistentes digitales personales (*PDA*s) y las *laptops* que se han vuelto la regla en vez de la excepción.

Microsoft .NET es una plataforma que proporciona las herramientas y tecnologías necesaria para construir aplicaciones *Web* distribuidas. Estas aplicaciones pueden comunicarse con una amplia gama de clientes sofisticados, como teléfonos celulares y PCs palm. Al mismo tiempo, la plataforma .NET permite una integración sin precedente entre lenguajes de programación, así como una variedad de servicios de tiempo de ejecución.

La naturaleza distribuida de la plataforma obliga a hacer un especial esfuerzo en lo que a seguridad y confianza se refiere. Todo el código y los servicios se desarrollan bajo unas características apropiadas de seguridad y confiabilidad.

La plataforma *.NET* soporta completamente las tecnologías de plataforma neutral, es decir tecnologías basadas en estándares, incluyendo el *Hypertext Transfer Protocol* (*HTTP*), *Extensible Markup Language* [1] (*XML*), y el *SOAP* debido a que las aplicaciones *Web* de hoy tienen interfaces interactivas que se han construido utilizando *HTML* dinámico (*DHTML*) y la tecnología *ActiveX*.

.NET está diseñado a partir de cero para que funcione bien con Internet al incorporar los estándares de Internet, incluyendo los servicios *Web XML*. De manera que los programas puedan obtener datos fácilmente a partir de diferentes sitios y utilizar *XML* de manera amplia para facilitar el intercambio de datos entre diferentes sistemas de cómputo. *.NET* también nos permite el acceder a los datos, y nos facilita la escritura de aplicaciones que se ejecutan bien en una gran variedad de dispositivos sin tener que reescribir la aplicación para cada uno. El Tiempo de ejecución de *.NET* soporta una instalación simplificada y confiable, así como se actualiza para facilitar el uso y dar una mayor confiabilidad.

Las organizaciones virtuales dependen de la integración. Lo que necesitan son aplicaciones que puedan interactuar con otras aplicaciones. Las aplicaciones deben exponer una interfaz programable que interactúe con otros servicios en Internet.

---

[1] *XML* es un estándar en la industria altamente soportado definido por *World Wide Web Consortium*, la misma organización que creó los estándares para el explorador *Web*. Fue desarrollada con una entrada amplia de *Microsoft*, pero no es propiedad de la tecnología *Microsoft*. Así *XML* proporciona los medios para separar los datos reales de la vista de presentación de esos datos. Es una clave para la Siguiente Generación de Internet y ofrece una manera de desbloquear información de mofo que se pueda organizar, programar y editar, una manera de distribuir datos en formas más útiles a una variedad de dispositivos digitales, y permite a los sitios *Web* colaborar y proporcionar una constelación de servicios *Web* que podrán interactuar con cada uno.

2.1. INTRODUCCIÓN A LA PLATAFORMA .NET

En 1998 un equipo de trabajo de *Microsoft* comenzó a trabajar en un proyecto que denominaron *Next Generation Windows Services* (*NGWS*). Este equipo se fusionó con el grupo encargado de desarrollar la versión 7 del *Visual Studio* con el fin de desarrollar un entorno de ejecución común para todos los lenguajes incluidos en él de forma que permitiese a las empresas crear lenguajes adaptados al entorno. Finalmente, en el 2000 *Microsoft* dio a conocer todo este trabajo que denominaron *Microsoft .NET*.

La plataforma *.NET* es un conjunto de tecnologías dispersas, que en muchos casos ya existían, que *Microsoft* ha integrado en una plataforma común con el objetivo de facilitar el desarrollo de este nuevo tipo de servicios de tercera[2] generación.

La plataforma *.NET* representa la visión de *Microsoft* del software como un servicio, diseñada con Internet en mente. La plataforma *.NET* cubre todas las capas del desarrollo de *software*, existiendo una integración entre las tecnologías de presentación, de componentes y de acceso a datos. *.NET* intenta establecer cierto orden sobre el caos existente sobre la anterior plataforma para el desarrollo de aplicaciones distribuidas, denominada Windows DNA (*Distributed Network Applications*), la cual se basa en un modelo de tres capas, con *ASP* en la capa de presentación, *COM* en la capa de objetos de negocio y *ADO* en la capa de datos; dicha plataforma tenía como problemas principales en el desarrollo con *CO* dado que resultaba complejo y poseía una integración con *ASP* un artificial, junto con un despliegue de las aplicaciones DNA bastante problemático.

La plataforma *.NET* no es un sistema operativo, al menos por el momento, si bien está bastante integrada con éste, y hace uso de los servicios que le proporciona. *Microsoft .NET* es una plataforma para construir, ejecutar y experimentar la tercera generación de aplicaciones distribuidas, que consiste en los siguientes *elementos*:

- ☑ Un modelo de programación basado en *XML*.
- ☑ Un conjunto de servicios *Web XML*, como *Microsoft .NET My Services* para facilitar a los desarrolladores integrar estos servicios.
- ☑ Un conjunto de servidores que permiten ejecutar estos servicios.
- ☑ *Software* en el cliente para poder utilizar estos servicios (como *Windows XP*, agendas electrónicas, etc.)
- ☑ Herramientas para el desarrollo como *Visual Studio.NET*.

Una parte importante de esta plataforma es el *software* de los dispositivos clientes y servidores, que ha sido el mercado habitual de *Microsoft*. Para los dispositivos clientes, *Microsoft* integra *.NET* en cualquier dispositivo imaginable, como *PCs* con *Windows*, agendas electrónicas con *Pocket PC*,

[2] El principal cambio que supone esta tercera generación de Internet es que se pasa a hablar de servicios en vez de aplicaciones. El objetivo es por tanto la de proporcionar servicios que resuelvan problemas. Esto servicios los pueden utilizar personas directamente o bien otros sistemas, que a su vez pueden proporcionar sus servicios.

teléfonos móviles, su consola de videojuegos *X-Box*, en *WebTV*, etc. Esto supone para las empresas aumentar el número de potenciales clientes que puedan utilizar sus servicios.

En la figura 2.1 se muestran los elementos que pueden componer la plataforma .NET.

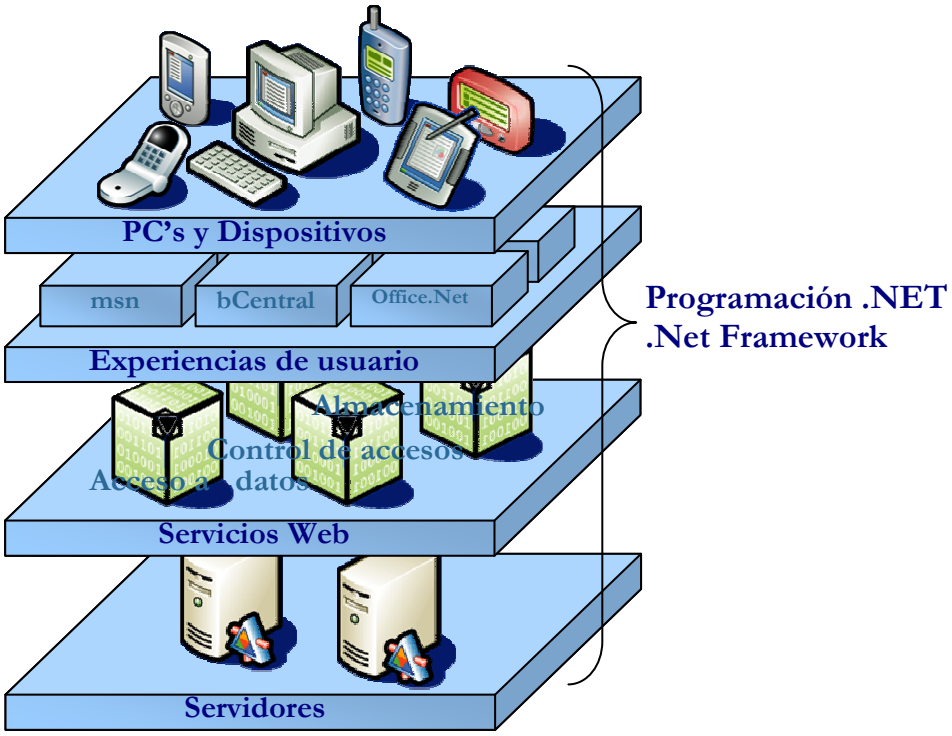


Figura 2.1. Elementos que pueden componer la plataforma

Para poder ejecutar estos servicios, Microsoft introduce una serie de software englobado dentro de los servidores, como es el *Application Center*, *Commerce Server*, etc. Estos servicios ofrecen al cliente, lo que *Microsoft* ha denominado Experiencias de Usuarios. Así, *Microsoft* ha pensado que *MSN* como canal para clientes domésticos y *bCentral* es el canal de comercio electrónico para empresas.

Como mencione anteriormente, la plataforma está orientada a Internet y los entornos distribuidos. Por eso hace especial hincapié en la distribución de código en diferentes ubicaciones y a la comunicación de estas partes de código mediante objetos remotos o servicios *Web* entre otras tecnologías.

Los *elementos* anteriormente mencionados es posible simplificarlos en la construcción desde cero sobre una arquitectura abierta de Microsoft .NET, constituida con las siguientes *tecnologías* centrales:

- ☑ *.NET Framework*
- ☑ *.NET Enterprise Servers*
- ☑ Servicios de bloques de construcción.

Microsoft ***Visual Studio .NET*** proporciona un ambiente de desarrollo de alto nivel para crear aplicaciones en *.NET Framework*. Proporciona tecnologías clave habilitadoras para simplificar la creación, implementación y evolución continua de aplicaciones seguras, escalables y altamente disponibles.

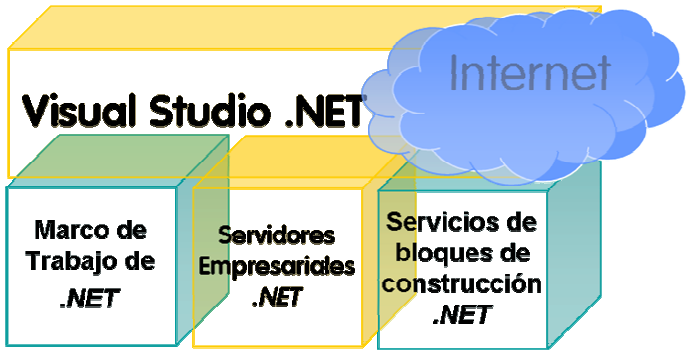


Figura 2.2. Tecnologías de la Plataforma .NET

Estas aplicaciones se pueden construir en varias plataformas, incluyendo *.NET Enterprise Servers*, y pueden utilizar servicios de bloques de construcción disponibles de *.NET*, como se muestra en la Figura 2.2.

El **Marco de Trabajo de *.NET (.NET Framework)*** se basa en el Tiempo de ejecución del lenguaje común (*common language runtime*). El *CLR* proporciona un sistema común de los servicios para los proyectos construido en el *Microsoft Visual Studio .NET*, sin importar el lenguaje. Estos servicios proporcionan llaves de construcción de bloques para aplicaciones de cualquier tipo, a través de todas las líneas de la aplicación. *Microsoft Visual Basic, Microsoft Visual C++*, y otros lenguajes de programación de *Microsoft* han tomado ventaja de estos servicios. *Microsoft Visual J# .NET* se ha desarrollado para los programadores del Leguaje *Java* que desean construir usos y servicios usando *.NET Framework*. El *.NET Framework* es explicado con mayor detalle más adelante en este capitulo.

Los **Servicios de bloques de construcción *.NET (.NET Building Block Services)***, son un sistema para usuarios enfocados a servicios *Web XML*. Con *.NET My Services,* como también se le conoce, los usuarios reciben la información relevante conforme la necesitan, entregado a los dispositivos que están utilizando, y basados en preferencias que hayan establecido. Usando el Servicio de bloques de construcción, las aplicaciones pueden comunicarse directamente usando *SOAP* y *XML* desde cualquier plataforma que soporte *SOAP*. Los servicios de bloque de construcción *.NET* incluyen por ejemplo:

- ☑ Autentificación de .NET *Microsoft Passport* y *Hailstorm*
- ☑ La habilidad para mantener el almacenamiento de documentos, información personal, salvar aplicaciones, grabar sitios *Web* favoritos, y configurar dispositivos propios.

Los **Servidores Empresariales *.NET*** (***.NET Enterprise Servers***) proporcionan escalabilidad, confiabilidad, administración, integración dentro y a través de organizaciones para el desarrollo e implementación de sistemas *Web*.

2.1.1. OBJETIVOS DE LA PLATAFORMA .NET

La plataforma .NET fue diseñada con la intención de satisfacer los siguientes objetivos:

- *Proporcionar un modelo de programación simple y consistente*. A diferencia del modelo actual, en el cual algunas facilidades del sistema operativo son ofrecidas mediante *DLLs* y otras mediante objetos COM, todos los servicios del *framework* son proporcionados de la misma forma mediante un modelo de programación orientado a objetos. Así mismo, se ha simplificado el modelo de programación, lo que permite a desarrolladores centrarse en las cuestiones relativas a la lógica de la aplicación; eliminado la necesidad de generar ficheros IDL, gestionar el registro, tratar con *GUIDs*, etc.

- *Liberar al programador de las cuestiones de infraestructura* (aspectos no funcionales). Así, el *.NET framework* se encarga de gestionar automáticamente tales cuestiones como la gestión de la memoria, de los hilos o de los objetos remotos.

- *Proporcionar integración entre diferentes lenguajes*. Con el auge de los sistemas distribuidos, la interoperabilidad se ha convertido en una de las principales cuestiones de desarrolladores de sistemas. El problema de la interoperabilidad ha sido considerado durante muchos años, desarrollándose varios estándares y arquitecturas con diferente nivel de éxito, como algunos de los siguientes:

  - *Estándares de representación de datos*, que solucionan las cuestiones relativas al paso de tipos de datos entre distintas máquinas, tales como formatos *littleendian* y *big-endian*.

  - *Estándares arquitecturales*, como RPC, CORBA o COM, que solucionan cuestiones relativas a la llamada de métodos entre diferentes lenguajes, procesos o máquinas.

  - *Estándares de lenguajes*, como ANSI C, que permite la distribución de código fuente entre distintos compiladores y máquinas.

  - *Entornos de ejecución*, como los proporcionados por las máquinas virtuales de *SmallTalk* y *Java*, que permiten la ejecución en diferentes máquinas físicas proporcionando un entorno de ejecución estandarizado.

  Sin embargo, ninguno de estos esquemas ha solucionado completamente los problemas asociados con un entorno distribuido, tal como el problema de la interoperabilidad entre lenguajes, entendida esta no como un modelo estandarizado de invocación, sino como un esquema que permite tratar a las clases y objetos de un lenguaje como clases y objetos del propio lenguaje. En la plataforma *.NET* esto es posible, con la posibilidad, por ejemplo, de heredar una clase de un lenguaje de otra clase en un lenguaje distinto, siendo esencial el papel de los metadatos para conseguir esta integración.

- *Proporcionar una ejecución multiplataforma*. *.NET* es diseñado para ser independiente de la plataforma sobre la cual se ejecutaran las aplicaciones, si bien actualmente sólo está disponible el *framework .NET* para las distintas plataformas Windows. Para conseguir este objetivo las aplicaciones .NET se compilan a un lenguaje intermedio denominado

Lenguaje Intermedio de *Microsoft* o *MSIL* (*Microsoft Intermediate Language*), el cual es independiente de las instrucciones del CPU.

- *Sistema de despliegue simple*. Se elimina la necesidad de tratar con el registro, con *GUIDs*, *DLL*…, de forma que la instalación de una aplicación es sencilla.
- *Mejora de la escalabilidad*. La gestión por parte del sistema de ejecución de .NET de cuestiones como la memoria permite mejorar la escalabilidad.
- *Proporcionar soporte para arquitecturas fuertemente acopladas y débilmente acopladas*. Para conseguir un buen rendimiento, escalabilidad y confiabilidad con grandes sistemas distribuidos, hay operaciones en las cuales los componentes están fuertemente acoplados. Sin embargo, también debe soportarse una comunicación débilmente acoplada orientada a mensajes, de forma que una transacción no quede interrumpida o bloqueada por cualquier dependencia en tiempo de ejecución.
- *Proporcionar un mecanismo de errores consistente*. En la actualidad, en la plataforma Windows no existe un sistema unificado para el manejo de los errores, de forma que este se realiza mediante códigos de error Win32, o mediante el lanzamiento de excepciones. En *.NET* todos los errores son manejados mediante un mecanismo de excepciones, el cual permite aislar el código de manejo de errores del resto, permitiéndose la propagación de excepciones entre distintos módulos y lenguajes.
- *Proporcionar un mecanismo de seguridad avanzado*. El aumento de la dependencia sobre el código móvil, como los *scripts Web*, la descarga de aplicaciones de Internet o los correos con binarios adjuntos, ha provocado que el modelo tradicional de seguridad basado en cuentas de usuario haya dejado, en parte, de tener sentido, pues asume que todo el código, tiene el mismo nivel de confianza. La plataforma *.NET* proporciona un modelo de seguridad basado en la evidencia, que posee un modelo de control de gran robustez, pudiendo basarse o no en quien escribió el código, que intenta hacer dicho código, donde está instalado, y quién está intentando ejecutar dicho código.

2.2. .NET ENTERPRISE SERVERS

.NET Enterprise Servers es la familia completa de aplicaciones de servidor de *Microsoft* para construir, implementar y administrar soluciones escalables e integradas, basadas en el *Web*. Diseñados en consideración al rendimiento de situaciones críticas, *.NET Enterprise Servers* proporciona la escalabilidad, confiabilidad y capacidad de administración para empresas globales habilitadas para *Web*.

Sus sistemas se construyen desde cero para proporcionar interoperabilidad al utilizar los estándares abiertos del *Web*, tales como el Lenguaje de marcación ampliado (*XML*). Al utilizar el estándar *XML*, *Microsoft* proporciona una nueva infraestructura de plataforma de desarrollo que ayuda a desarrolladores a crear e implementar aplicaciones distribuidas.

A continuación se presentan las descripciones de varios Servidores .NET en la Tabla 2.1:

SERVIDOR	DESCRIPCIÓN
Microsoft SQL Server	La base de datos *Windows* ofrece capacidades mejoradas para la toma de decisiones en todos los niveles de empresa con soluciones de negocios escalables, además de la integración con *Microsoft Office*. Incluye la funcionalidad rica de *XML*, ayuda para los estándares mundiales del consorcio de la *Web* (*WorldWideWeb Consortium - W3C*), la capacidad de manipular datos de *XML* usando *Transact SQL (T-SQL)*, flexible y de gran alcance, y acceso seguro a sus datos sobre la *Web* usando *HTTP*.
Microsoft BizTalk Server	Proporciona la integración para aplicaciones de la empresa (*EAI*), integración del *business-to-business*, y la tecnología avanzada de *BizTalk Orchestration* para construir los procesos dinámicos del negocio que atraviesan aplicaciones, plataformas, y organizaciones sobre el Internet. BizTalk Server es una plataforma que permite desarrollar y administrar la integración de las aplicaciones dentro y entre organizaciones, utilizando XML como un formato de documento común.
Microsoft Host Integration Server	Proporciona la mejor manera de utilizar Internet, Intranet, y tecnologías cliente/servidor mientras que preserva inversiones en sistemas anteriores existentes al tiempo que maximiza las inversiones en los sistemas heredados existentes.
Microsoft Exchange Enterprise Server	Construye en la tecnología del gran alcance de la mensajería y colaboración de tecnología introduciendo nuevas, variadas e importantes características aumentando la confiabilidad, escalabilidad, y el funcionamiento de su arquitectura. Otras características realzan la integración del intercambio con *Microsoft Windows*, *Microsoft Office*, e Internet.
Microsoft Application Center	Permite que las aplicaciones *Web* construidas sobre *Microsoft Windows* logren una disponibilidad de misión crítica (tiempo activo de 99.999%) a través del escalamiento de software al tiempo que reducen la complejidad operativa y los costos.
Microsoft Internet Security and Acceleration Server	Proporciona conexión segura, rápida, y manejable de Internet. *Internet Security and Acceleration Server* integra un *firewall* extensible, múltiplicapa y escalable de alto rendimiento. Internet Security and Acceleration (ISA) Server además agiliza la entrega de contenidos Web a través de una memoria caché del Web escalable y confiable.
Microsoft Commerce Server	Proporciona un *Framework* de aplicación, mecanismos sofisticados de la regeneración, y capacidades analíticas. *Commerce Server* además es un sistema completo para crear rápidamente soluciones escalables y personalizadas de empresa a consumidor (*B2C*) y de empresa a empresa (*B2B*) para el comercio electrónico en *Windows*.
Microsoft SharePoint Portal Server	Proporciona la capacidad de crear portales corporativos en *Web* con la administración del documento, búsqueda en el contenido, y características de la colaboración del equipo.
Microsoft Mobile Information Server	Integra con *Microsoft .NET Enterprise Servers* y *Microsoft Windows* para proporcionar comunicaciones e intercambio de datos seguros con los dispositivos móviles. La alta confiabilidad, escalabilidad, y funcionamiento son alcanzados usando *clustering*, réplica, carga balanceada, y entrega satisfactoria.

Microsoft Content Management Server	Ofrece completas características en los sistemas para que la distribución y la entrega sean satisfactorias, mediante el desarrollo y gerencia del sitio que permitan crear negocios con eficacia, de manera similar despliegan, y manejan Internet, el Intranet, y sitios de la Web.
Microsoft Windows 2003 Server	Windows 2003 Server se construye sobre las fortalezas de la tecnología de Microsoft Windows NT, integrando los servicios basados en estándares para directorio, aplicaciones del Web, comunicación y archivo e impresión con una alta confiabilidad, administración eficiente además del soporte para los avances más recientes en el hardware de red. El sistema operativo de servidor de siguiente generación, Windows .NET Server, proporcionará una mayor integración con .NET.

Tabla 2.1. Servidores .NET

Los siguientes servidores se agregarán a .NET en el futuro:

Microsoft Mobile Information Server: Esta aplicación de servidor amplía el alcance de *Microsoft .NET Enterprise Servers*, los datos empresariales y el contenido de la intranet hacia un nuevo ámbito del usuario móvil. Ofrece la intranet corporativa a la generación más reciente de dispositivos móviles de manera que los usuarios puedan acceder en forma segura a su correo electrónico, contactos, calendarios, tareas o cualquier aplicación de línea de negocios en la intranet, en tiempo real donde quiera que se encuentren.

Microsoft Windows .NET Server: Construido sobre la generación actual de soporte incorporado a los servidores *Windows* para *XML* y escalabilidad, *Windows .NET Server*, la siguiente generación de productos de servidor *Windows* agregará la integración de *Active Directory* y el sistema de autenticación *Microsoft Passport* además de que se incluirá con *.NET Framework*.

Supplier Accelerator: Es una parte clave de la iniciativa *E-Business Acceleration* que resuelve el problema de conectarse a múltiples mercados electrónicos. Proporciona una abstracción inteligente para administrar la publicación de catálogos y el procesamiento de pedidos a lo largo de los mercados electrónicos líderes y directamente con las aplicaciones del comprador, además de ser ampliable para poder acomodar otros canales de ventas adicionales del comercio electrónico. El *Supplier Accelerator* está construido sobre *Commerce Server 2000*, *BizTalk Server* y *SQL Server*, lo cual proporciona un fundamento poderoso para el comercio electrónico.

2.3. .NET BUILDING BLOCK SERVICES

Los **servicios de bloque de construcción** son un conjunto de servicios *Web XML* centrados en el usuario que mueven el control de los datos del usuario desde las aplicaciones hasta los usuarios. Los servicios permiten una sencillez personalizada y consistencia a través de las aplicaciones, servicios y dispositivos. *Microsoft* ha desarrollado servicios privados y seguros construidos alrededor de la identidad, notificación y almacenamiento que pueden actuar como bloques de construcción para otros servicios *Web XML* y experiencias *.NET*.

Microsoft Passport y *Hailstorm* son dos componentes centrales de la iniciativa *Microsoft .NET* que facilitan la integración de diferentes aplicaciones.

MICROSOFT PASSPORT: *Passport* es un componente central de *Microsoft .NET*, el cual permite desarrollar y ofrecer servicios *Web XML* distribuidos a través de una amplia gama de aplicaciones, dispositivos y servicios complementarios, todo basado en una experiencia Internet común. El servicio de registro único de *Passport* ofrece una forma rápida y conveniente para que los consumidores se registren y realicen transacciones en forma segura. El servicio de registro único (*SSI*) de *Microsoft Passport* permite que los miembros de *Passport* utilicen un solo nombre de registro y una contraseña para todos los sitios *Web* incorporados. Debido a que los nombres de registro de *Passport* están vinculados con personas y no con computadoras, los miembros pueden acceder a los sitios *Passport* en cualquier momento y a partir de una gran variedad de dispositivos. Al implementar *Passport* en algun sitio, puede realizarse lo siguiente:

- ☑ Aumentar el tráfico al simplificar el proceso de ingreso y registro.
- ☑ Mejorar la retención de clientes al proporcionar contenido personalizado basado en datos centrales del perfil *Passport*.
- ☑ Aumentar sus ventas al simplificar el proceso de compras.
- ☑ Ofrecer registro y compras fáciles y más seguras a millones de miembros *Passport*.

HAILSTORM: es el nombre código para los servicios de bloques de construcción centrados en el usuario de *Microsoft*. Es la arquitectura y conjunto de servicios centrados en el usuario para *.NET* que proporciona información relevante a nivel personal a través del Internet a un usuario, al software que se ejecuta en nombre del usuario o a dispositivos que trabajan para el usuario. A continuación se presentan las funciones de *Hailstorm*:

- ☑ Se basa en el sistema de autenticación de usuario de *Passport*.
- ☑ Ayuda a administrar y proteger la información del usuario así como las interacciones a través de todas las aplicaciones, dispositivos y servicios.
- ☑ Proporciona servicios centrales de identificación, notificación y almacenamiento que permiten interacciones transparentes y personalización de la plataforma *.NET*.
- ☑ Pone al usuario en control de sus propios datos e información, proporcionando un nuevo nivel de facilidad de uso y personalización.
- ☑ Los servicios se acceden a través de *SOAP* (Protocolo simple de acceso a objetos) y *XML* (Lenguaje de marcación ampliado), los cuales constituyen tecnologías de acceso abierto, pueden ser invocados a partir de cualquier dispositivo conectado a la red que soporte *SOAP*, independientemente del sistema operativo o el proveedor de servicio.

2.4. .NET FRAMEWORK

.NET Framework es un modelo de programación de *Microsoft* para desarrollar, implementar y ejecutar servicios *Web XML* y todos los tipos de aplicaciones: de escritorio, para dispositivos móviles o basadas en *Web*. *Microsoft* persigue dos objetivos principales en la creación de la plataforma *.NET*. El primero mejorar el desarrollo de *Microsoft Windows* haciéndolo más orientado a objetos y simplificando el modelo de objetos subyacente. El segundo, ofrecer un marco de trabajo moderno de tercera generación para crear e implementar servicios *Web XML*.

Con *.NET Framework*, *Microsoft* mejora enormemente la productividad del programador, la facilidad de desarrollo y la ejecución de aplicaciones confiables. *.NET Framework* incorpora servicios *Web XML*, un concepto nuevo en informática. Los servicios *Web XML* integran aplicaciones y componentes poco complementados diseñados para el actual heterogéneo entorno informático mediante la comunicación con protocolos de Internet estándar como *SOAP*, *WSDL* (Lenguaje de descripción de servicios *Web*) y *UDDI* (Integración, descubrimiento y descripción universal).

.NET Framework es una biblioteca de componentes y ambiente de ejecución neutral a los lenguajes. El *.NET Framework* permite construir aplicaciones integradas y orientadas a servicios que satisfacen necesidades empresariales e institucionales actuales, aplicaciones que recopilan información e interactúan con una amplia variedad de fuentes, sin importar las plataformas o lenguajes utilizados. La arquitectura de *.NET Framework* se muestra en la figura 2.3.

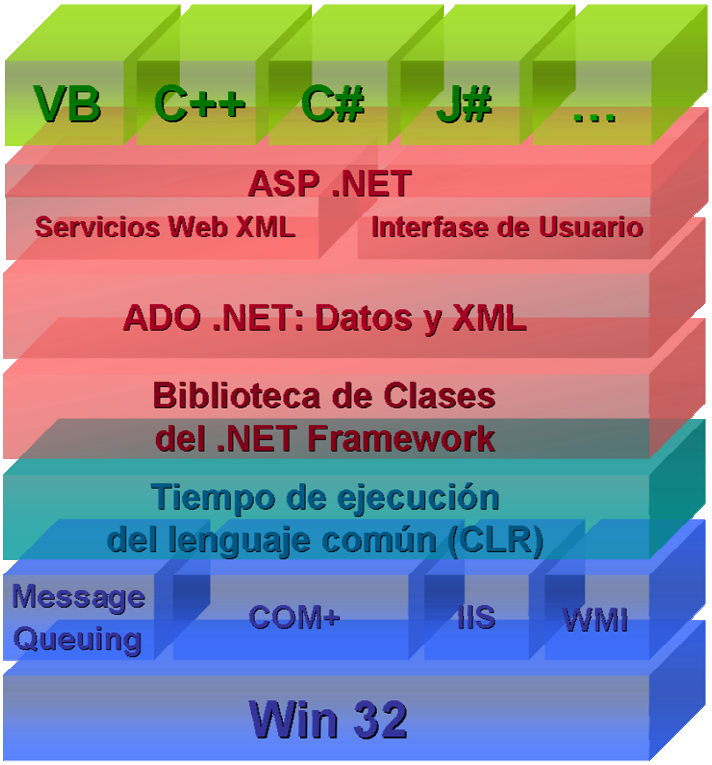


Figura 2.3. Componentes del *.NET Framework*

El *.NET Framework* debe funcionar en un sistema operativo. Actualmente, el *.NET Framework* se construye para funcionar en los sistemas operativos de *Microsoft Win32*. En el futuro, el *.NET Framework* será extendido al funcionamiento en otras plataformas, tales como *Microsoft Windows CE*.

Al funcionar en *Windows*, los servicios de aplicaciones, tales como (vea figura 2.3) *Component Services*, *Message Queuing*, *Windows Internet Information Server (IIS)*, y *Windows Management Instrumentation (WMI)*, están disponibles para el programador. El *.NET Framework* expone servicios de aplicaciones a través del *.NET Framework class library*.

El Tiempo de ejecución del lenguaje común (*CLR*) simplifica el desarrollo de la aplicación, proporciona un ambiente de ejecución robusto y seguro, soporta múltiples lenguajes y simplifica la implementación y administración de las aplicaciones. El ambiente también se conoce como un ambiente administrado, en el cual los servicios comunes, tales como la recolección de basura y la seguridad, se proporcionan automáticamente.

.NET Framework proporciona un conjunto de clases de bibliotecas (*APIs*) unificado, orientado a los objetos, jerárquico y ampliable para que lo puedan utilizar los desarrolladores. Anteriormente, los desarrolladores C++ utilizaban las Clases de fundamentos Microsoft. Los desarrolladores de Visual Basic utilizaron las clases proporcionadas por el tiempo de ejecución de Visual Basic. Mientras otros lenguajes utilizaban sus propias bibliotecas de clases y entornos. *.NET Framework* unifica los marcos dispares, así los desarrolladores ya no tienen que entender diferentes entornos para realizar su trabajo. Por el contrario, al crear un conjunto común de *APIs* a lo largo de los lenguajes de programación, *.NET Framework* permite la herencia entre lenguajes, el manejo de errores y la depuración.

ADO.NET es la siguiente generación de la tecnología de Objeto de datos *ActiveX* (*ADO*). *ADO.NET* está estrechamente integrada con *XML*, lo cual permite a los desarrolladores transferir conjuntos de datos (caché de datos dentro de la memoria) entre diferentes componentes de una solución empresarial.

ASP.NET se construye sobre las clases de programación de *.NET Framework*, por lo cual proporciona un modelo de aplicación *Web* en la forma de un conjunto de controles e infraestructura que facilitan la creación de aplicaciones *Web*. Los desarrolladores tienen acceso a un conjunto rico de controles *Web ASP.NET* que encapsulan las funciones comunes de la interfaz de Lenguaje de marcación de hipertexto (*HTML*) tales como cuadros de texto, menús desplegables, etcétera. Estos controles se ejecutan en el servidor Web y simplemente proyectan su interfaz como HTML a un explorador. En el servidor, los controles exponen un modelo de programación orientado a objetos que proporciona el poder de la programación orientada a objetos al desarrollador *Web*. *ASP.NET* también proporciona servicios de infraestructura, tales como la administración del estado de la sesión y el reciclado de los procesos, que reducen aún más la cantidad de códigos que debe escribir un desarrollador e incrementan la confiabilidad de la aplicación. *ASP.NET* también utiliza estos mismos conceptos para permitir a los desarrolladores ofrecer software como un servicio. Al utilizar los servicios Web XML, los desarrolladores pueden escribir su lógica de negocios y la infraestructura *ASP.NET* será responsable de ofrecer el servicio a través del protocolo SOAP.

Los servicios *Web* de *XML* son los componentes programables del *Web* que se pueden compartir entre aplicaciones en Internet o Intranet. El *.NET Framework* proporciona las herramientas y los servicios *Web XML* de las clases para construir, probar, y distribuir.

El *.NET Framework* soporta tres tipos de interfaces[3] para usuarios:

- ☑ Formas Web (*Web Forms*), donde el trabajo es a través de ASP.NET

[3] Ver el tema 2.5.2. Ambiente de Desarrollo, para mayor explicación sobre Web Forms y Window Forms.

- ☑ Formas de Windows (*Windows Forms*), donde la ejecución es sobre computadoras clientes de Win32
- ☑ Aplicaciones desde línea de comandos (*Console Applications*), donde, por facilidad, son usadas para la enseñanza

.NET Framework es un ambiente de desarrollo de componentes que permiten que un programa se ejecute y estén disponibles para los lenguajes de programación Visual Basic, Visual C#, Visual C++ y *JScript* de Microsoft. Los lenguajes de programación de terceros que están escritos para la plataforma .NET también tienen acceso a los mismos servicios. Más adelante en este mismo capitulo se describirá con mayor detalle los componentes del *Framework*.

2.4.1. CARACTERÍSTICAS Y BENEFICIOS DEL .NET FRAMEWORK

A continuación se presentan algunas de las características claves proporcionadas por *.NET Framework*:

- ☑ *Modelo de programación consistente*: Todos los servicios de la aplicación son accedidos a través de un modelo de programación orientado al objeto común, a diferencia de los actuales donde algunas instalaciones del sistema operativo son accedidas a través de las funciones *DLL*, y otras facilidades son accedidas a través de objetos *COM*.
- ☑ *Modelo de programación simplificada*: *.NET* busca simplificar, en gran medida, la tubería y construcciones secretas requeridas por *Win32* y *COM*. Los desarrolladores ya no tendrán que ganar un entendimiento del registro, mecanismos de control de referencia, etc... Estos conceptos ya no existen en .NET.
- ☑ *Ejecutar una vez, ejecutar siempre*: Todos los desarrolladores están familiarizados con Hell DLL. Ya que al instalar componentes para una nueva aplicación pueden sobrescribir los componentes de una aplicación anterior, la aplicación anterior puede exhibir un comportamiento extraño o detener el funcionamiento en conjunto. Ahora, la arquitectura .NET separa los componentes de la aplicación de manera que una aplicación siempre carga los componentes en los que fue desarrollada y probada. Si una aplicación se ejecuta después de la instalación, entonces la aplicación siempre deberá ejecutarse.
- ☑ *Ejecutarse en varias plataformas*: Una vez escrito y compilado para MSIL, una aplicación .NET administrada puede ejecutarse en cualquier plataforma que soporte CLR de .NET (incluyendo plataformas no basadas en Windows). De inmediato se aprecia el valor de este amplio modelo de ejecución cuando necesita soportar las configuraciones de hardware de cómputo múltiple o sistemas operativos.
- ☑ *Integración de lenguaje*: .NET permite que los lenguajes se integren unos con otros. Por ejemplo, es posible crear una clase en *C#* que se derive de una clase implementada en *Visual Basic.NET*. *.NET* puede habilitarlo debido a que define y proporciona un Sistema de Tipo Común (*CTS*) para todos los lenguajes *.NET*. La Especificación de Lenguaje común Microsoft (*CLS*) describe qué proveedores del compilador deben hacer, con el fin de que sus lenguajes se integren con otros lenguajes. *Microsoft* está proporcionando diversos compiladores que producen códigos enfocándose en *.NET*

- ☑ *Reutilización de código*: Al utilizar los mecanismos recién descritos, se pueden crear sus propias clases que ofrezcan servicios a aplicaciones de terceros. Esto por supuesto simplifica en gran medida el código de reutilización y amplía el mercado para los proveedores de componentes.

- ☑ *Administración de recursos automática a través de la colección de basura*: La programación requiere gran habilidad y disciplina, especialmente cuando se van a administrar recursos tales como archivos, memoria, conexiones de red, recursos de base de datos, etc. Uno de los tipos más comunes de *bug* ocurren cuando una aplicación se rehúsa a liberar uno de estos recursos, haciendo que la aplicación u otro falle. *CLR* de *.NET* traza automáticamente el uso de recurso, garantizando que su aplicación nunca filtrará recursos, esto se denomina colección de basura. No hay manera de liberar explícitamente un recurso de memoria, ya que la basura se colecciona automáticamente cuando ya no es necesaria.

- ☑ *Seguridad de tipo*: *CLR* de *.NET* puede verificar que todos sus códigos sean de tipo seguro. La seguridad de tipo asegura que los objetos asignados siempre podrán ser accedidos en formas compatibles. Por esto, si un parámetro de entrada de método se declara como aceptando un valor de *4-bytes*, *CLR* detecta y traza los intentos para pasar a un valor de *8-bytes* utilizando este parámetro. Similarmente, si un objeto ocupa *10 bytes* en la memoria, la aplicación no puede coaccionar esto dentro de una forma que permita leer más de *10 bytes*. La seguridad de tipo también significa que el flujo de ejecución únicamente se transferirá a ubicaciones bien conocidas (por nombre, puntos de entrada de métodos). No hay forma de construir una referencia arbitraria para una ubicación de memoria y código de causa en la ubicación para comenzar la ejecución. La seguridad de tipo y la falta de manipulación de indicador directa sienta las bases del modelo de seguridad, y también elimina muchos errores de programación común y ataques clásicos al sistema tales como la explotación de aumentos de buffer.

- ☑ *Rico soporte debugging*: Debido a que el CLR de .NET es la base común para muchos lenguajes, ahora es mucho más fácil implementar componentes en su aplicación desarrollada, utilizando el lenguaje que mejor se adapte a cada uno. *CLR* de *.NET* soporta por completo *debugging* a través de componentes escritos en múltiples lenguajes. Por ejemplo, puede pasar un código de fuente de un control escrito en *C#*, desde un programa de cliente *Visual Basic.NET*.

- ☑ *Manejo consistente de errores*: Uno de los aspectos más frustrantes de la programación de la plataforma Windows es la forma inconsistente en la que se reportan errores. Algunas funciones retornan códigos de error *Win32*, otras retornan *HRESULTS*, y algunas más originan excepciones. En *.NET*, todos los errores se reportan a través de excepciones. La ejecución normal del programa se detiene cuando ocurre una excepción, y es visto un controlador de excepción adecuado. Asimismo, las excepciones permiten aislar el código de manejo de error de la otra lógica del programa. Esto simplifica en gran medida la escritura, lectura y mantenimiento de código.

- ☑ *Implementación*: Las aplicaciones basadas en Windows pueden ser increíblemente difíciles de instalar e implementar. Por lo regular existen múltiples archivos, configuraciones de

registro y accesos rápidos que tienen que ser creados. Los componentes .NET no están referenciados en el registro, y al instalar la mayoría de las aplicaciones basadas en .NET no se requiere más que copiar los archivos al directorio. Desinstalar una aplicación *.NET* es tan sencillo como delegar estos archivos. El *CLR* permite implementaciones de lado a lado, donde, por ejemplo, se pueden tener dos versiones de la misma aplicación ejecutándose al mismo tiempo en la misma máquina mientras migra sus usuarios.

- ☑ *Seguridad*: La seguridad del sistema operativo tradicional proporciona aislamiento y control de acceso basado en las cuentas del usuario. Esto ha probado ser un modelo útil, pero se asume que todos los códigos son igualmente dignos de confianza. Esta suposición fue justificada cuando se instalaron los códigos de los medios físicos (CD-ROM) o servidores corporativos confiables. Pero con el aumento de preferencia en el código móvil, tales como *scripts* Web, las descargas de aplicación de Internet y archivos adjuntos de e-mail, existe la necesidad de un control más granular de comportamiento de la aplicación. El modelo Seguridad de Acceso de Código .NET ofrece este control.

Es posible mencionar a partir de las características algunos de los beneficios que el .NET Framework ofrece para resolver las metas por lo que fue creado:

- ☑ *Basado en estándares y practicas Web*: El *.NET Framework* soporta totalmente las tecnologías existentes de Internet, incluyendo el Lenguaje de marcación de hipertexto (*HTML*), el Lenguaje de marcación ampliado (*XML*), el Protocolo de acceso a objetos simples (*SOAP*), el Lenguaje para hojas de estilo ampliables para transformaciones (*XSLT*), *Xpath* y otros estándares del *Web*. *.NET Framework* favorece los servicios *Web XML* conectados libremente y sin estado.

- ☑ *Designado para usar con modelos de aplicación indefinidos*: La funcionalidad completa de una clase *.NET* está disponible desde cualquier lenguaje compatible en el*.NET* o el modelo de programación.

- ☑ *Fácil de usar por los programadores*: En *.NET Framework*, el código se organiza en clases dentro de espacios de nombre (*namespaces*) jerárquicos. *.NET Framework* proporciona un sistema de tipo común (*common type system*), conocido como un sistema de tipo unificado, utilizado por todos los lenguajes compatibles con *.NET*. En el sistema de tipo unificado, todos los elementos de lenguaje son objetos. No existe un tipo *Variant*, existe un tipo de cadena (*string*) común y los datos de cadena son *Unicode*.

  El *.NET Framework* permite que los desarrolladores sean más productivos ya que ahora pueden enfocarse en mejorar algoritmos y resolver problemas de negocios en lugar de en los detalles de implementación. *.NET* soporta actualmente más de 20 lenguajes de programación, incluyendo *Perl, Python, COBOL, JScript, Eiffel, Haskell, Pascal, ML, Ada, APL, C, C++, Visual Basic, C#, SmallTalk, Oberon, Scheme, Mercury, Oz y Objective Caml.* Esto ahorra el problema de aprender lenguajes, reglas o APIs nuevas. De hecho, el código escrito para *.NET* puede interoperar de manera transparente con el código existente. Por lo tanto, puede utilizar las clases de *Visual Basic .NET* desde *Visual Basic 6*. Más importante aún, puede utilizar sus objetos *COM* existentes desde *Visual Basic .NET*. Además, los componentes desarrollados en diferentes lenguajes son interoperables,

habilitando funciones como herencia entre lenguajes. Herramientas como los depuradores funcionan con todos los lenguajes soportados.

- ☑ *Clases Extensibles*: La jerarquía de las clases en *.NET Framework* no está oculta para el desarrollador. Puede acceder y ampliar las clases .NET (a menos que estén selladas) a través de procesos heredables. También puede implementar herencia entre lenguajes.
- ☑ *Más fácil de implementar, ejecutar y mantener*: .NET *Framework* reduce el costo total de propiedad de las aplicaciones, ya que las aplicaciones desarrolladas son robustas, seguras y autodescriptivas. Por ejemplo, no se requiere registro alguno para las aplicaciones. Sólo necesita copiar componentes a una carpeta en el equipo objetivo. Además, se pueden ejecutar múltiples versiones de componentes lado a lado sin afectarse entre sí.

2.4.2. COMPONENTES DE .NET FRAMEWORK

La meta de Microsoft *.NET Framework*, como mencione con anterioridad, es facilitar el desarrollo de servicios y aplicaciones *Web*, pero también tiene un efecto importante en cualquier tipo de aplicación, desde aplicaciones cliente simples hasta varios tipos de aplicaciones distribuidas, pasando por sistemas que se ejecutan en maquinas personales sin conexión a Internet o Intranet.

.NET Framework consiste en tres partes principales, con esto no quiero decir que sean los únicos: el tiempo de ejecución de lenguaje común, un conjunto jerárquico de bibliotecas de clase unificada y una versión basada en componentes de *Microsoft Active Server Pages* llamado *Microsoft ASP.NET*.

Anteriormente dentro de este capitulo he mencionado que el tiempo de ejecución de lenguaje común se desarrolla encima de los servicios del sistema operativo, además de ser el responsable de ejecutar realmente la aplicación, asegurando que se cumplan todas las dependencias de la misma, manejando la memoria, manejando la seguridad, haciendo la integración del lenguaje, etc. El tiempo de ejecución proporciona varios servicios que ayudan a simplificar el desarrollo de códigos y la implementación de aplicaciones, mejorando la confiabilidad de la aplicación

Sin embargo, el desarrollador no interactúa realmente con el tiempo de ejecución. Los desarrolladores utilizan un conjunto unificado de clases desarrolladas encima del tiempo de ejecución. Estas clases se pueden utilizar desde cualquier lenguaje de programación.

Como parte de estas bibliotecas de clase, *.NET Framework* incluye un modelo de programación de aplicaciones *Web* llamada *ASP.NET*, la cual proporciona componentes y servicios de un nivel mayor que tienen el objetivo específico de desarrollar servicios y aplicaciones *Web*.

Dado que he descrito cómo interactúan tres de los componentes más importantes del *.NET Framework* a continuación se describen los integrantes, por separado, del *.NET Framework*, para comprender la funcionalidad individual dentro del mismo.

2.4.2.1. TIEMPO DE EJECUCIÓN DEL LENGUAJE COMÚN (CLR)

El *CLR* o tiempo de ejecución del lenguaje común (*Common Language Runtime*) es un agente o cliente de la plataforma que gestiona los procesos de ejecución y los servicios de estos procesos.

El CLR es el motor de ejecución para las aplicaciones del *.NET Framework*. El *CLR* puede considerarse como el núcleo de dicho *Framework*, desempeñando el papel de una máquina virtual que se encarga de gestionar la ejecución del código y de proporcionar una serie de servicios a dicho código (el código escrito para ajustarse a los servicios del *CLR* se denomina código gestionado, mientras que el código que no utiliza el *CLR* se denomina código no gestionado).

Los tiempos de ejecución no son nuevos para la programación: Muchos otros lenguajes de programación han usado tiempos de ejecución. *Microsoft Visual Basic* siempre ha tenido un tiempo de ejecución (*VBRUN* hasta la versión 4.0 y después *MSVBVM* hasta la versión 6.0). *Microsoft Visual C++* tiene uno (*MSVCRT*), como lo tiene *Microsoft Visual FoxPro* y *Microsoft JScript* y lenguajes que no son de *Microsoft*, como *SmallTalk*, *Perl* y *Java*. El rol vital de *.NET Framework*, y lo que realmente marca la diferencia contra otros tiempos de ejecución, es que proporciona un ambiente unificado a través de todos los lenguajes de programación. El ambiente también se conoce como un ambiente administrado.

A pesar de su nombre, *CLR* en realidad tiene un papel en el desarrollo de un componente, así como en el tiempo de ejecución. Mientras que el componente se está ejecutando, el tiempo de ejecución es responsable de administrar la asignación de memoria, iniciar y eliminar cadenas y procesos, hacer respetar la política de seguridad y satisfacer cualquier dependencia que el componente pudiera tener con otros componentes. Durante el desarrollo, el rol del tiempo de ejecución cambia ligeramente. Ya que automatiza una gran parte de la funcionalidad (como administración de memoria), el tiempo de ejecución facilita la experiencia del desarrollador. En particular, el *CLR* asegura la exactitud del código y la seguridad del tipo. El CLR también reduce dramáticamente la cantidad de código que debe escribir un desarrollador para convertir la lógica en un componente reutilizable.

La figura 2.4 muestra los componentes de CLR (tiempo de ejecución del lenguaje común).

Figura 2.4 Common Language Runtime (CLR)

COMPONENTE	DESCRIPCIÓN
Cargador de clases *(Class Loador)*	Administra metadatos y la carga y diseño de las clases.
Microsoft Intermediate Language (MSIL) para compiladores nativos	Convierte MSIL a código nativo (Justo a tiempo (JIT) y Generación nativa (NGEN)).
Administrador de código *(Code Manager)*:	Administra la ejecución del código.
Recolector de basura (GC) *(Garbage Collector)*:	Proporciona administración automática del tiempo de vida de todos sus objetos. El recolector de basura está multihilado y es escalable.
Motor de seguridad *(Security Engine)*:	Proporciona seguridad basada en las evidencias, según el origen del código además de la identidad del código de invocación.
Motor de depuración *(Debug Engine)*:	Le permite depurar su aplicación y rastrear la ejecución del código.
Verificador de tipo *(Type Checker)*:	No permite transmisiones inseguras o variables no inicializadas. Se puede verificar el MSIL para garantizar la seguridad del tipo.
Administrador de excepciones *(Exception Manager)*:	Proporciona manejo estructurado de excepciones, el cual se integra con Windows *Structured Exception Handling* (SEH). Se ha mejorado el reporte de errores.
Soporte para hilos *(Thread Support)*:	Proporciona clases e interfaces que permiten la programación multihilada.
Clasificador COM *(COM Marshaler)*:	Proporciona organización desde y hacia COM.
Soporte para la Biblioteca de clase de .NET Framework *(Base Class Library Support)*:	Integra el código con CLR que soporta la Biblioteca de clases de *.NET Framework*.

Tabla 2.2. Descripción de componentes del CLR

Entre los servicios proporcionados por el *CLR* a las aplicaciones *.NET* se encuentran los siguientes:

- ☑ Gestión del código, encargándose de la carga y ejecución del código *MSIL*.
- ☑ Aislamiento de la memoria de las aplicaciones, de forma que desde el código perteneciente a un determinado proceso no pueda accederse al código o datos pertenecientes a otro proceso, lo que permite que un error en una aplicación no afecte al resto.
- ☑ Verificación de la seguridad de los tipos, garantizando la robustez del código mediante la implementación de un Sistema de Tipos Común o *CTS* (*Common Type System*).
- ☑ Conversión del código *MSIL* al código nativo, utilizándose para ello técnicas de compilación "*Just In Time*" (*JIT*).
- ☑ Acceso a los *metadatos*, que contienen información sobre los tipos, y sus dependencias, definidos en el código.
- ☑ Gestión automática de la memoria, encargándose de gestionar las referencias de los objetos y de las tareas de recolección de basura (objetos que no serán referenciados nunca más), lo que permite eliminar una de las fuentes de errores más común de las aplicaciones.

- ☑ Asegurar la seguridad en los accesos del código a los recursos, la cual estará en función del nivel de confianza del que goce el código, lo que dependerá de una serie de factores tales como su origen.
- ☑ Manejo de las excepciones, incluyendo las excepciones entre código escrito en diferentes lenguajes.
- ☑ Interoperabilidad con el código no gestionado, lo que incluye desde objetos *COM* hasta código incluido en *DLLs*.
- ☑ Soporte de servicios para los desarrolladores, tales como la depuración.

Existe un "segundo" *CLR* que el es que ejecuta los servicios *Web* y aplicaciones *ASP.NET*; este agente *CLR* es específico para funcionar sobre *Internet Information Server* (*IIS*) y hace uso de las funcionalidades adicionales y restricciones de este entorno. En este caso el agente no se comunica con el sistema operativo si no que lo hace con éste último mediante el *IIS*.

2.4.2.1.1. COMPONENTES DEL CLR

El *CLR* es el que posibilita la integración entre diferentes lenguajes, proporcionando a su vez una mejora en el rendimiento como consecuencia de los servicios que ofrece, tales como la gestión automática de la memoria. Del *CLR* es necesario hacer énfasis en los componentes principales:

- ☑ Un ***Sistema de Tipos Común*** (*CTS*), formado por un amplio conjunto de tipos y operaciones que se encuentran presentes en la mayoría de los lenguajes de programación.
- ☑ Un ***Sistema de Metadatos*** es el que permite almacenar dichos metadatos junto con los tipos a los que se refieren en tiempo de compilación, así como obtenerlos en tiempo de ejecución.
- ☑ Un ***Sistema de Ejecución***, que se encarga de ejecutar las aplicaciones del *framework* .NET, haciendo uso del sistema de información de metadatos para desarrollar los servicios tales como la gestión de la memoria.
- ☑ El ***Especificación del Lenguaje Común*** (*CLS*) es un subconjunto de especificaciones reducidas del CTS. El CLS permite a los desarrolladores de diferentes lenguajes compartir código y componentes. Para que dos elementos implementados en lenguajes distintos puedan interactuar entre sí, sus interfaces públicos deben cumplir los requisitos de tipos especificados en el CLS.

2.4.2.1.1.1. SISTEMA DE TIPOS COMÚN (CTS)

Para conseguir la interoperabilidad entre lenguajes es necesario adoptar un sistema de tipos común. Así, el sistema de tipos común (*CTS*) define como se declaran, utilizan y gestionan los tipos en el *CLR*. El *CTS* desarrolla las siguientes funciones:

- ☑ Establece un *framework* que permite la integración entre lenguajes, la seguridad de tipos, y la ejecución de código con un alto rendimiento.

- ☑ Proporciona un modelo orientado a objetos que soporta la implementación de muchos lenguajes de programación.
- ☑ Define una serie de reglas que los lenguajes deben seguir para permitir la interoperabilidad de los mismos.

Una definición de un tipo construye un nuevo tipo a partir de tipos existentes. Los tipos valor predefinido, los punteros, *arrays* y delegados son definidos al ser utilizados, por lo que a estos tipos se les conoce como tipos implícitos. La definición de un tipo incluye los siguientes elementos:

- Los atributos definidos sobre el tipo (cómo se verá en la sección de los metadatos, los atributos son un mecanismo de extensión de los mismos).
- La visibilidad del tipo. Un tipo puede ser visible a todos los ensamblados (visibilidad pública), o sólo para el ensamblado que lo define (visibilidad de ensamblado).
- Nombre del tipo. Un tipo queda definido dentro de un ensamblado, por lo que sólo tiene que ser único dentro del ensamblado.
- El tipo base del tipo definido. Un tipo definido sólo puede tener un tipo base.
- Las interfaces implementadas por el tipo.
- Las definiciones de cada uno de los miembros del tipo. Dentro de un tipo pueden definirse los siguientes miembros:
  - Eventos. Definen incidentes a los que se puede responder, así como los métodos para suscribirse/de-suscribirse de recibir la notificación del evento, y métodos para la generación del evento. Los eventos son implementados mediante delegados.
  - Campos (variables). Describen y contienen el valor de un tipo.
  - Tipos anidados. Definen a un tipo dentro del ámbito del tipo que lo contiene.
  - Métodos. Definen las operaciones disponibles para un tipo.
  - Propiedades. Nombran a un valor lógico o al estado de un tipo, y constituyen una alternativa a los tradicionales métodos de acceso/modificación *get*/*set*, de forma que internamente las propiedades son mapeadas a métodos *get* y *set*. Las propiedades pueden contener lógica interna, así como lanzar excepciones si fuera necesario. La utilización de las propiedades es idéntica a la de los campos, es decir, utilizando la notación punto Objeto.Propiedad Las propiedades son utilizadas frecuentemente para mantener la interfaz pública de un tipo independiente de la representación actual de dicho tipo.

El *CTS* se divide en dos categorías, ver Figura 2.5 generales de tipos:

- ***Tipos Valor***. Las instancias de los tipos Valor son almacenadas como la representación de su valor como una secuencia de bits en memoria. Dentro de los tipos valor se

encuentran los predefinidos (implementados por el CLR), los definidos a medida por el usuario, y las enumerados. Los tipos Valor suelen localizarse en la pila de ejecución, y tienen la propiedad de ser tipos sellados (*sealed*), es decir, no pueden ser extendidos mediante herencia. El framework soporta clases de tipos Valor:

- *Tipos valor predefinidos*. Dentro de esta clase se sitúan los tipos como los números enteros, de coma flotante, los booleanos, etc. los cuales son idénticos a los tipos primitivos utilizados por los lenguajes de programación.
- *Tipos valor definidos por el usuario*. El *framework* permite definir tipos valor propio, los cuales derivan de *System.ValueType*. Estos tipos valor, que representan un concepto que pueden poseer todos los elementos típicos en la definición de un tipo, es decir, métodos, campos, propiedades, eventos y tipos anidados.
- *Enumerados*. Un enumerado es una forma especial de un tipo Valor, el cual hereda de *System.Enum* y proporciona nombres alternativos para los valores del tipo primitivo entero (con signo o sin él). Un tipo enumerado tiene un nombre, un tipo para sus elementos, y un conjunto de campos, los cuales son campos estáticos que representan a constantes. Sin embargo, a diferencia de los tipos valor, un enumerado no puede ni definir métodos, ni implementar interfaces, ni definir propiedades o eventos.

- ***Tipos Referencia***. Las instancias de los tipos Referencia son almacenadas como referencias a la localización de su valor. Los tipos referencia son una combinación de una localización, la cual les dota de identidad, y una secuencia de bits (su valor). Dentro de los tipos referencia se encuentran los tipos interfaz, los tipos punteros, y los tipos autodescriptivos. Los tipos autodescriptivos son aquellos en los cuales es posible obtener el tipo de su valor por inspección; todos los tipos autodescriptivos heredan de la clase base *Object*, encontrándose dentro de esta categoría los *arrays* y las clases, dividiéndose esta última en clases definidas por el usuario, tipos Valor encajados (*boxed*), y los delegados. Las localizaciones, que denotan las áreas de memoria en las cuales los valores pueden ser almacenados, poseen seguridad de tipos, de forma que sólo pueden asignarse tipos compatibles. A continuación se describen los distintos tipos Referencia del CTS.
  - *Clases*: Como en cualquier sistema orientado a objetos, el *CTS* incluye el concepto de clase. Implícitamente, cualquier clase hereda de *System.Object*, la cual proporciona una serie de métodos, tales como *Equals* para determinar la igualdad, *GetType* para obtener el tipo del objeto, etc. Una clase siempre tiene como clase raíz *System.Object*, puede heredar como mucho de una única clase, y puede implementar cualquier número de interfaces.
  - *Delegados*: El *CTS* soporta un tipo de objetos denominados delegados, los cuales tienen una finalidad similar a los punteros a funciones de C++, pero con la diferencia en que estos cuentan con la seguridad del sistema de tipos, de forma que siempre apuntan a un objeto válido, no pudiendo corromper la memoria de otro objeto. Los delegados heredan de *System.Delegate*, y cuentan con una lista de invocación con los métodos que son ejecutados cuando el delegado es invocado.

Los delegados declaran una signatura de un método, de forma que dicho delegado puede referenciar a cualquier método con una signatura coincidente. Los delegados son utilizados en el manejo de eventos y en la realización de *callbacks*.

- *Arrays*: Los *arrays* son definidos especificando el tipo de sus elementos, su número de dimensiones y sus límites inferior y superior para cada dimensión. La definición de cada tipo *array*, la cual hereda de *System.Array*, es creada automáticamente por el *CLR*, de forma que no es necesaria una definición por separado.
- *Interfaces*: Un tipo interfaz es la especificación parcial de un tipo, actuando como contratos que ligan a los implementadores con lo especificado en la interfaz. Una interfaz puede contener métodos, campos estáticos, propiedades y eventos, diferenciándose de las clases y de los tipos Valor en que no puede contener campos (o variables) de instancia.
- *Punteros*: El *CTS* soporta tres tipos de punteros: punteros gestionados, punteros no gestionados, y punteros no gestionados a funciones. Los punteros gestionados son generados al pasar los argumentos por referencia en la llamada a un método. El CTS proporciona dos operaciones que respetan la seguridad de los tipos (*type safe*) para los punteros: la carga de un valor referenciado por un puntero, y la escritura de un valor a una localización referenciada por un puntero.

En la terminología del *framework .NET*, una instancia de un tipo Valor o de un tipo Referencia es conocida como valor, de forma que el valor de un tipo Valor es su representación binaria, mientras que el valor de un tipo Referencia es la localización de la representación binaria. Cada valor tiene un tipo, el cual define su representación y las operaciones que pueden ser invocadas sobre dicho tipo; sin embargo, en tiempo de ejecución no siempre es posible determinar el tipo de un valor por inspección, como es el caso de los tipos Valor y los tipos Punteros.

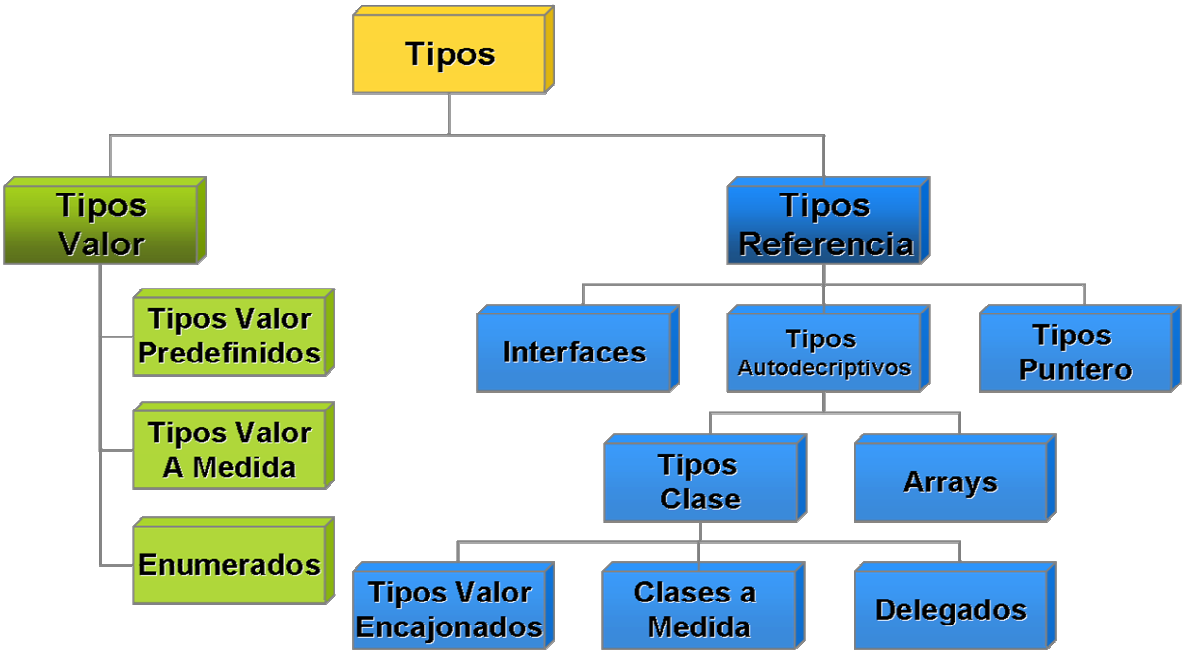


Figura 2.5 Clasificación de Tipos del CTS

Todo lenguaje compatible o integrable en la plataforma debe ser concordante y coherente con estas especificaciones. Para permitir interactuar entre los elementos sin importar el lenguaje en que se implementan, estos elementos deben mostrar sólo aquellas características que son comunes a todos los lenguajes.

2.4.2.1.1.2. SISTEMA DE METADATOS

Los metadatos son información binaria que describe los tipos implementados por un programa y se almacenan en un fichero Ejecutable Portable (PE) o en memoria, de forma que cuando un fichero con código es compilado, los metadatos son almacenados junto con el código MSIL, de forma que todos los compiladores para *.NET* están obligados a emitir metadatos sobre cada tipo contenido en un fichero fuente.

Los metadatos son la evolución de tecnologías como los ficheros *IDL*, siendo el puente que enlaza el sistema de tipos común (*CTS*) y el motor de ejecución del *.NET*. Los metadatos solucionan dos de los problemas existentes en muchos de los sistemas actuales basados en componentes, como son que la información sobre los componentes, como los ficheros *IDL*, son almacenados separados de los componentes, y que la descripción de los componentes que poseen muchos de estos sistemas sólo especifican la sintaxis de sus interfaces, y no su semántica. En *.NET* se ha solucionado este problema proporcionando un mecanismo de extensión de los metadatos, conocido como atributos, los cuales explico más adelante.

Como se ha dicho, los componentes .NET almacenan el código *MSIL* junto con los metadatos, constituyendo así unas unidades autodescriptivas denominadas ensamblados, mediante los cuales se simplifica enormemente el despliegue de las aplicaciones del *framework .NET*. Debido a su importancia, los ensamblados serán explicados más adelante en un apartado específico.

Los metadatos proporcionan los siguientes beneficios:

- Proporcionan ficheros de código autodescriptivos, eliminando la necesidad del registro y manteniéndose siempre sincronizados con las descripciones de los tipos y el código que los implementan.
- Proporcionan la información necesaria para conseguir la interoperabilidad entre distintos lenguajes.
- Proporcionan la información necesaria que requiere el sistema de ejecución para la gestión de los objetos. Así mismo, los metadatos permiten las invocaciones remotas en la plataforma .NET.
- Mediante los atributos es posible especificar una serie de aspectos que permiten especificar más en detalle como se comporta un programa en tiempo de ejecución.

Los metadatos se almacenan en una sección del fichero ejecutable portable (PE), mientras que el código MSIL es almacenado en otra sección. La porción relativa a los metadatos contiene una serie de tablas y estructuras de datos, de forma que en la porción MSIL contiene código MSIL y *tokens* que referencian a porciones de metadatos. Cada tabla mantiene la información sobre los distintos elementos de un programa, de forma que hay una tabla dedicada a las clases, otra a los

campos, etc. Cada fila de una tabla se refiere a un elemento del tipo que es descrito por dicha tabla, de tal manera que dicha fila puede referenciar a otras tablas y estructuras de datos. Los tokens situados en la porción MSIL juegan un papel similar al de los punteros, de forma que cada token apunta a una fila de una tabla.

Atributos

Un atributo es un objeto que representa datos que están asociados a elementos de un programa, tales como tipos, métodos, propiedades, etc. Los atributos son un mecanismo de extensión de los metadatos de un programa, almacenándose dichos atributos con los metadatos del elemento al cual están asociados. Los lenguajes suelen contar con instrucciones que proporcionan información declarativa, tales como los modificadores "*public*" y "*private*", los cuales proporcionan información adicional sobre los miembros de una clase (en este caso sobre la visibilidad de los mismos); sin embargo, estos tipos de información declarativa suelen estar predefinidos en el lenguaje, y no pueden ser ampliados por los usuarios del lenguaje. Los atributos constituyen un mecanismo para ampliar estos tipos de información declarativa, los cuales suelen denominarse *aspectos*.

Los *aspectos* son propiedades que afectan a la semántica o comportamiento del sistema, representando decisiones de diseño que se encuentran entremezcladas entre los aspectos funcionales de un sistema (la lógica de la aplicación) y que son difíciles de encapsular en una unidad de código. Así, ejemplos de aspectos son las características no funcionales de una aplicación, tales como la seguridad, las transacciones, la concurrencia, la persistencia, las optimizaciones en la ejecución, etc. La plataforma *.NET*, con la introducción de los atributos, proporciona soporte a una técnica de programación denominada "Programación Orientada a Aspectos", que presenta la ventaja de poder describir con mayor facilidad ciertas características de una aplicación, tales como las propiedades no funcionales. Mediante la "Programación Orientada a Aspectos", el desarrollador le indica al *CLR* las características que necesita, bien en tiempo de programación o de configuración, mediante el uso de atributos, de forma que el *CLR* interceptará las llamadas a una clase, examinará sus atributos y proporcionará las características demandadas, eliminando la necesidad de escribir código específico para soportar dichas características.

La plataforma *.NET* posee dos tipos de atributos: los intrínsecos o predefinidos, los cuales son proporcionados como parte del CLR, y los atributos definidos y creados por los usuarios, los cuales son utilizados normalmente junto con el mecanismo de reflexión. Una gran cantidad de atributos predefinidos están relacionados con la interoperabilidad con el código no gestionado, aunque también existen atributos predefinidos para otra serie de cuestiones, tales como los atributos para indicar las características de serialización de una clase.

Los atributos presentan una serie de características relativas a la aplicación de los mismos, las cuales se indican a continuación:

- *Objetivo de un atributo*. El objetivo de un atributo son los elementos del programa sobre los cuales puede aplicarse dicho atributo, siendo posible restringir los elementos sobre los cuales puede aplicarse un atributo, pudiendo indicarse que un atributo es aplicable

a cualquier elemento, sólo a un ensamblado, o a una clase, o a un método, o a un parámetro, etc., o a una combinación de los anteriores.

- *Heredabilidad de un atributo*. Es posible marcar un atributo como heredable o como no heredable, de forma que un atributo heredable que es aplicado a una clase es heredado por sus subclases, y por lo tanto, también es aplicado a dichas subclases. En el caso de tener un atributo del tipo no heredable, éste no es aplicado a las subclases de la clase que contiene al atributo.
- *Múltiples instancias de un atributo*. Es posible indicar si múltiples instancias de un mismo atributo pueden ser aplicadas o no a un mismo elemento de un programa.

2.4.2.1.1.3. SISTEMA DE EJECUCIÓN

El motor de ejecución del *CLR* es el responsable de asegurar que el código es ejecutado como requiere, proporcionando una serie de facilidades para el código MSIL como:

- ☑ **Carga del código y verificación.**
- ☑ **Gestión de las excepciones.**

El *código intermedio* MSIL generado por los compiladores del *framework .NET* es independiente del juego de instrucciones de una *CPU* específica, pudiendo ser convertido a código nativo de forma eficiente. El lenguaje *MSIL* es un lenguaje de un nivel de abstracción mucho mayor que el de la mayoría de los lenguajes máquina de las CPUs existentes, incluyendo instrucciones para trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), instrucciones para el manejo de excepciones, de tablas, etc.

La principal ventaja del *MSIL* es que proporciona una capa de abstracción del hardware, lo que facilita la ejecución multiplataforma y la integración entre lenguajes. Otra ventaja que se deriva del uso de este lenguaje intermedio es la cuestión de la seguridad relativa a la verificación del código, pues el motor de ejecución puede examinar la intención del código independientemente del lenguaje de alto nivel utilizado para generarlo. Sin embargo, dado que las *CPUs* no pueden ejecutar directamente *MSIL*, es necesario convertirlo a código nativo de la *CPU* antes de ejecutarlo.

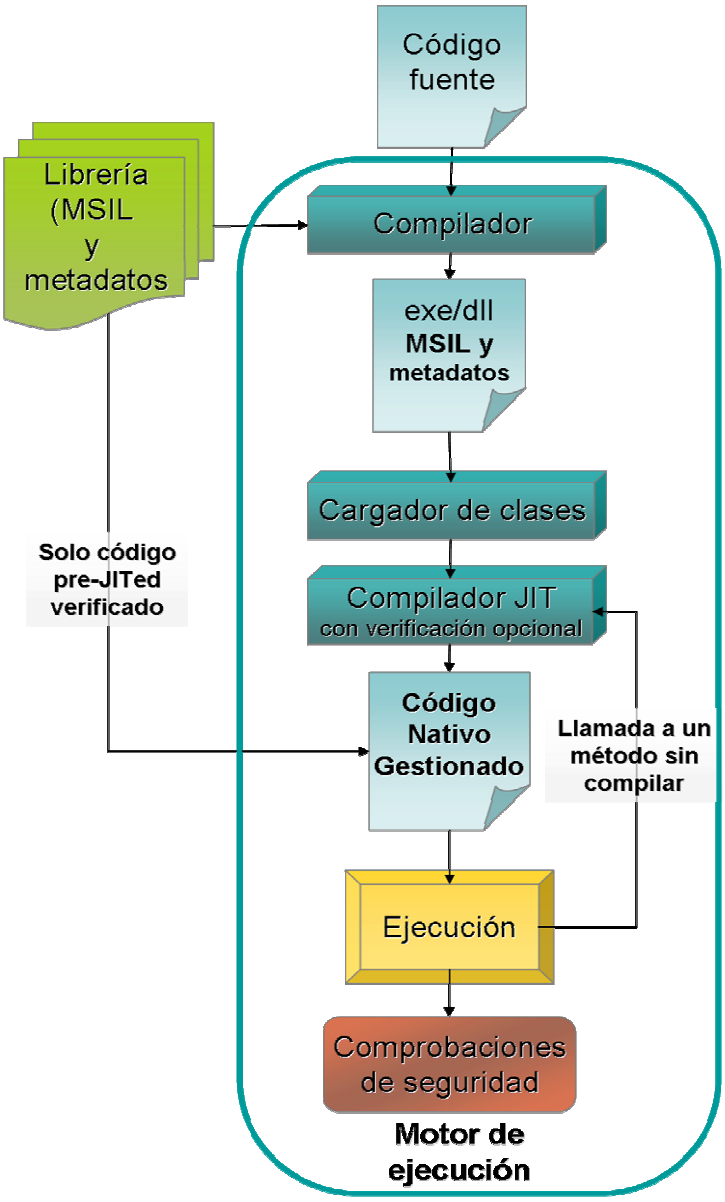


Figura 2.6 Proceso de compilación del código MSIL

☑ Compilación "Just In Time" (JIT).

La traducción de *MSIL* a código nativo de la *CPU* es realizada por un compilador "***Just In Time***" o *jitter*, que va convirtiendo dinámicamente el código MSIL a ejecutar en código nativo según sea necesario. Este proceso se muestra en la siguiente figura:

La ***compilación JIT*** tiene en cuenta el hecho de que algunas porciones de código no serán llamadas durante la ejecución, por lo que en lugar de invertir tiempo y memoria en convertir todo el código *MSIL* a código nativo, únicamente convierte el código que es necesario durante la ejecución, almacenándolo por si fuera necesario en futuras llamadas. El cargador crea y liga un *stub* a cada uno de los métodos de un tipo cuando este es cargado, de forma que en la primera llamada a un método el *stub* pasa el control al compilador *JIT*, el cual traduce el código MSIL a código nativo y modifica el *stub* para que apunte al código nativo recién traducido, de forma que las siguientes llamadas ejecutarán directamente dicho código nativo. Durante la ejecución, el código gestionado recibe del sistema de ejecución servicios como la gestión automática de la memoria, seguridad, interoperabilidad con el código no gestionado, soporte para la depuración entre lenguajes, etc.

Como parte del proceso de compilación del código *MSIL*, debe pasarse un proceso de verificación que examine el código MSIL y los metadatos, y determine si el código respeta los tipos seguro (*type safe*)

de forma que pueda tenerse la seguridad de que solo son permitidos los accesos a memoria seguros, permaneciendo los objetos aislados unos de otros. También es posible verificar si el código *MSIL* ha sido generado correctamente, pues el código MSIL incorrecto podría violar la condición del código tipo seguro.

El compilador JIT se distribuye en tres versiones:

Jitter normal: Este compilador examina el código MSIL con el objetivo de optimizar el código nativo generado. Es el que se suele utilizar por defecto.

Jitter económico: Funciona de forma similar al normal, pero no realiza ninguna optimización al compilar, limitándose a sustituir cada instrucción MSIL por la instrucción/es equivalentes en código nativo. Como consecuencia, el proceso de compilación se realiza de una forma mucho más rápida, y aún a pesar de producir un código nativo mucho menos eficiente, la ejecución sigue siendo mucho más rápida. Está pensado para ser utilizado en dispositivos empotrados que dispongan de poca potencia de CPU y poca memoria, como los dispositivos móviles.

Prejitter: Se distribuye como una herramienta mediante la cual es posible compilar completamente cualquier ensamblado y convertirlo a código nativo, de forma que posteriores ejecuciones del mismo se harán usando la versión ya compilada, ahorrándose el tiempo de realizar la compilación dinámica.

Este esquema puede producir la impresión de que introduce mucha sobrecarga, sin embargo, si bien no se ejecuta tan rápidamente como el código no gestionado, sí es mucho más eficiente que otras plataformas como Java, ya que el código no es interpretado, sino que es compilado la primera vez que es ejecutado. Un dato interesante es el hecho de que los ingenieros de Microsoft están convencidos que en el futuro el código gestionado se ejecutará más rápidamente que el no gestionado, pues cuando el jitter traduce el código MSIL a código nativo posee mucha más información del entorno de ejecución que la que posee un compilador tradicional. Así, el jitter puede detectar, por ejemplo que la máquina sobre la que se encuentra es una Pentium III y genera instrucciones especiales para dicho procesador, mientras que un compilador tradicional siempre tendrá que limitarse a una máquina concreta, que por lo general tendrá prestaciones inferiores a las máquinas existentes en el mercado, pues la opción de producir múltiples versiones de una aplicación para las distintas versiones de una misma familia de procesadores parece poco factible.

☑ **Gestión de la memoria.**

El ***Recolector de basura (GC)*** es el responsable de eliminar los objetos de la memoria *heap* que no van a ser referenciados nunca más, compactando el resto de objetos, y actualizando tras esto la referencia a la última posición de memoria libre del *heap*. El proceso de recolección de basura puede ser lanzado automáticamente por el *CLR* o por una aplicación que lo invoca explícitamente. Para averiguar qué objetos no van a ser referenciados nunca más, el recolector de basura comienza por obtener las referencias "raíces", que son aquellos objetos referenciados directamente por la aplicación. Una vez obtenidas dichas referencias "raíces", el recolector obtiene a su vez los objetos referenciados por cada referencia "raíz", y así sucesivamente hasta que obtiene el conjunto de todos los objetos válidos, tras lo cual el recolector de basura es libre de eliminar los objetos no válidos y compactar el *heap*.

Para mejorar el rendimiento del recolector de basura, éste implementa el concepto de generaciones. La idea que hay detrás de este concepto es que es más eficiente realizar la recolección de basura sobre una porción del *heap* que sobre todo el *heap*. Los objetos que han sido creados recientemente tienden a tener una mayor probabilidad de ser eliminados por el recolector que los objetos que permanecen vivos en el sistema durante algún tiempo; las

generaciones proporcionan un mecanismo para identificar a los objetos recientemente creados frente a los objetos que permanecen vivos durante cierto tiempo.

La generación de un objeto es, básicamente, un número que indica el número de veces que el objeto ha sobrevivido a un proceso de recolección de basura. Como cada vez que se realiza una recolección de basura el *heap* es compactado y la referencia a la última posición libre actualizada, los objetos pertenecientes a una misma generación permanecen agrupados juntos, de forma que los objetos de las generaciones con un número mayor (las más antiguas) permanecerán agrupados en el fondo del *heap*, mientras que los objetos de la generación 0 (la generación actual en un momento dado) permanecerán en lo alto del *heap*. Así pues, dado que los objetos de la generación 0 tienen una mayor probabilidad de "morir" antes que los objetos del resto de generaciones más antiguas, el recolector de basura puede optar por recolectar únicamente la generación o generaciones mas recientes, obteniéndose una ganancia en el rendimiento al aplicarse la recolección a sólo una parte del *heap*.

☑ **Seguridad.**

Tradicionalmente, las arquitecturas de ***seguridad*** se han basado en proporcionar aislamiento y control de acceso basándose en cuentas de usuario del sistema operativo, asumiendo que todos los programas ejecutados por un usuario poseen el mismo nivel de confianza. Alternativamente, los programas que no gozan de total confianza son ejecutados en modo restringido, de forma que son aislados del entorno sin poder acceder a la mayoría de los servicios. Ambos modelos son demasiado extremistas, por lo que *.NET* proporciona un modelo de seguridad intermedio más refinado, en el cual los programas gozan de distintos niveles de confianza en función de una serie de factores.

El sistema de seguridad de *.NET* se basa en el código gestionado, de forma que las reglas de seguridad se aseguran por el *CLR*. La mayor parte del código gestionado es verificado para asegurar la correspondencia de tipos y el comportamiento definido de otras propiedades. Este mecanismo de verificación también garantiza que el flujo de ejecución sea transferido únicamente a localizaciones bien definidas, como los puntos de entrada de los métodos, lo que elimina la posibilidad de saltar a una localización arbitraria. El proceso de verificación impide que el código que no respete los tipos sea ejecutado, y evita una serie de errores como el desbordamiento de los buffer, la lectura de una posición de memoria arbitraria, etc.

La plataforma .NET proporciona dos modelos de seguridad para las aplicaciones: la seguridad basada en la evidencia y la seguridad basada en roles, que en el capitulo 3 se estudiará a detalle.

2.4.2.1.1.4. ESPECIFICACIÓN DEL LENGUAJE COMÚN (CLS)

El CLR proporciona, mediante el sistema de tipos común CTS y los metadatos, la infraestructura necesaria para lograr la interoperabilidad entre lenguajes, pues todos los lenguajes siguen las reglas definidas en el CTS para la definición y el uso de los tipos, y los metadatos definen un mecanismo uniforme para el almacenamiento y recuperación de la información sobre dichos tipos. Pero a pesar de esto, no hay ninguna garantía de que la funcionalidad de los tipos escritos por un desarrollador en un lenguaje determinado pueda ser completamente utilizado por otros desarrolladores que utilizan otros lenguajes, pues cada lenguaje de programación utiliza los

elementos del CTS y de los metadatos que necesita para soportar su propio conjunto de características, pudiendo existir características del CLR que no son soportadas por un lenguaje concreto.

Para asegurar que el código escrito en un lenguaje sea accesible desde otros lenguajes se ha definido la Especificación del Lenguaje Común o CLS (*Common Language Specification*), que establece el conjunto mínimo de características que deben soportarse para asegurar la interoperabilidad, siendo dicho conjunto de características mínimas un subconjunto del *CTS*. El *CLS* ha sido diseñado para ser lo suficientemente grande como para que incluya las construcciones que son utilizadas comúnmente en los lenguajes, y lo suficientemente pequeño para que la mayoría de los lenguajes puedan cumplirlo.

Así, para que un objeto pueda interactuar con otros objetos, independientemente del lenguaje en el que hayan sido implementados, estos objetos deben exponer únicamente las características que están incluidas en el *CLS*. Los componentes que están adheridos a las reglas del *CLS* y utilizan sólo las características incluidas en el *CLS* se denominan como componentes conformes con *CLS* (*CLS-compliant*). Sin embargo, el concepto de conforme con *CLS* tiene un significado más específico, dependiendo de si se trata de código conforme con *CLS* o de herramientas de desarrollo conformes con *CLS*.

Para que el código de un programa sea conforme con *CLS*, este debe exponer su funcionalidad mediante características incluidas en el CLS en los siguientes lugares: Definiciones de clases públicas, de los miembros públicos de clases públicas, o de miembros accesibles mediante herencia, de los parámetros y tipos de retorno de los métodos públicos pertenecientes a clases públicas o de los métodos accesibles mediante herencia.

2.4.2.2. BIBLIOTECA DE CLASES DEL .NET FRAMEWORK

La Biblioteca de clases de *.NET Framework (Base Class Library)* es una colección de clases o tipos reutilizables que se integra estrechamente con el tiempo de ejecución de lenguaje común.

Los objetivos de diseño de la Biblioteca de clases de *.NET Framework* son:

- ☑ Permitir la creación de factores y la capacidad de ampliación.
- ☑ Implementar estándares y prácticas *Web* como fundamento.
- ☑ Unificar modelos de aplicaciones.
- ☑ Incrementar la productividad del desarrollador al proporcionar un solo modelo de programación jerárquico e intuitivo.
- ☑ Permitir la herencia y depuración entre lenguajes.
- ☑ Facilitar agregar o modificar las funciones de *.NET Framework*.
- ☑ Permitir la creación de aplicaciones seguras.

La Biblioteca de clases de *.NET Framework* proporciona una vasta funcionalidad, incluyendo:

- ☑ Administrar colecciones de objetos
- ☑ Acceso a bases de datos

- ☑ Acercarse a la pantalla
- ☑ Proporcionar seguridad y encriptación

Dada la amplitud de la *BCL*, es necesario organizar las clases que contiene en esquemas lógicos de nombrado denominados espacios de nombres (*namespaces*), los cuales son utilizados para agrupar clases con funcionalidades similares y son totalmente independientes de los ensamblados que contienen a los tipos de la librería de clases. A continuación se muestra la figura con el espacio de nombre **System,** uno de los espacios de nombres más utilizados de la librería de clases.

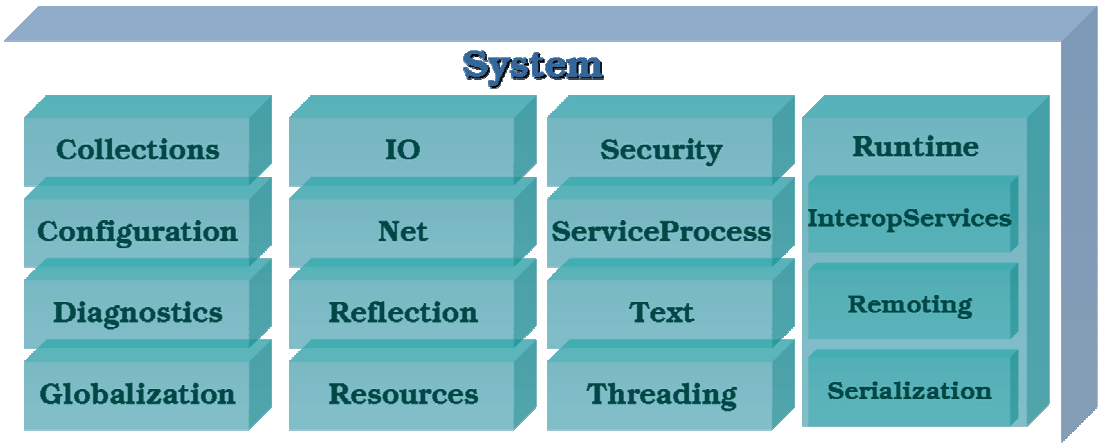


Figura 2.7 Espacio de Nombre **System**

El *namespace System* contiene clases y las bases de clases que definen tipos de datos de uso general por valor y referencia, eventos y manejo de eventos, interfaces, atributos, y el proceso fundamental de excepciones. Otras clases proporcionan servicios y apoyo en la conversión del tipo de datos, la manipulación de parámetros en métodos, matemáticas, la invocación local y remota del programa, la administración del ambiente de la aplicación, y la supervisión del control de aplicaciones, tales como:

System Tipos frecuentemente utilizados, como los tipos básicos, tablas, Excepciones, fechas, números aleatorios, entrada/salida en consola, etc.
System.Collections Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
System.Data Tipos para la manipulación de las bases de datos (ADO.NET).
System.IO Manipulación de ficheros y otros flujos de datos.
System.Net Comunicaciones en red, esto es, soporte a servicios de Transmission Control Protocol/Internet Protocol (TCP/IP) y soporte para sockets.
System.Reflection Acceso a los metadatos que acompañan a un módulo de código.
*System.Runtime.*Remoting Acceso a objetos remotos.
System.Security Acceso a las políticas de seguridad del CLR.
System.Threading Manipulación de hilos.
System.Web.UI.WebControls. Creación de interfaces de usuario basadas en ventanas para aplicaciones Web.
System.Winforms Creación de interfaces de usuario basadas en ventanas para aplicaciones estándar.
System.XML Acceso a datos en formato XML.

El *Namespace System* permite realizar:

- Aplicaciones de consola. (*System.Console*) Este espacio de nombres permite utilizar y hacer uso de los flujos de entrada, salida y error básicos de la aplicación.
- Aplicaciones basadas en formularios *Windows*. (*System.Windows.Forms*) Este espacio de nombres nos permite hacer uso de la *GUI* de *Windows*: ventanas, botones, etc.
- Aplicaciones *Web ASP.NET* y servicios *Web* (*System.Web*) Este espacio de nombres nos permite implementar aplicaciones Web basadas en formularios *ASP* y desplegar servicios *Web*.
- Acceso a fuentes de datos (*System.Data*)
- Acceso de ficheros y flujos (*System.IO*)

Además, define interfaces de funcionalidades básicas como el *IComparable* para establecer los operadores de comparación de elementos, el *IEnumerable* para hacer uso de la colección en un bucle *foreach*, *ICloneable* para soportar el clonado de objetos, entre otras acciones, pero además, clases base como *BaseCollection* sirve para implementar colecciones específicas de elementos mediante herencia.

La Biblioteca de Clases Base del framework .NET permite acceder a los servicios ofrecidos por el *CLR* y a las funcionalidades más frecuentemente utilizadas a la hora de desarrollar aplicaciones. Esta librería de clases está construida sobre la naturaleza orientada a objetos del CLR, proporciona una serie de tipos cuya funcionalidad puede ser extendida mediante la herencia. Al estar escrita la Biblioteca en *MSIL*, ésta puede ser utilizada desde cualquier lenguaje cuyo compilador genere *MSIL*, de forma que a través de las clases suministradas es posible desarrollar cualquier tipo de aplicación, desde las tradicionales aplicaciones de ventanas, consola o servicio de *Windows NT* hasta servicios *Web* y aplicaciones *ASP.NET*.

2.4.2.3. ADO.NET: DATOS Y XML

.NET Framework proporciona un conjunto de clases *ADO.NET* para manejar datos. *ADO.NET* ofrece soporte mejorado para el modelo de programación desconectado. También otorga rico soporte *XML*, como se muestra en la figura 2.8.

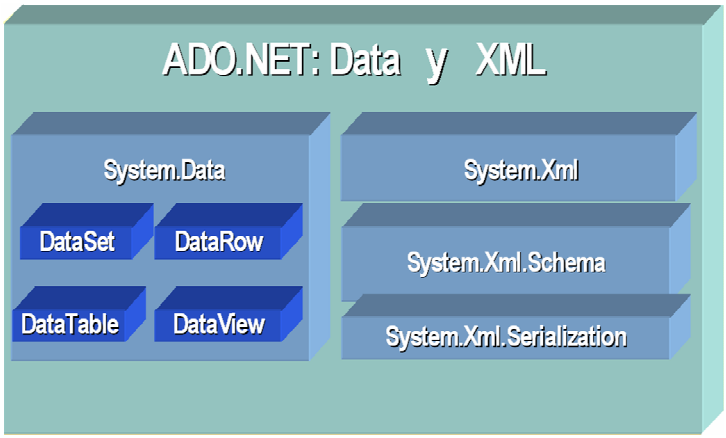


Figura 2.8. Datos y XML

Espacio de nombre ***SYSTEM.DATA***

El espacio de nombre *System.Data* consiste en clases que constituyen el modelo de objetos *ADO.NET*. En general, el modelo de objetos *ADO.NET* se divide en dos capas: la capa conectada y la capa desconectada.

El espacio de nombre *System.Data* incluye la clase *DataSet*, lo que representa múltiples tablas y sus relaciones. Estos *DataSets* son estructuras de datos completamente autocontenidas que se pueden llenar desde una amplia gama de fuentes de datos. Una fuente de datos podría ser *XML*, otra podría ser *OLEDB*, y una tercera fuente de datos podría ser el adaptador directo para *SQL Server*.

Espacio de nombre ***SYSTEM.XML***

El espacio de nombre *System.Xml* proporciona soporte para *XML*. Incluye un analizador *XML* y un escritor, ambos cumplen con *W3C*. *XSL* y la transformación de *XSL* se proporciona en el espacio de nombre *System.XML.XSL*. La construcción de *XPath* permite la navegación de gráficos de datos en *XML*. El espacio de nombre *System.XML.Serialization* proporciona toda la infraestructura básica para los servicios *Web XML*, incluyendo funciones como moverse hacia atrás y hacia delante desde los objetos hasta una representación *XML* y soporte para *SOAP*.

2.4.2.4. ASP.NET

Internet está evolucionando rápidamente a partir de los sitios *Web* actuales que sólo proporcionan páginas de interfaz a exploradores hacia una siguiente generación de sitios *Web* programables, los que vinculan directamente a organizaciones, aplicaciones, servicios y dispositivos. *ASP.NET* es un esquema de programación construido sobre *CLR* que se puede utilizar en un servidor para construir aplicaciones *Web* poderosas. Las *Web Forms* de *ASP.NET* proporcionan una manera fácil de construir interfaces *Web* dinámicas. Los servicios *Web XML* creados con *ASP.NET* proporcionan los cimientos para construir aplicaciones distribuidas basadas en el *Web*. Los servicios *Web XML* se basan en estándares abiertos de Internet, como *HTTP y XML*.

El *CLR* proporciona soporte integrado para crear y exponer servicios *Web XML* al utilizar una abstracción de programación consistente y familiar para desarrolladores tanto *Active Server Pages* (ASP) como *Visual Basic*. El modelo resultante es escalable y ampliable. Este modelo se basa en estándares abiertos de Internet (*HTTP, XML, SOAP, SDL*) para que pueda ser accedido e interpretado por cualquier equipo cliente o dispositivo habilitado para Internet.

A continuación en la figura 2.9. describo algunas de las clases *ASP.NET* más comunes.

System.Web: En el espacio de nombre *System.Web,* hay servicios de nivel inferior tales como creación de memoria caché, seguridad, configuración y otros que comparten los servicios *Web XML* y la interfaz *Web*.

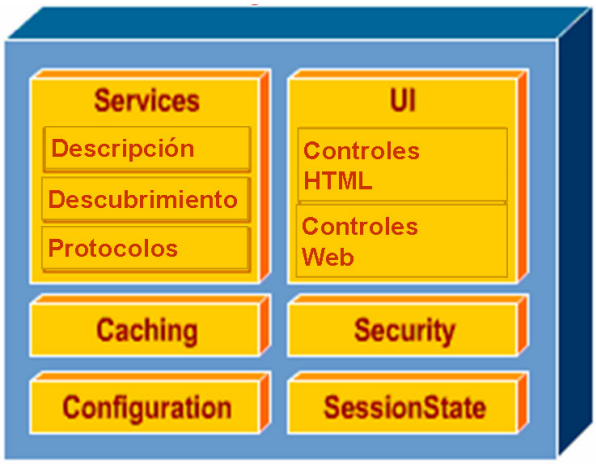


Figura 2.9. La clase *System.Web*

System.Web.Services: El espacio de nombre *System.Web.Services* incluye clases que manejan servicios *Web XML* como protocolos y descubrimiento.

System.Web.UI: El espacio de nombre *System.Web.UI* proporciona dos clases de controles: *HTMLControls* y *WebControls*. *HTMLControls* que muestran una correlación directa de las etiquetas *HTML*, como entrada. *WebControls* le permiten estructurar controles con plantillas.

2.4.3. COMPILACIÓN Y EJECUCIÓN

Las unidades base de una aplicación basada en *.NET* se conoce como ensambles o ensamblados (*Assemblies*). Un ensamblado es la unidad de versión e implementación. Por lo general cuando se compila un código fuente en un *EXE* o en un *DLL*, también se puede crear módulos, que más adelante pueden vincularse en un *EXE* o en un *DLL*. Éstas son las formas más sencillas de un ensamblado.

2.4.3.1. ENSAMBLES Y METADATOS

Los ensamblados son los bloques de construcción de las aplicaciones para la plataforma .NET, siendo la unidad fundamental de despliegue, de recuso y de control de versiones, así como la unidad sobre la que se aplican una política de seguridad. Un ensamblado es una colección de tipos y recursos que constituyen una unidad lógica de funcionalidad, proporcionando la información que el *CLR* necesita sobre las implementaciones de dichos tipos. Un Ensamblado posee las siguientes características:

- Contiene el código intermedio (*MSIL*) que será ejecutado por el *runtime*, así como los metadatos generados por el compilador y el manifiesto del ensamblado. Los ensamblados son unidades autodescriptivas, eliminándose toda dependencia con el registro de *Windows*, lo que permite simplificar el despliegue de los mismos.
- Define una frontera de encapsulación para los tipos que contiene. La identidad de un tipo queda definido, en parte, por el ensamblado al que pertenece, de forma tal que dos tipos con idéntico nombre definidos en ensamblados diferentes son considerados independientes.

- Constituye una frontera del ámbito de las referencias. El manifiesto del ensamblado contiene metadatos que son utilizados para la obtención de los tipos y recursos solicitados, especificando los tipos y recursos expuestos por el ensamblado así como los ensamblados de los cuales depende.
- Constituye la unidad mínima de versión. La política de versiones es aplicada sobre todos los tipos y recursos contenidos en el ensamblado. La política de versiones asegura que es cargado el ensamblado correcto ante la invocación de un ensamblado.
- Constituye la unidad de despliegue. Al arrancar una aplicación, solo los ensamblados que son llamados inicialmente tienen que estar presentes. El resto de ensamblados pueden ser obtenidos bajo demanda.
- Permite el aislamiento de las aplicaciones. La existencia de ensamblados privados favorecen el aislamiento de las aplicaciones, de forma que los cambios realizados en una aplicación no afecten al comportamiento del resto.
- Definen un contexto de seguridad. En la arquitectura *.NET*, las medidas de seguridad son tomadas a nivel de los ensamblados, quedando definidas mediante los metadatos del ensamblado, concretamente en su manifiesto.
- Soportan la ejecución de múltiples versiones simultáneas (*side-by-side execution*). El *runtime* tiene la capacidad de ejecutar múltiples versiones del mismo ensamblado en una única máquina, permitiendo aislar versiones incompatibles de un mismo ensamblado y simplificar la actualización de los mismos.

Los compiladores generan ensamblados que contienen *Microsoft Intermediate Language*. Sin embargo, aparte del código *MSIL*, el compilador inserta metadatos en el ensamblado[4]. Los metadatos son una colección de información que describe todos los tipos, clases, métodos, campos, eventos y demás contenidos en el ensamble. Esto es de alguna manera similar a la idea de una biblioteca de tipos. Sin embargo, a diferencia de un servidor *COM* que puede o no tener un recurso de biblioteca de tipos incrustada, un ensamble y sus metadatos son inseparables. Esto, por supuesto, significa que el ensamblado es autodescriptivo.

En muchos casos, un ensamblado podría considerarse como un solo *EXE* o *DLL*. En algunas situaciones, un solo ensamblado hace que la construcción sea mucho más sencilla, ya que todos los componentes que se requieren están agrupados. Sin embargo, un número de ensamblados se pueden vincular entre sí en un ensamblado (un ensamble de archivos múltiples). Para lograr esto puede utilizar la herramienta Vinculador de ensambles, *AL.EXE*. En algunas situaciones, por ejemplo en las aplicaciones basadas en el *Web*, el hecho de que los ensamblados

[4] En general, la estructura lógica de un ensamblado estático consta de cuatro elementos:
- ☑ El manifiesto del ensamblado, que contiene los metadatos del ensamblado.
- ☑ Los metadatos que describen los tipos del ensamblado.
- ☑ El código en lenguaje intermedio (MSIL) que implementa los tipos.

Un conjunto de recursos.

estén contenidos dentro de archivos separados puede ser una gran ventaja, ya que sólo se pueden descargar los módulos que se requieren.

2.4.3.2. EL MODELO DE EJECUCIÓN CLR

Antes de poder ejecutar el código *MSIL*, se debe convertir a instrucciones binarias nativas. La compilación normalmente la realiza por un compilador *JIT*. La figura 2.10 ilustra el proceso que se utiliza para compilar y ejecutar el código administrado, esto es, el código que utiliza el CLR.

El código fuente escrito en *Visual Basic .NET*, o en algún otro lenguaje que se dirija al *CLR*, primero se transforma en *MSIL* a través del compilador de lenguaje apropiado. Antes de ejecutarse, este código *MSIL* se agrupa en código nativo en un compilador *JIT* para cualquier procesador en el que se ejecute el código. El compilador *JIT* no simplifica todo el código en una sola vez. La función predeterminada es compilar en *JIT* cada método cuando se invoca por primera vez, pero también es posible compilar "previamente en *JIT*" el código *IL* utilizando el Generador de imagen nativa (*NGEN.EXE*). Con esta opción, todos los métodos se compilan antes de cargar la aplicación, por lo que se elimina el costo de la compilación *JIT* en cada invocación de método inicial.

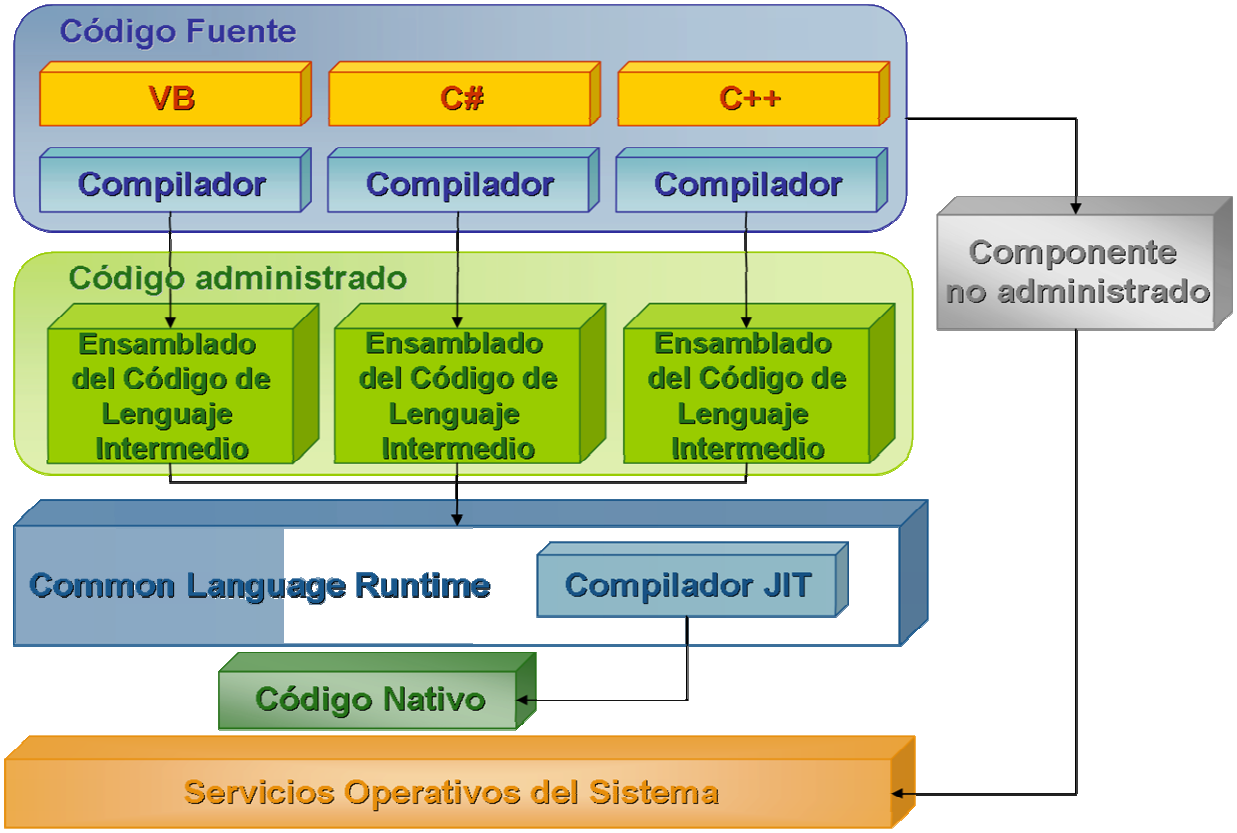


Figura 2.10. Compilar y ejecutar el código administrado

Todos los lenguajes que se dirigen al *CLR* deben exhibir prácticamente el mismo rendimiento. Mientras que algunos compiladores pueden producir mejor código *MSIL* que otros, es poco probable que surjan grandes variaciones en la velocidad de ejecución.

2.5. VISUAL STUDIO .NET

Microsoft Visual Studio .NET es un paquete de herramientas de programación multilingüe. Proporciona un ambiente completo de desarrollo para construir sobre la plataforma *Microsoft .NET*.

Al usar *Visual Studio .NET*, puede desarrollarse aplicaciones *Web* y servicios *Web XML* que funcionan en cualquier explorador y prácticamente en cualquier dispositivo. Al utilizar su lenguaje preferido, puede aprovechar sus inversiones existentes en habilidades y sistemas. El resultado es mayor productividad, desarrollo *Web* de extremo a extremo y un menor tiempo para su entrega.

Las nuevas funciones de *Visual Studio .NET* crean un ambiente de desarrollo completo para construir aplicaciones en *Microsoft .NET Framework*, la plataforma de aplicaciones *Web* de siguiente generación de *Microsoft*. Proporciona tecnologías habilitadoras clave para simplificar la creación y desarrollo de servicios *Web XML* seguros, confiables, escalables y de alta disponibilidad al tiempo que utiliza las habilidades existentes del desarrollador.

Además, *.NET Framework* proporciona funciones para ayudar a los desarrolladores *Web* a utilizar los servicios *Web XML* como si fueran objetos locales en su lenguaje preferido de desarrollo, simplificando así el desarrollo de servicios y aplicaciones. El resultado es un tiempo más corto para entrar al mercado, productividad mejorada del desarrollador y, en última instancia, un software de mayor calidad.

El uso de *XML*, un estándar abierto que administra el *World Wide Web Consortium* (W3C), elimina barreras al uso compartido de datos y a la integración de software. *XML* le facilita intercambiar datos y el *software*. *.NET* permite que los usuarios puedan trabajar con los datos una vez que se reciben.

El soporte para servicios *Web XML* está profundamente integrado en *Visual Studio .NET*. Puede exponer fácilmente cualquier método utilizando un servicio *Web XML* y consumir servicios *Web XML* como si fueran objetos *COM*. Los servicios *Web XML* pueden ejecutarse en cualquier plataforma y en cualquier hardware, de tal forma que un cliente puede exponer una aplicación de *mainframe IBM* como un servicio *Web XML* y permitir que sus desarrolladores codifiquen contra ese servicio utilizando todas las funciones claves de productividad en *Visual Studio .NET*, tales como *IntelliSense*.

En *Visual Studio .NET* puede exponerse fácilmente cualquier función, escrita en cualquier lenguaje, como un servicio *Web XML*. No hay necesidad de aprender *XML* y *SOAP* para aprovechar los servicios *Web XML*. Al compilar sus objetos de negocios, *Visual Studio .NET* generará automáticamente un archivo *XML* que describe la función, y al invocarlo, la función mandará y recibirá automáticamente paquetes *XML*.

En *Visual Studio .NET*, puede arrastrar cualquier servicio *Web XML* expuesto directamente hacia su aplicación. Esto permite a *Visual Studio* tratar el servicio *Web XML* como una clase. Llamar el servicio *Web XML* es tan fácil como crear una instancia nueva de clase de servicio *Web XML* y luego llamar sus métodos expuestos.

2.5.1. OBJETIVOS DE DISEÑO

Visual Studio .NET aumenta la productividad de desarrolladores y ayuda a enfrentar las demandas de un mercado rápidamente cambiante y competitivo. La idea fundamental detrás de *Visual Studio .NET* es simplificar el desarrollo de soluciones *Web* empresariales, poderosas y confiables, al ofrecer capacidades de desarrollo *Web* de extremo a extremo así como componentes del lado del servidor escalables y reutilizables.

Las metas de diseño para *Visual Studio .NET* son:

☑ Maximizar la productividad del desarrollador

Para que los desarrolladores maximicen su productividad, Visual Studio .NET proporciona herramientas que pueden adaptarse a un conjunto siempre cambiante de requerimientos de los individuos, los equipos y las aplicaciones. Proporciona un modelo rico para personalizar, automatizar y ampliar el ambiente de desarrollo integrado (IDE). Los desarrolladores obtienen asistencia apropiada a través de funciones como Ayuda dinámica, IntelliSense mejorado y tareas automatizadas de desarrollo de rutina.

☑ Simplificar el desarrollo basado en servidores

Una de las tareas más complejas al desarrollar aplicaciones distribuidas es construir componentes de lado del servidor que implementen la lógica de negocios de una aplicación. Visual Studio .NET facilita la construcción de componentes basados en el servidor al aprovechar los principios del Desarrollo rápido de aplicaciones (RAD) y al aplicarlos al desarrollo de componentes.

☑ Entregar herramientas de diseño poderosas

Puede aprovechar un enfoque de programación familiar para una gran variedad de interfaces, incluyendo el explorador, los dispositivos móviles y los clientes ricos de Windows. Los editores compartidos de HTML, XML y hojas de estilo facilitan el desarrollo de aplicaciones Web desde cualquier lenguaje de Visual Studio.

Web Forms son parte del nuevo *Microsoft .NET Framework* y aprovechan las tecnologías nuevas, incluyendo un marco de trabajo de aplicaciones común, un ambiente de ejecución administrado, seguridad integrada y principios de diseño orientados a objetos. Además, *Windows Forms* ofrecen soporte completo para conectarse rápida y fácilmente con los servicios *Web XML* y desarrollar aplicaciones ricas y conscientes de datos basado en el modelo de datos *ADO.NET*.

2.5.2. AMBIENTE DE DESARROLLO

Al diseñar aplicaciones que involucran una interfaz, tiene dos opciones: *Windows Forms* y *Web Forms*. Ambas tecnologías pueden proporcionar una interfaz rica y funcionalidad de aplicación avanzada para resolver problemas de negocios. Sin embargo, puede elegir dependiendo de los requerimientos de negocios.

Por ejemplo, si está creando un sitio *Web* de comercio electrónico que estará abierto al público por Internet, desarrollaría la aplicación utilizando *Web Forms*. Por otro lado, para construir una

aplicación altamente receptiva y que haga un uso intensivo de los procesos que aproveche la funcionalidad completa del equipo cliente, utilizaría *Windows Forms*.

Web Forms se utilizan para crear aplicaciones en las cuales la interfaz primaria es un explorador. Esto incluye aplicaciones que tienen la intención de estar disponibles públicamente a través del *World Wide Web*, como aplicaciones de comercio electrónico.

Modeladas a partir de las Formas de *Visual Basic*, las *Web Forms* permiten a los desarrolladores crear aplicaciones de escritorio basadas en formas para desarrollar rápidamente aplicaciones *Web* programables que operen entre plataformas y entre exploradores y que utilizan las mismas tecnologías que ya aparecen en *Visual Basic*.

Las ***Web Forms*** hacen que desarrollar aplicaciones Web sea tan fácil como construir aplicaciones de *Visual Basic* basadas en formas. Una página estándar de *Web Forms* consiste en un archivo *HTML* que contiene la representación visual de la página y un archivo fuente con código para manejo de eventos. Los desarrolladores diseñan visualmente sus aplicaciones de Web Forms y luego implementan la lógica de negocios con *Visual Basic, C++ o C#* completamente separados de la interfaz.

La fuente se compila en código ejecutable, proporcionando un rendimiento rápido en el tiempo de ejecución. El código se compila y se ejecuta en el servidor para un máximo rendimiento y escalabilidad. Así, el rendimiento de las *Web Forms* es mayor que lo logrado anteriormente con código interpretado y *ASP*. Adicionalmente, las *Web Forms* se pueden mantener mejor debido a que separan claramente la interfaz (el archivo HTML) del código (un archivo de clase).

Los desarrolladores también pueden aprovechar funciones de extremo superior en exploradores más nuevos como Internet Explorer o Netscape ó pueden reducir la funcionalidad para soportar dispositivos inalámbricos que utilicen el Protocolo de aplicaciones inalámbricas (*WAP*).

Debido a que el desarrollo del *Web* permea estrechamente a *Visual Studio .NET*, la funcionalidad que originalmente se encontraba en *Visual InterDev* ahora es parte central del ambiente mismo y es accesible desde los productos en diferentes lenguajes. Sin importar el lenguaje que se elija para el desarrollo, ahora sólo hay que aprender, configurar y usar un ambiente.

Windows Forms son parte de la biblioteca de *Microsoft .NET Framework* y aprovechan las tecnologías nuevas, incluyendo un marco de trabajo de aplicaciones común, un ambiente de ejecución administrado, seguridad integrada y principios de diseño orientados a objetos. *Windows Forms* ofrecen soporte completo para conectarse rápida y fácilmente con los servicios *Web XML* y desarrollar aplicaciones ricas y conscientes de datos basado en el modelo de datos AD*O.NET*. Con el nuevo ambiente de desarrollo compartido en *Visual Studio .NET*, los desarrolladores podrán crear aplicaciones de *Windows Forms* utilizando cualquiera de los lenguajes que soporta la plataforma .NET, incluyendo Microsoft Visual Basic y C#.

Windows Forms se utilizan para desarrollar aplicaciones que dependen del poder de la computadora de escritorio para procesamiento y mostrar contenidos de alto rendimiento. Éstas incluyen aplicaciones clásicas de escritorio *Win32*, como aplicaciones de dibujo o gráficos, sistemas de registro de datos, sistemas de punto de ventas y juegos.

Algunas aplicaciones de *Windows Forms* pueden autocontenerse totalmente y realizar todo el procesamiento de aplicaciones en la computadora del usuario. Otras pueden ser parte de un sistema más grande y utilizar principalmente la computadora de escritorio para procesar las entradas del usuario. Por ejemplo, un sistema de punto de venta con frecuencia requiere una interfaz sensible y sofisticada que se crea en la computadora de escritorio, pero que se vincula con otros componentes que realizan procesamientos *back-end*.

Debido a que una aplicación *Windows* que utiliza *Windows Forms* se desarrolla alrededor del marco de trabajo de *Windows*, cuenta con acceso a recursos del sistema en la computadora cliente, incluyendo archivos locales, el Registro de *Windows*, la impresora, etc. Este nivel de acceso puede restringirse a eliminar cualquier riesgo de seguridad o problemas potenciales que surjan de accesos no deseados. Adicionalmente, las *Windows Forms* pueden aprovechar las clases de interfaz de dispositivos gráficos (GDI+) de *.NET* para crear aplicaciones ricas en gráficos que con frecuencia se requieren para aplicaciones de *data-mining* y de juegos.

Con una aplicación de *Windows Forms* no hay necesidad de implementar una aplicación en el escritorio del usuario final. De manera alterna, un usuario puede iniciar la aplicación simplemente escribiendo un URL en un explorador. La aplicación se descargará en el equipo cliente, se ejecutará en un ambiente de ejecución seguro y se auto-quitará al terminar.

2.5.3. PRODUCTIVIDAD DE VISUAL STUDIO

Microsoft Visual Studio .NET permite a desarrolladores aumentar la productividad, permitiéndoles personalizar la apariencia del ambiente de desarrollo, mejorar y ampliar su funcionalidad, automatizar tareas repetitivas e integrar Visual Studio .NET con otras aplicaciones.

Microsoft Visual Studio .NET proporciona un ambiente de desarrollo (*IDE*) único, integrado y compartido para todos los lenguajes que incorpora. Fue diseñado para ayudar a los desarrolladores a construir sus soluciones más rápido, con menos desorden y con todas las herramientas accesibles fácilmente en cualquiera de los lenguajes en el sistema de desarrollo Visual Studio. El *IDE* de *Visual Studio .NET* tiene un gran número de funciones que proporcionan información a los desarrolladores cuándo la necesitan y cómo la necesitan.

Un ambiente visual integrado estrechamente y unificado simplifica el proceso de desarrollar aplicaciones *Web* y reduce la curva de aprendizaje. Los editores compartidos de *HTML, XML* y hojas de estilo facilitan el desarrollo de aplicaciones *Web* desde cualquier lenguaje de *Visual Studio*.

Para permitir la personalización de casi cualquier aspecto del ambiente, *Visual Studio .NET* expone un modelo de objetos completo. Con casi 200 objetos, este modelo proporciona acceso directo al editor de código, jerarquía de proyectos, modelo de código, depurador, menús y

comandos, el proceso de construcción y las diferentes ventanas de herramientas de Visual Studio que componen *Visual Studio .NET*, incluyendo una lista de tareas y caja de herramientas.

2.5.4. LENGUAJES

La plataforma *.NET* es neutral al lenguaje. El número mismo de lenguajes disponibles en la plataforma .NET es un excelente testigo de su neutralidad hacia los lenguajes y su capacidad de uso. Microsoft proporciona los siguientes lenguajes que soportan la plataforma .NET: Microsoft *Visual Basic, Microsoft Visual C++, C# y JScript*. Microsoft también proporciona las herramientas para los desarrolladores Java. Otros lenguajes que soporta la plataforma *.NET* son *Perl, Python, COBOL, Eiffel, Haskell, Pascal, ML, Ada, APL, C, SmallTalk, Oberon, Scheme, Mercury, Oz y Objective Caml*. La neutralidad de lenguaje de la plataforma *.NET* da como resultado varios beneficios para desarrolladores:

☑ Capacidad para volver a utilizar y compartir el código

En el pasado, los desarrolladores con frecuencia tenían problemas al compartir el código, ya que algunos desarrolladores en el equipo utilizaban C++ mientras que otros trabajaban en otros lenguajes, tales como *Visual Basic*. Dependiendo de la manera en que se escribiera el código C++, podía o no ser accesible para los desarrolladores de *Visual Basic*. *.NET* elimina este problema, ya que ahora cualquier lenguaje compatible con *.NET* puede invocar cualquier clase *.NET* e incluso puede extender esa clase para sus propios fines. Reutilizar el código ahora es automático.

☑ Mismo acceso de *API* para todos los lenguajes

El mismo conjunto de clases base está disponible para los programadores, independientemente del lenguaje de código que utilicen.

☑ Herencia de implementación

Una clase o componente escrito en un lenguaje se puede extender en otro.

☑ Manejo de excepciones entre lenguajes

Tradicionalmente, el modelo de manejo de errores de un lenguaje dependía ya sea de la forma exclusiva del lenguaje para detectar los errores y localizar manejadores para ellos, o del un mecanismo de manejo de errores proporcionado por el sistema operativo. En la plataforma .NET, el Tiempo de ejecución de lenguaje común (*CLR*) ayuda en gran medida en el diseño de software tolerante a errores al proporcionar una plataforma para un manejo de errores uniforme. El *CLR* maneja las excepciones para todos los lenguajes compatibles con *.NET*. Sin importar el lenguaje utilizado para generar la excepción o el que se haya invocado para manejarla.

Todos los lenguajes que soportan la plataforma .NET proporcionan funcionalidad similar. En todos los lenguajes, todos los objetos se derivan de *System.Object*. Las Propiedades, Métodos, Eventos y Atributos se soportan en todos los lenguajes.

Por predeterminación, los tipos de valor simple, enumeraciones y estructuras se pasan por valor en todos los lenguajes. De manera similar, las interfaces, clases y arreglos se pasan por referencia. Además, debido a que todos los lenguajes se compilan en el mismo *Microsoft*

Intermediate Language (*MSIL*), todos son capaces de proporcionar prácticamente el mismo nivel de rendimiento.

Hablando de un lenguaje en particular y el de interés para este proyecto puedo mencionar que existen varios cambios significativos en *Visual Basic .NET*. Por ejemplo, *Visual Basic .NET* permite escribir aplicaciones que pueden realizar varias tareas de manera independiente usando un proceso conocido como multihilado. El multihilado permite escribir aplicaciones más receptivas a la información ingresada por el usuario, ya que puede generar que tareas complicadas se ejecuten en hilos separados de su interfaz.

Se han realizado otros cambios a *Visual Basic* para asegurar que se comporte como otros lenguajes que soporta la plataforma .NET. Por ejemplo, los arreglos ahora se basan en cero en *Visual Basic*, igual que en C# y C++. *Visual Basic .NET* soporta funciones de programación orientada a objetos como herencia. Visual Basic .NET también soporta el manejo de excepciones estructurado. Además, *Visual Basic .NET* proporciona varias maneras de reducir los errores de programación.

El *.NET Framework* proporciona, como ya mencioné, la ayuda para varios lenguajes de programación. C # es el lenguaje de programación diseñado específicamente para la plataforma *.NET*, pero *Visual Basic* también se han mejorado para ser apoyado completamente por el *.NET Framework*.

A continuación en la Tabla 2.3 se listan, con una breve descripción, los lenguajes disponibles en la plataforma *.NET*.

LANGUAGE	DESCRIPTION
C#	C # fue diseñada para la plataforma .NET y es el primer lenguaje moderno en la familia de C y de C++. Puede ser embebida en páginas ASP.NET. Algunas de las características importantes de este lenguaje es que incluyen clases, interfaces, delegados, boxing y unboxing, namespaces, propiedades, índices, eventos, operadores sobrecargados, versioning, atributos, código inseguro, y generación de documentación con XML. No son necesarios headers o Interface Definition Language (IDL).
Managed Extensions to C++	El control de C++ es una extensión mínima del lenguaje de C++. Esta extensión proporciona el acceso al .NET Framework que incluye la colección de basura, herencia de fácil implementación y múltiples interfaces. Esta mejora también elimina la necesidad de escribir el código de tubería para los componentes. Ofrece el acceso a código de bajo nivel cuando sea útil.
Visual Basic .NET	Visual Basic .NET proporciona excedentes innovaciones en comparación de versiones anteriores del lenguaje Visual Basic .NET soporta herencia, constructores, polimorfismo, excepciones estructuradas, sobrecarga del constructor, estricta revisión de tipos, libre uso de hilos, y muchas otras características. Hay solamente una forma de asignar, no permitir (let) o fijar métodos. Nuevo desarrollo de aplicaciones rápidas (RAD), tales como XML Designer, Server Explorer, y Web Forms designer, están disponibles para Visual Basic en Visual Studio .NE. Con este lanzamiento, la edición de Visual Basic Scripting provee funcionalidad completa para VB.

JScript .NET	JScript .NET es reescrito para ser completamente consistente con .NET. Incluye soporte para clases, herencia, tipos, compilación, y proporciona características mejoradas del funcionamiento y de la productividad. JScript .NET también se integra con Visual Studio .NET. Es posible aprovechar cualquier clase del .NET Framework en el .NET Framework.
Visual J# .NET	Visual J# .NET es una herramienta de desarrollo para los programadores del lenguaje Java que desean construir aplicaciones y servicios en el .NET Framework Visual J# .NET está completamente constituido por .NET e incluye las herramientas para actualizar automáticamente y convertir y ajustar proyectos existente de Visual J++ 6.0 al nuevo formato de Visual Studio .NET. Visual J# .NET es parte de la estrategia *Java User Migration Path to Microsoft .NET* (JUMP to .NET).
Third-party languages	Varios lenguajes de tercera persona están apoyando en la plataforma .NET. Estos lenguajes incluyen el APL, COBOL, PASCAL, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, y Smalltalk.

Tabla 2.3. Lenguaje de la plataforma .NET

2.6. CONCLUSIÓN

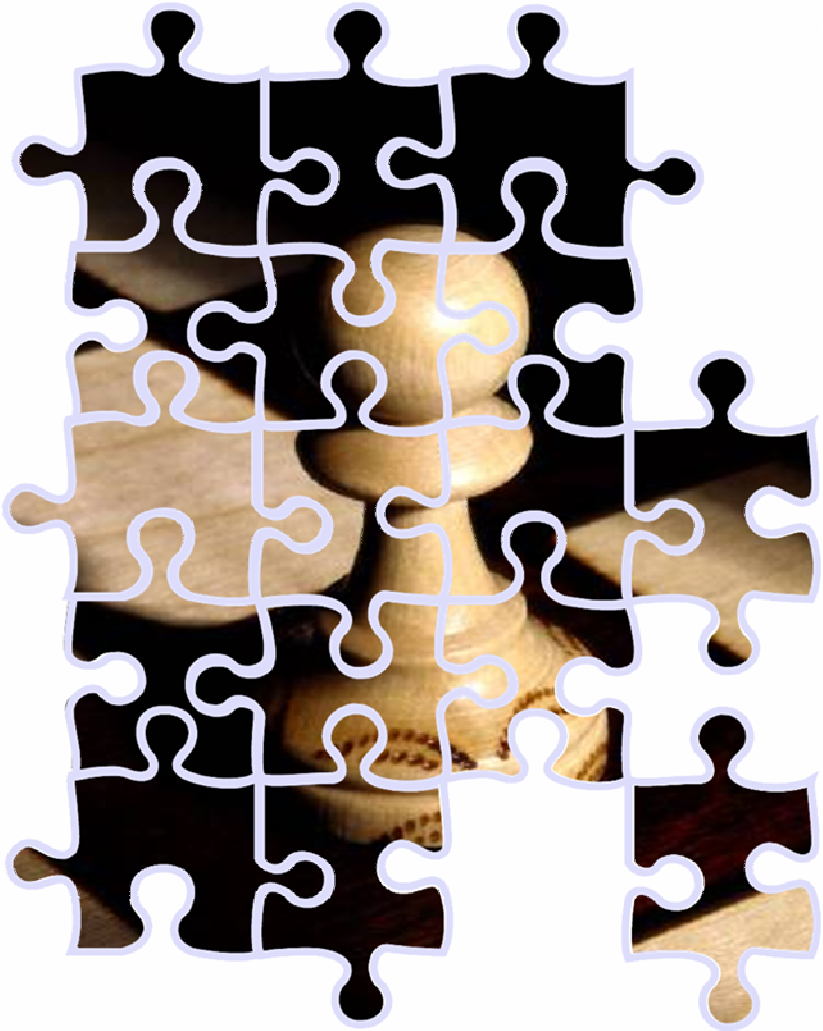
En este capítulo es posible aprender que la plataforma .NET es neutral a los lenguajes. Es posible aprender acerca de los beneficios de .NET Frameworky de sus componentes.

Microsoft .NET controla la mayoría de las deficiencias de computación actuales para realizar la visión de la habilitación de acceso a todos los datos y aplicaciones del usuario en cualquier parte y desde cualquier dispositivo. Los usuarios pueden interactuar con sus datos a través de manuscrita, voz y tecnologías de visión. Los datos viven de manera "segura" en Internet o Intranets por lo que pueden acceder desde las PCs en el hogar o en la oficina, desde sus teléfonos celulares o radiolocalizadores, desde sus PDAs o Pocket PCs. Las aplicaciones se pueden adoptar a la funcionalidad que ofrecen para las limitaciones y oportunidades presentadas por el dispositivo con el que interactúa el usuario. Las aplicaciones pueden actuar en nombre del usuario utilizando un conjunto predefinido de preferencias y directrices. Con esto, los negocios se benefician del radical aumento en la eficacia y productividad a medida que .NET proporciona a los empleados, clientes, datos y aplicaciones de negocios una forma coherente e inteligente de interactuar de manera global.

La visión de Microsoft para la nueva generación de computación distribuida en Internet es donde el software se ofrece como servicio, es accesible por cualquier dispositivo en todo momento, en cualquier lugar, y es completamente programable y personalizado. Para permitir esta visión, Microsoft está ofreciendo una plataforma .NET y las experiencias del usuario .NET, que se integran en los estándares y protocolos de Internet, con herramientas y servicios que integran la computación y comunicaciones en nuevas formas productivas. La plataforma Microsoft .NET está explícitamente diseñada para permitir el rápido desarrollo, integración e instrumentación de cualquier grupo de servicios y aplicaciones en una sola solución, y representa la evolución de la arquitectura Windows DNA.

Capítulo 3

Principales conceptos de seguridad existentes en .NET



3. PRINCIPALES CONCEPTOS DE SEGURIDAD EN .NET

Tradicionalmente las arquitecturas de seguridad se han basado en proporcionar aislamiento y control de acceso a partir de las cuentas de usuario del sistema operativo, asumiendo que todos los programas ejecutados por un usuario poseen el mismo nivel de confiabilidad. Alternativamente, los programas que no gozan de total confianza son ejecutados de manera restringida, de forma que son aislados del entorno sin poder acceder a la mayoría de los servicios. Ambos modelos son demasiado extremistas, por lo que .NET proporciona un sistema de seguridad intermedio más refinado, en el cual los programas gozan de distintos niveles de confiabilidad en función de una serie de factores.

El sistema de seguridad de .NET se basa en el código gestionado, de forma que las reglas de seguridad se aseguran por el CLR. La mayor parte del código gestionado es verificado para asegurar la correspondencia de tipos y el comportamiento definido de otras propiedades. Este mecanismo de verificación también garantiza que el flujo de ejecución sea transferido únicamente a localizaciones bien definidas, como los puntos de entrada de los métodos, lo que elimina la posibilidad de saltar a una localización arbitraria. El proceso de verificación impide que el código que no respete los tipos sea ejecutado, y evita una serie de errores como el desbordamiento de los buffer, la lectura de una posición de memoria arbitraria, etc.

El código es posible encontrarlo en varias fuentes, desde correos electrónicos, pasando por el contenido en documentos hasta llegar a descargarse de Internet. Dicho código puede ser malicioso intencional o accidentalmente. Así puede diseñarse para actuar como un caballo de Troya, capturando y enviando contraseñas, duplicándose asimismo o para realizar otro daño.

Para ayudar a proteger los sistemas de cómputo de esos códigos maliciosos, la seguridad .NET proporciona el concepto de ubicar diferentes códigos y distintos niveles de confianza, no sólo de acuerdo con los derechos del usuario que ejecuta el código sino también tomando en cuenta quién autorizó el código y las políticas del sistema.

La seguridad tradicional que ofrecen los sistemas operativos toma en cuenta sólo la identidad del usuario del código. En otras palabras, si se confía en el usuario, también se confía en cualquier código que ejecuta.

Con este capítulo pretendo definir adecuadamente los principales conceptos que tienen relación con seguridad referida al .NET Framework y todo lo que hace esta funcionalidad en esta plataforma.

3.1. SEGURIDAD EN .NET FRAMEWORK

.NET Framework proporciona tres mecanismos fundamentales para proteger los recursos y el código de usuarios no autorizados.

Seguridad de la aplicación Web ASP.NET: La seguridad de la aplicación Web ASP.NET proporciona una manera de controlar el acceso a un sitio al comparar las credenciales autenticadas (o representaciones de éstas) con los permisos del sistema de archivos Microsoft

Windows NT o a un archivo XML que enumera usuarios autorizados, roles autorizados o verbos HTTP autorizados.

Seguridad de acceso a código: La seguridad de acceso a código usa permisos para controlar el código de acceso para recursos y operaciones protegidos. Ayuda a proteger los sistemas de cómputo de código móvil malicioso y proporciona una manera de permitir que el código móvil se ejecute de manera segura. (La seguridad de acceso a código junto con las políticas que la regulan se conoce como seguridad basada en evidencias.)

Seguridad basada en roles: La seguridad basada en roles proporciona la información necesaria para tomar decisiones acerca de lo que puede hacer un usuario. Estas decisiones se pueden basar ya sea en la identidad o membresía del rol del usuario o en ambos.

Estos mecanismos de seguridad usan un modelo simple y consistente para que los desarrolladores familiarizados con la seguridad de acceso a código puedan usar fácilmente seguridad basada en roles, y viceversa. Tanto la seguridad de acceso a código como la seguridad basada en roles se elaboran usando una infraestructura común proporcionada por el tiempo de ejecución de lenguaje común. Debido a que utilizan el mismo modelo e infraestructura, la seguridad de acceso a código y la seguridad basada en roles comparten varios conceptos subyacentes, los cuales se describen más adelante.

3.1.1. SEGURIDAD DE IDENTIDAD (IDENTITY SECURITY)

La mayoría de los sitios Web necesitan restringir de manera selectiva el acceso a algunas partes del sitio. Puede considerar un sitio Web como algo análogo a una tienda comercial. La tienda está abierta para que el público entre y explore, pero hay algunas partes de la instalación, como las oficinas administrativas, que están restringidas a las personas con ciertas credenciales, como los empleados.

De manera similar, cuando un sitio Web registra la información de las tarjetas de crédito de los usuarios, el archivo o la base de datos que almacena esa información, debe estar asegurada para que no tenga acceso del público. ASP.NET, junto con *Internet Information Services* (IIS), puede corroborar las credenciales del usuario, como nombres y contraseñas, utilizando cualquiera de los siguientes métodos de autenticación. Estos métodos se cubren en detalle en la sección "Autenticación" de este capítulo.

- ☑ Microsoft Windows: Autenticassem Windows Básica, Resumen o Integrada.
- ☑ Microsoft Passport Authentication.
- ☑ Formas.
- ☑ Certificados de cliente.

El ***Identity security*****,** por lo tanto**,** es el mecanismo encargado de otorgar permisos y privilegios en función de los metadatos de identidad del *assembly*. El ámbito, el conjunto de código y la unidad básica de granulado es el *assembly*.

El equipo tiene asignadas unas políticas de seguridad. En estas políticas se definen qué privilegios se asignan a cada *assembly* en función de algún o algunos metadatos de identidad. Por

ejemplo, los *assemblies* cuya zona de origen (lugar donde se ubica el *assembly*) sea Internet, no perteneciente a los sitios seguros de Internet Explorer, no tendrá privilegios para escribir en disco o ver propiedades del sistema, entre otras acciones. Estos metadatos de identidad pueden ser el nombre del *assembly*, la versión, el autor del *assembly* (se comprueba por la clave pública de firmado), la zona de ubicación del *assembly*, la URL de ubicación del *assembly* y lo demás del *assembly*.

Puede que un *assembly* esté afectado por más de una de estas reglas de seguridad, si es así el *assembly* recibe los permisos de ejecución unión de todos los otorgados por cada regla.

Además se definen tres diferentes políticas y una cuarta opcional. Se denominan de empresa, de equipo, de usuario y de dominio de la aplicación. La primera es común para todo el dominio, la segunda es común para todo el equipo, la tercera es propia de cada usuario y la cuarta de cada dominio de aplicación (mas adelante en este capitulo lo explico con mayor amplitud). Los privilegios resultantes para cada *assembly* son la intersección de los otorgados por cada política.

3.1.2. SEGURIDAD DE ACCESO A CÓDIGO (CODE ACCESS SECURITY)

Todas las aplicaciones que tengan como objetivo el tiempo de ejecución de lenguaje común, deben interactuar con el sistema de seguridad del tiempo de ejecución. Cuando se ejecuta una aplicación, se evalúa automáticamente y el tiempo de ejecución proporciona un conjunto de permisos. Dependiendo de los permisos que reciba la aplicación, se ejecuta adecuadamente o genera una excepción de seguridad. Las configuraciones locales de seguridad en una máquina en particular deciden en última instancia los permisos que recibe el código.

Es necesario conocer los siguientes conceptos de seguridad de acceso al código para escribir aplicaciones efectivas que tienen como objetivo el tiempo de ejecución de lenguaje común:

- ☑ **Escribir código seguro en su tipo**

Para permitir que el código se beneficie de la seguridad del acceso al código, los desarrolladores de aplicaciones y de componentes deben utilizar un compilador que genera código verificable en la seguridad del tipo.

- ☑ **Sintaxis imperativa y declarativa**

La interacción con el sistema de seguridad del tiempo de ejecución se realiza utilizando llamadas de seguridad imperativa y declarativa. Las llamadas declarativas se realizan utilizando atributos, aunque las llamadas imperativas se realizan utilizando instancias nuevas de clases dentro de su código. Algunas llamadas se pueden realizar sólo de manera imperativa, mientras que otras se pueden realizar sólo de manera declarativa. Algunas llamadas se pueden realizar en ambas formas.

La Sintaxis Declarativa consiste en establecer atributos al código sea a un tipo o sea un miembro de un tipo para definir que es lo que requerimos, por ejemplo:

```
  <Security.Permissions.SecurityPermission(Security.Permissions.SecurityAc
tion.Demand)>

   Public Class UsaSec
```

```
  <Security.Permissions.FileIOPermission(Security.Permissions.SecurityActi
on.PermitOnly)>

  Public Function LeerArchivo(ByVal NombreArchivo As String) As String

  End Function

  End Class
```

La clase **UsaSec**, tiene definido que hay ciertas acciones que son demandadas, donde es necesario que se tengan ciertos derechos para que la clase pueda ser accedida por parte del usuario, por otra parte, el miembro **LeerArchivo** tiene atributos[5] que indican que debe permitirse la ejecución solamente ante el cumplimiento de ciertas condiciones de seguridad.

La otra forma sintáctica es la **imperativa**, en este caso nuestra función no tiene atributos, sin embargo en el momento de comenzar a ejecutarse crea un objeto de seguridad para poder investigar los limites, precisamente, de seguridad que tienen el usuario, la evidencia, a través de las políticas de seguridad establecidas por usuario, por equipo o empresa para conocer, adecuadamente, si se es permitido ejecutar ese función especifica por parte del usuario que accedió.

☑ Solicitar permisos para el código

Se aplican solicitudes al enfoque del ensamble, en donde el código informa al tiempo de ejecución sobre los permisos que necesitan ejecutarse o que específicamente no lo desean. El tiempo de ejecución evalúa las solicitudes de seguridad cuando se carga el código en la memoria. Las solicitudes no pueden afectar el tiempo de ejecución para dar al código más permiso que el tiempo de ejecución le hubiera dado, en caso de que no se hubiera hecho la solicitud. Sin embargo, las solicitudes es la forma en que su código informa al tiempo de ejecución sobre los permisos que requiere para ejecutarse.

.NET Framework proporciona una función especial, almacenamiento aislado, para almacenar datos incluso cuando no se permite el acceso a los archivos; por ejemplo, cuando se descarga y ejecuta un control administrado desde Internet, se le da un conjunto limitado de permisos pero no el derecho para leer o escribir archivos.

El almacenamiento aislado es un nuevo conjunto de tipos y métodos compatibles con .NET Framework para el almacenamiento local. Básicamente, a cada ensamblado se le da acceso a una zona de almacenamiento segregada del disco duro. No se permite el acceso a otros datos y el almacenamiento aislado está disponible sólo para el ensamblado específico para el que se creó.

Una aplicación podría utilizar el almacenamiento aislado para mantener registros de actividad, guardar configuraciones o datos de estado en el disco para utilizarlos más adelante. Puesto que la ubicación del almacenamiento aislado está predeterminada, este tipo de almacenamiento proporciona un modo cómodo de especificar espacio exclusivo para el almacenamiento sin necesidad de determinar rutas de archivo.

[5] Los atributos como se muestran son simplemente ejemplos, por lo que la forma de declarar el atributo no se encuentra completa en el ejemplo.

El código de la intranet local se limita de manera similar, pero menos, y puede tener acceso a una cuota mayor de almacenamiento aislado. Finalmente, el código de la zona de sitios restringidos (sitios que no son confiables) no tiene acceso al área de almacenamiento aislado.

☑ Usar bibliotecas de clase seguras

La seguridad de acceso a códigos permite a sus bibliotecas de clase especificar los permisos que necesitan para accederlas. Deberá estar consciente de los permisos que se requieren para acceder a cualquier biblioteca que use su código y hacer las solicitudes adecuadas en su código.

Para crear *Assemblies* propios es importante tratar de utilizar siempre otros *Assemblies* que "**aseguren**" a través de los mecanismos de atributos por *demand* los derechos del llamador y solamente permitan la ejecución en los casos que el llamador tenga derechos suficientes para realizarlos, obviamente, los *assemblies* propios deben cumplir con este requisito, es decir, controlar cada miembro las características de seguridad que se requieren. Esto no significa que un código bien escrito en .NET deba tener atributos de seguridad para cada clase o para cada tipo y para cada miembro, simplemente es importante establecer los atributos de seguridad cuando se piensa que es factible que el código pueda ser utilizado para realizar tareas para las cuales no fue definido directamente.

Por ejemplo: Supóngase que se tiene un miembro, dentro de un tipo, que requiere escribir un archivo en disco o eliminar un archivo en disco. Dada esta característica, que por si misma sería riesgosa, desde el punto de vista de seguridad, dado que se podría eliminar cualquier archivo, se necesita establecer atributos por demanda para que el usuario que quiera acceder al miembro de nuestro tipo tenga mínimo los derechos para su directorio correspondiente para poder eliminar el archivo. Pero además se podría asegurar realmente que el código cumpla estrictamente todas las necesidades de seguridad que nosotros consideramos si agregamos atributos por lo cual no se permita la ejecución a menos que el usuario tenga prohibido eliminar archivos por ejemplo del directorio winNT o el directorio Windows donde se encuentra instalado el sistema operativo.

Más allá que habitualmente las normativas de seguridad, propias del sistema operativo, no permitirían hacer esto, podemos asegurarnos que eficientemente nuestro código no permita eliminar otros elementos.

Los privilegios y permisos de acceso al código se definen específicamente en el código, en los metadatos del propio *assembly*. Lo que se definen van a ser dos cosas, los privilegios que se exigirán estar otorgados para la ejecución de la aplicación y en segundo lugar los que se denegarán en la ejecución si ya estuviesen otorgados. Por ejemplo, la aplicación puede requerir para ejecutarse correctamente permisos de escritura en el directorio C:\MyApp, luego no se ejecutará si la seguridad de tipo *Identity* no le ha otorgado este permiso. De la misma manera mi aplicación no va a hacer uso de otro directorio que no sea ese, luego puedo denegar el acceso de cualquier tipo a cualquier otra parte del disco, y aunque la seguridad de tipo *Identity* le haya otorgado este permiso la aplicación no lo poseerá y será más segura (frente a bugs que pueda alguien aprovechar de forma maliciosa, por ejemplo).

Estos permisos se establecen en los metadatos del *assembly* en dos puntos: en el código (imperativos), o en los metadatos de descripción del *assembly* (declarativos).

Los permisos declarativos se muestran como atributos en el código también, pero en el *assembly* los almacenan en los metadatos de descripción del mismo. Hacen referencia a todo el *assembly*, no a una parte del código. Indican qué permisos son necesarios para ejecutar el *assembly* (si no, el CLR no lo ejecuta) y cuáles se deniegan (aunque la seguridad de tipo *Identity* los haya otorgado la aplicación no los poseerá). Se comprueban en tiempo de carga antes de ejecutar la aplicación. También es posible solicitar permisos de forma optativa, si no se otorgan se procederá a ejecutar la aplicación en este caso y deberemos comprobar si han sido otorgados de forma imperativa

```
  //Si esto no es garantizado la aplicación fallará al comenzar
  [assembly : UIPermissionAttribute (SecurityAction.RequestMinimum)]
  //Si esto no es garantizado la aplicación continuará ejecutandose
  [assembly : SecurityPermissionAttribute
(SecurityAction.RequestOptional,Flags=SecurityPermissionFlag.UnmanagedCode)]
  // El assembly completo sera bloqueado para ingresar a este driver.
  [assembly : FileIOPermissionAttribute (SecurityAction.RequestRefuse,
Read="C:\\")]
```

Los permisos imperativos o solicitados de forma imperativa, se solicitan en el propio código y se comprueba si se han otorgado mediante el código. Son por lo tanto específicos a una parte del código y al desarrollo particular de la ejecución de la aplicación. Se comprueban en tiempo de ejecución.

```
  try {
      FileIOPermission miPermiso = new
FileIOPermission(FileIOPermissionAccess.AllAccess,@"c:\\")
      miPermiso.Demand();
  }
  catch (SecurityException ex) {
      MessageBox.Show("Excepción de Seguridad: "+ex.Message)
  }
```

3.1.3. SEGURIDAD BASADA EN ROLES (ROLE-BASED SECURITY)

Con frecuencia, los roles se utilizan en aplicaciones financieras o de negocios para aplicar políticas. Por ejemplo, una aplicación puede imponer límites en el tamaño de la transacción que se procesa, dependiendo de que el usuario que hace la solicitud esté en un rol especificado. Los secretarios pueden tener autorización para procesar transacciones que son menos que un umbral específico, los supervisores pueden tener un límite más alto y los vicepresidentes pueden tener un límite aún mayor o no tener límites.

La seguridad basada en roles también se puede utilizar cuando una aplicación requiere varias aprobaciones para completar una acción. Este caso puede ser un sistema de compras en el cuál cualquier empleado puede generar una solicitud de compra, pero sólo un agente de compras puede convertir esa solicitud en una orden de compra que se puede enviar a un proveedor.

La seguridad basada en roles de *Microsoft .NET Framework* soporta autorizaciones al hacer que la información del usuario, que se construye desde una entidad asociada, esté disponible al hilado actual. La identidad puede basarse en una cuenta Windows o ser una identidad personalizada no relacionada con una cuenta de Windows.

Para proporcionar facilidad de uso y consistencia con la seguridad de acceso a códigos, la seguridad basada en roles de *.NET Framework* proporciona objetos *PrincipalPermission* que permiten que el tipo de ejecución de lenguaje común realice la autorización de una manera similar a las verificaciones de seguridad de acceso a códigos. La clase *PrincipalPermission* representa la identidad o rol con el que debe corresponder el principal y es compatible tanto con las verificaciones de seguridad declarativas como con las imperativas. También puede acceder directamente a información de identidad de un principal y realizar verificaciones de roles y de identidad en su código, cuando sea necesario.

.NET Framework proporciona soporte a la seguridad basada en roles que es lo suficientemente flexible y ampliable para cumplir con las necesidades de una amplia gama de aplicaciones. Puede elegir interoperar con infraestructuras existentes de autenticación, como COM+ 1.0 *Services*, o crear un sistema personalizado de autenticación. La seguridad basada en roles es particularmente adecuada para usarse en aplicaciones Web ASP.NET, que se procesan principalmente en el servidor. Sin embargo, puede usar la seguridad basada en roles de *.NET Framework* en el cliente o en el servidor.

Hasta ahora solo he comprobado los privilegios de las entidades de código. Pero además de estos permisos y privilegios existe un elemento más que influye en estos: el usuario de la aplicación.

Aparecen entonces dos nuevos elementos en la arquitectura el autentificador y el autorizador.

El autentificador es el elemento que indica que usuario es el que está accediendo a la aplicación o servicio. En el caso de aplicaciones de formularios Windows es el propio sistema operativo Windows, en el caso de servicios Web y aplicaciones Web es el *Internet Information Server*. Aunque en ambos se casos se puede buscar una vía alternativa de autentificación propia.

El autorizador es el elemento que se encarga de otorgar permisos y privilegios en función del usuario autentificado. La autorización se realiza de forma imperativa en el código haciendo uso de las clases *Identity* que define la identidad del usuario y Principal que define su pertenencia a grupos. Cada autentificador declara subtipos (el sistema operativo Windows define WindowsPrincipal y *WindowsIdentity*) de estos que se pueden instanciar en la ejecución del código para saber que usuario está ejecutando ese código y autorizarle o no.

```
WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();
If (MyIdentity.IsGuest) {…}
If (MyIdentity.IsAnonymous) {…}
If (MyIdentity.IsSystem) {…}
String Str=MyIdentity.AuthenticationType;
Str=MyIdentity.Name;
WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);
If (MyPrincipal.IsInRole("Administradores")) {…}
```

3.1.4. SEGURIDAD DE ACCESO AL CÓDIGO FRENTE A SEGURIDAD BASADA EN ROLES

La principal diferencia entre seguridad de acceso al código y la seguridad basada en los roles definidos en las aplicaciones, es que es el propio sistema forza la seguridad de acceso al código (en concreto, el *runtime de .NET*), mientras que la seguridad basada en roles es implementada por el programador en el código. La seguridad de acceso al código no permite ninguna opción (algo que resulta adecuado en este caso), es decir, el sistema determina automáticamente las operaciones que puede realizar el código. La seguridad basada en roles, por otro lado, está abierta a los deseos del programador. Tendrá que elegirse si necesita implementar un modeló de seguridad basada en roles en la aplicación, así como el nivel con el que se va a aplicar, en el caso de que lo haga.

La seguridad de acceso al código es prioritaria frente a la seguridad basada en roles de las aplicaciones

Los permisos de seguridad de acceso al código se evalúan de forma independiente de los permisos de seguridad basados en roles.

Si una aplicación no tiene los permisos necesarios para crear un archivo en el disco, el sistema de seguridad de acceso al código impedirá a su aplicación escribir en el disco. Se producirá una excepción de seguridad y el sistema detendrá su aplicación. Esta operación de grabación de un archivo en el disco no se podrá realizar, con independencia de cuál sea el componente que haya sido autorizado por el sistema de seguridad basado en roles de su aplicación.

3.2. PERMISOS

En principio se dice que los permisos permiten establecer qué elementos y recursos dentro del sistema operativo y por lo tanto del equipo donde se ejecuta la aplicación, puede nuestro código tener acceso. Esto es el concepto más básico, sin embargo se debe de tener en cuenta que dentro de las aplicaciones y sobre todo con los componentes que se realizan en Visual Studio .Net o con el *.Net Framework*, sea cual fuese la herramienta de decodificación que se utilice para realizar la aplicación, pueden ser configurados de distinta forma. Esto quiere decir que no solo vamos a depender de los derechos que el usuario tenga cómo para acceder a algún recurso sino que además las aplicaciones en si mismas pueden ser configuradas de esta forma.

Cuando se valúa el acceso a un determinado recurso se puede llegar a obtener un nivel de permisos menor del que se supone el componente tiene y esto está dado porque se va a realizar un análisis en cascada de todos los usuarios que van realizando las llamadas aun cuando existan "impersonalizaciones" para establecer el menor nivel posible como para asegurarnos que nuestro código no realice acciones que pueden ser dañinas para el sistema operativo o para cualquier parte de lo que está almacenado en disco. Básicamente, el CLR nos permite administrar esta seguridad tanto sea por establecimiento de derechos para nuestro código como por la correspondencia a roles, esto es a políticas de seguridad en el momento de la ejecución. Se debe tener en cuenta entonces que son niveles totalmente distintos, el primero los derechos del código son los que establecemos para nuestro código especifico en el momento que estamos realizando

la aplicación. En cambio las políticas de seguridad se valúan estrictamente en términos de tiempo de ejecución dependiendo de quien es el que llama este código.

El Tiempo de ejecución de lenguaje común (CLR) permite al código realizar sólo aquellas operaciones para las que tiene permiso. Para que se apliquen las restricciones en código administrado, el tiempo de ejecución usa objetos llamados permisos. Los usos principales de los permisos son los siguientes:

- ☑ El código puede solicitar los permisos que necesita para acceder a los recursos o realizar operaciones.
- ☑ El tiempo de ejecución puede otorgar permisos para codificar con base en las características de la identidad del código, en los permisos que se solicitan y qué tan confiable es el código.
- ☑ El código puede demandar que sus solicitantes tengan permisos específicos.

Por otra parte, también el código puede exigir que se cumpla con ciertas normas de seguridad para que este mismo pueda ser ejecutado. Lo que significa dos cosas, en principio es posible definir que se necesitan ciertos derechos para poder realizar una acción, básicamente todos lo que podemos llegar a comprender, pero además, el código puede prohibir taxativamente ciertos derechos y exigir que esos derechos no existan para que nuestro código se ejecute.

El tiempo de ejecución otorga permisos a dominios y ensambles de aplicaciones. El proceso que sigue el tiempo de ejecución al otorgar permisos puede incluir dos pasos: Computar el conjunto permitido de permisos y determinar los permisos otorgados.

Computar el conjunto permitido de permisos: Al momento de cargar, el tiempo de ejecución determina el conjunto de permisos que cada tipo de política permite que tenga el código. Luego, el tiempo de ejecución intercepta los conjuntos permitidos de permisos para cada nivel relevante de política, resultando en un conjunto de permisos permitidos para el dominio o ensamble de la aplicación.

Determinar los permisos otorgados: El tiempo de ejecución compara este conjunto final de permisos permitidos con los permisos que solicita el ensamble, resultando un conjunto de permisos que se otorga al ensamble. Este paso no aplica para otorgamiento de permisos de dominios de aplicación.

3.2.1. REQUERIMIENTOS DE PERMISOS

Los requerimientos de permisos se establecen a través del análisis de la evidencia pero además es un mecanismo que es posible utilizar por código para poder investigar, adecuadamente, la estructura de seguridad completa y por lo tanto, saber hasta donde podemos llegar con la tarea a realizar.

Ese mecanismo depende de los atributos que serán asignados a los tipos y a su vez los derechos dados por las políticas y la evidencia completa.

Un miembro o método podría ser una función o un procedimiento puede preguntarle a *System.Security.Permissions* si existe o no, un permiso dado y a partir de aquí poder conocer si el usuario tiene los derechos suficientes para realizar la tarea. Se debe tener en cuenta que este requerimiento de permisos hace una evaluación en cascada hacia atrás de todas las llamadas que se realizan hasta llegar a este método.

Esto permite que si intencionalmente un código que pretende ser agresivo, sin derechos, termine encontrando un objeto que impersona para acceder a ciertos recursos el motor que permite la evaluación de seguridad, el conjunto de clases de *System.Security.Permissions*, igualmente va a evaluar que el origen primario de la llamada es un usuario anónimo y por lo tanto no se le pueden dar derechos para realizar la tarea que requiere.

3.2.2. TIPOS DE PERMISO

En el capítulo 7 de este trabajo se retoman los grupos de permisos que a continuación se muestran desde una perspectiva zona de seguridad, como parte de una técnica para comprender y utilizar los permisos. Se puede definir entonces, como resumen conceptual, que existen básicamente tres grupos de permisos.

Los **permisos establecidos por acceso del código (*Code Access Security*)**, son aquellos que definen a qué recursos nuestro código puede acceder. Dentro de la descripción en metadatos del *assembly,* entre los metadatos descriptivos y los atributos, se indican qué privilegios y permisos requiere el *assembly* para ejecutar y cuáles, aún teniéndolos, no desea que se le asignen.

En la Tabla 3.1 se listan las clases relacionados con permisos por acceso del código. Los permisos de acceso a código representan el derecho del código para acceder a recursos protegidos o realizar una operación protegida.

NOMBRE DE CLASE DEL PERMISO	DERECHO REPRESENTADO
DirectoryServicesPermission	Accede a clases System.DirectoryServices
DnsPermission	Accede a DNS (Sistema de nombre de dominio)
EnvironmentPermission	Lee o escribe variables ambientales
FileDialogPermission	Accede a archivos que han sido seleccionados por el usuario en un cuadro de diálogo Abrir
EventLogPermission	Lee o escribe acceso a servicios de registro de eventos
FileIOPermission	Lee, coloca apéndices o escribe archivos o directorios
IsolatedStoragePermission	Accede a almacenamiento aislado, que es el almacenamiento que está asociado con un usuario específico y con algún aspecto de la identidad del código, como su sitio, editor o firma en el Web
PrintingPermission	Accede a impresoras

ReflectionPermission	Descubre información sobre un tipo de tiempo de ejecución
OleDbPermission	Accede a bases de datos utilizando OLE DB
PerformanceCounterPermission	Accede a contadores de rendimiento
RegistryPermission	Lee, escribe, crea o elimina claves y valores de registro
SqlClientPermission	Accede a bases de datos SQL
ServiceControllerPermission	Accede a servicios en ejecución o detenidos
MessageQueuePermission	Accede a colas de mensaje a través de las interfaces administradas Microsoft Message Queue (MSMQ)
SecurityPermission	Ejecuta, afirma permisos, invoca código no administrado, salta la verificación y otros derechos
SocketPermission	Hace o acepta conexiones en una dirección de transporte
UIPermission	Accede a la funcionalidad de la interfaz
WebPermission	Hace o acepta conexiones en una dirección Web

Tabla 3.1. Descripción de Permisos

Los **permisos relacionados con Identidad (*Identity*)**, permiten el poder identificar fehacientemente que el usuario es un usuario autorizado al realizar las tareas. Los privilegios se asignan en función de los metadatos de identidad (nombre, versión, cultura, autor, clave pública de firmado,…) del *assembly* a ejecutar.

Es posible definir permisos específicos por identificación del código en si mismo, esta es la primera versión de un motor de ejecución permite establecer niveles de permisos, no solamente por el usuario del sistema operativo, sino además por la compañía o el productor del código que se va a ejecutar. En la Tabla 3.2 se muestran dichas identidades con su respectiva clase.

NOMBRE DE CLASE	IDENTIDAD REPRESENTADA
PublisherIdentityPermission	La firma digital del editor de software
SiteIdentityPermission	El sitio Web en el que se originó el código
StrongNameIdentityPermission	El nombre fijo (compartido) del ensamble
ZoneIdentityPermission	La zona en la que originó el código
URLIdentityPermission	El URL en donde se originó el código (incluyendo el prefijo de protocolo http, https, ftp, etc.)

Tabla 3.2. Descripción de Clases de una Identidad

Por tanto los permisos de identidad indican que el código tiene credenciales que soportan un tipo particular de identidad. De esta forma un Administrador va a poder directamente definir, por ejemplo, que todo componente generado de la empresa de desarrollo A puede tener estos derechos y tal vez estos otros no.

El tercer elemento importante que proporciona un mecanismo para descubrir si un usuario (o el agente que actúa en representación del usuario) tiene una identidad en particular o es un

miembro de un rol específico son los **permisos basados en políticas de seguridad (*Role-based security*)**, esto se define siempre administrativamente estableciendo que el usuario, que ejecuta una determinada tarea, debe pertenecer a cierto rol dentro de la estructura de seguridad. Esto es casi idéntico a decir que el usuario pertenece a un grupo dentro del directorio activo (*active directory)* o dentro del controlador de dominio si el sistema operativo fuese Windows NT pero además este criterio de evaluación de permisos también se pueden establecer en casos en que se realice una "impersonalización" que el código cambia de usuario en el que se va a utilizar en ejecución por motivos precisamente de seguridad, por más personalización que se realice siempre el usuario definido debe tener cierta pertenencia a algunas políticas de seguridad o roles de seguridad para realizar ciertas tareas. *PrincipalPermission* es el único permiso de seguridad basado en roles

El tiempo de ejecución, como ya mocioné, proporciona clases de permisos integrados en el espacio de nombre *System.Security.Permissions* y también proporciona soporte para diseñar e implementar clases de permisos personalizadas.

La ejecución final de la aplicación o servicio, y los permisos y privilegios con que ésta corra, dependerán de estos tres elementos (de la intersección de ellos).

Si un elemento de código necesita un permiso para ejecutar, ese permiso debe de estar asignado tanto a ese elemento como a todos los que le llaman. Es decir, los privilegios deben estar otorgados en todos los *assemblies* que se llaman.

Por ejemplo, si el *assembly* A llama o hace uso del assembly B y este último requiere del permiso P, sólo ejecutará si A también tiene asignado este permiso.

3.3. TIPOS SEGUROS

Se definen como tipos seguros los objetos que solamente pueden acceder al espacio de memoria que tienen definidos como propiedades o campos para si mismo.

Este tipo de características es controlada por el compilador en tiempo de ejecución, *Just in Time (JIT)* en el momento que termina de compilar cada uno de los tipos definidos dentro del *.NET Framework* si en el momento de controlar esto se detecta que hay declaraciones que implican que posiblemente el código no sea seguro, a través de la directiva *andsave,* por ejemplo en *C#* o *C++,* o en la utilización de carga de bibliotecas dinámicas, entonces el motor de ejecución solamente permite ejecutar este código, siempre y cuando, se haya obtenido de una unidad física, como se definió anteriormente, un disco rígido. Significa entonces, que las aplicaciones que realizan esta tarea no pueden "descargarse" directamente desde la *web*.

Un elemento a tener en cuenta es que el código generado con el *Compac .NET Framework* siempre debe ser de "tipo seguro".

Dado que las aplicaciones para instalar en dispositivos de tipo *Pocket PC*, etc., que ejecuten Windows CE, como sistema operativo, deben ser instalados a través de otro equipo, por ejemplo una PC con la cual está relacionado, se considera casi lo mismo que si ese código fuese descargados de la *web*, por lo tanto, todo código para dispositivo inteligente, debe ser estrictamente de "tipo seguro" en caso contrario el *Compact Framework* no permitirá su ejecución.

Para aquellos casos en que se requiera controlar estrictamente que los tipos sean seguros, existe un ejecutable que se llama "Peverify.exe" este permite a los que generan lenguajes de programación compatibles con el *Microsoft Intermedia Language* (MSIL) comprobar que todo el código generado, a través del compilador, termine siendo de tipo seguro, a su vez esta herramienta le puede permitir al administrador controlar que un aplicativo, un conjunto de *assemblies* o solo un *assembly*, cumpla estrictamente con la normativa de tipo seguro antes de que el mismo sea utilizado en alguno de los tipos de la red.

Se ha definido en el capítulo anterior las características más importantes que tiene el código generado para .NET dado que el mismo es ejecutado por un sólo motor de ejecución el *CLR*. Este tipo de código se suele llamar *manager code* esto es código administrado. No solamente el código administrado permite controlar adecuadamente el descarte de los elementos de memoria que ya no se utilizan, a través del *Garbage Colector* o la compilación final, *Just In Time*, en el momento que se comienza a ejecutar el código o características similares como las que establece el *Common Language System* o *Common Language Specification* que permiten heredar objetos desde distintos lenguajes.

Hay una característica importante y es que: el código administrado permite establecer que el motor de ejecución siempre controlará que ningún objeto pueda acceder a información que este definida como privada de otro. Esto para un desarrollador de VB es extraño de pensar, sin embargo si utilizamos o conceptualizamos código en términos de lenguaje como por ejemplo C++, inclusive desde VB, llamando funciones de bajo nivel comúnmente conocidas como API, sería factible acceder a segmentos de memoria simplemente porque se "pierde el control" de la misma o se obtiene un puntero a un segmento y se escribe directamente en él.

Desde el punto de vista del código que se ejecuta dentro del *CLR* esto es literalmente imposible dado que cada objeto tiene su propio espacio de memoria y no puede ser accedido a este por ningún otro código. En términos generales el código administrado nos permite controlar adecuadamente que no existan violaciones de espacio de memoria de objetos de código entre los distintos componentes que se ejecuten a través de este motor. Sin embargo cabe la posibilidad de en algún momento sea necesario salirse del esquema y acceder directamente a ciertos recursos, si bien esto es posible, es necesario indicarlo explícitamente y en esos momentos lo controles de seguridad se hacen aun más críticos dado que el motor de ejecución solo va permitir ejecutar este código bajo ciertas condiciones.

Cuando se establece explícitamente que el código no va a ser código administrado la responsabilidad es sin embargo absolutamente del lado del desarrollador porque tiene que asegurarse no excederse en términos de tamaño de memoria que va escribir o que va acceder ni que acceda a ningún sementó de memoria que sea prohibitivo por parte del sistema operativo.

Si bien es factible salirse del esquema de código administrado, esto es un riesgo muy grande en términos de seguridad, tanto es así que el lenguaje más utilizado estadísticamente es *Visual Basic*, en su versión *Visual Basic .NET*, hace casi imposible definir una operación que escape estrictamente al código administrado, la única salvedad es cuado se define a través del atributo "*dllimport*", que se va llamar una función de bajo nivel, la función de bajo nivel del sistema operativo (API) es en realidad código que no va a ser ejecutado por el motor sino directamente

por el sistema operativo. Ese es el único caso en el que Visual Basic .NET nos permitiría de alguna forma escapar del código administrado. En otros lenguajes como es el caso de C# no solamente se puede establecer de esta forma una operación fuera del código administrado si no que es factible definir funciones de miembros de objetos enteros con el atributo ***andsave*** en el momento que esto se hace la ejecución de dicho código, hasta que termine su declaración, corre estrictamente por responsabilidad del desarrollador y hay que evitar, bajo cualquier punto de vista, caer en este riesgo dado que es absolutamente imposible asegurarnos que nuestro código no cometa ningún tipo de violación en cuanto al sistema operativo, los recurso que se accede, etc.

3.4. POLÍTICAS DE SEGURIDAD

La política de seguridad es un conjunto configurable de reglas que sigue el tiempo de ejecución al decidir cuáles permisos otorgar al código. Los administradores establecen la política de seguridad y el tiempo de ejecución que aplica. El tiempo de ejecución asegura que el código pueda acceder sólo a los recursos e invocar únicamente al código que la política de seguridad le permite. La política de seguridad define varias zonas y asocia cada una de ellas de manera predeterminada con un conjunto nombrado de permisos. Si no se conoce ninguna otra característica de identificación del código, el tiempo de ejecución usa la política para la zona desde la que se origina el código.

En tiempo de ejecución, las políticas de seguridad son las que gobiernan el modo en que un código va a ser ejecutado según el conjunto de reglas que el CLR tiene que seguir para poder ejecutar el mismo. Básicamente esta política de seguridad se deriva de distintos elementos, por una parte es configurable por los administradores a través de las herramientas administrativas de seguridad del *NET Framework*, pero además en el momento que se carga el *assembly* el CLR controla la política que se va a aplicar, el conjunto de relación que determina la política de seguridad completa y se denomina "**evidencia**".

Esencialmente, la evidencia surge desde el momento en que existe un *host*, esto es un motor, el cual va utilizarse como entorno para ejecutar dicho código, por ejemplo: Internet Explores es en si mismo un *Host* para todo aquello que sean aplicaciones *web* pero desde el punto de vista del cliente, desde el punto de vista del servidor, una aplicación *web,* utiliza un *host* tipo servidor como es el caso del servidor de ejecución de ASP .NET. Además de las características que tenga el *host,* hay ciertos elementos que dependen de cómo el motor de ejecución evalúa en el momento en qué comienza a cargar este *assembly*, sobre todo cuando el *assembly* es el primario, el dominio de una aplicación o *application domine*.

Uno de los elementos que se utiliza para evaluar la evidencia es: desde dónde se está cargando el código, esto puede ser por ejemplo directamente un disco local, en este caso, por defecto, habitualmente las aplicaciones, *aplication domine* o *host*, tienen todos los derechos necesarios para acceder a cualquier recurso. Esto puede ser modificado si por ejemplo: un administrador determina derechos específicos para el código generado por una compañía de desarrollo, esto es por el publicador del código. Supóngase por el contrario que la aplicación se carga no desde el disco local sino desde un origen externo, en estos casos, las políticas de seguridad van a estructurar la evidencia dependiendo de dos atributos más, el sitio y de zona.

Cuando se habla de sitio se refiere específicamente a una dirección *DNS* que haga referencia a un sitio que está definido como de confianza, por ejemplo: www.compania.com. Por otra parte, hablamos de zona cuando nos estamos refiriendo a característica de seguridad estrictamente establecidas a través de las zonas de acceso del navegador, esto es de *InternerExplorer*, por ejemplo: los atributos de la evidencia, van a ser distintos si se trata de una pagina local, si se trata de un grupo de paginas o una aplicación *web*, está publicada dentro de la intranet o si se trata directamente de código que se intenta descargar desde Internet. Son tres zonas totalmente distintas.

Establecer una zona puede darse sobre todo en términos de intranet por dos mecanismos.

El primero es la utilización de NetBios, esto es acceder por nombre de equipo, directamente al origen del código, esto es lo que se comprende comúnmente como intranet. Si se pretende acceder al código a través de su dirección DNS, si está bien definido el conjunto de árboles DNS de la red interna, igualmente el motor de ejecución va a detectar que se trata precisamente del mismo nombre de dominio o nombre DNS, al cual pertenece el equipo que quiere ejecutar este código, por lo tanto, si bien no es un nombre NetBios, igualmente se considera como zona intranet, cualquier otra que no pueda ser resuelta directamente como el grupo DNS o dominio DNS al cual pertenece este equipo, entonces pasará a considerarse como zona Internet, a menos que se halla identificado implícitamente, dentro de la zona de seguridad del Internet Explores de cada equipo. Se debe tener en cuenta que este establecimiento de zonas de confianza, puede dispararse directamente desde el *Active Directory*, esto quiere decir que nos es necesario obligadamente que un administrador deba acceder a cada Internet Explorer a cada equipo dentro de la red para definir zonas seguras, características de intranet, características de seguridad de Internet, etc. Sencillamente en toda esta configuración, un administrador de red puede establecerla en las política dentro del *Active Directory*, por lo tanto, derivar estas características, a cada uno de los equipos que se autentica con el propio *Active Directory*.

La seguridad puede ser establecida no solamente por el origen del código, la compañía que lo generó, si no además por el sitio dirección completa para la *web* que es dirección DNS, o por la zona, que es la pertenencia a uno de los grupos zonales, establecidos en Internet Explorer, sea individualmente o a través del *Active directory*.

Por otra parte si en el motor de ejecución, el *host* que va ejecutar el código, se trata una aplicación de tipo win32, pasa a tener otras características de seguridad, dado que habitualmente el código se ejecuta en modo local. En estos casos el administrador debe autorizar adecuadamente en general por identificador del publicador del código para que el mismo pueda ejecutarse con los derechos necesarios.

3.4.1. NIVELES DE POLÍTICA

Más halla de la identificación del origen del código, una vez que éste ha sido establecido, por ejemplo por zonas, pasan a aplicarse las políticas de seguridad que administrativamente se establecen. Las políticas de seguridad se dividen en cuatro grupos o niveles.

- ☑ Política empresarial

- ☑ Política de máquina
- ☑ Política de usuario
- ☑ Política de dominio de la aplicación

Cada nivel de política tiene sus propias configuraciones. El tiempo de ejecución intersecta estos niveles de política al calcular el conjunto de permisos permitido. Por predeterminación, las políticas de dominio de usuario y de la aplicación son menos restrictivas que la política de máquina.

El de mayor control es la **directiva de empresa,** solamente un administrador dentro del *active directory* puede establecer este tipo de derechos o directivas. En este caso se aplica a todo el código que sea ejecutado por el motor de .NET, que es el código administrado, que se halla publicado dentro de la red de la compañía. Habitualmente se define esta directiva de empresa por un archivo de configuración de empresa que es posible enviar a cada equipo para que se aplique directamente de nuevo a través de los distintos mecanismos de administración a través de *active directory*.

Un segundo grupo, que sólo puede ser establecido por administradores, son las **directivas de equipo,** en este caso el administrador central de la red o un usuario autorizado como administrador local, de un equipo especifico, pueden cambiar estas configuraciones y se refieren estrictamente al código administrado que se ejecuta cargándose desde una unidad física local[6].

El tercer grupo es la **directiva de usuario,** el administrador es el que establece esta directiva, en algunos casos un usuario puede definir que una cierta aplicación, que el quiere ejecutar, le va a dar derechos a hacer ciertas tareas pero esto es estrictamente ejecutado por el motor de ejecución, *CLR*, para evitar que, aun cuando el usuario tiene derechos, estos sobrepasen los derechos de la zona, los derechos del sitio o los derechos de cualquiera de las otras directivas. Todos los procesos asociados a un usuario especifico son los que se pueden controlar a través de estas política de seguridad, de directiva de usuario, y se suman a la características anteriores, así puede existir directiva de empresa, directiva de equipo y además directiva de usuario.

Finalmente existe la **directiva de dominio de aplicación**, que son los derechos que el *host* tiene para ejecutar un determinado tipo de código, por ejemplo: el navegador Internet Explorer, donde el código *script* que ejecuta, tiene prohibidas ciertas tareas como por ejemplo: acceder al disco físico, escribir en el registro de Windows, etc. esto está dado precisamente por una directiva de seguridad del *host* que ejecuta el *script*. Para el mismo caso, aun cuando, se cargue código de manera local, que el usuario tenga derechos necesarios para ejecutarlo, que en el equipo este autorizado y que la directiva de empresa defina que efectivamente se puede ejecutar este código, las directivas del dominio de la aplicación, es decir del *host*, van a prohibir ciertas tareas dado que los derechos del *host* no abarcan esas posibilidades.

[6] Cuando se habla de unidad física local se dice que son los discos rígidos y no se consideran como unidades físicas locales, las lectoras de disquete, dispositivos de lectores ópticos, etc. Todo aquel elemento que sea removible no se considera como administrado por el equipo y por lo tanto no puede formar parte de la directiva de equipo.

El análisis de la evidencia se realiza de la siguiente forma: en el momento en que un *host* carga código administrado, esto es que carga en memoria un *assembly*, se evalúa en principio la evidencia de dicho *assembly*, esa evidencia esta dada por: a) los derechos del *host* de dominio de aplicación, b) los derechos específicos que este *assembly* tiene otorgados. Una vez que se definen estos elementos se reanalizan los derechos desde abajo hacia arriba por la tabla 3.3 de las políticas de seguridad.

TIPO DE POLÍTICA	ESPECIFICADA POR	APLICA A
Política empresarial	Administrador	Todo el código administrado en una configuración empresarial
Política de máquina	Administrador	Todo el código administrado en la computadora
Política de usuario	Administrador o usuario	Código en todos los procesos asociados con el usuario actual del sistema operativo cuando inicia el tiempo de ejecución
Política de dominio de la aplicación	Código *host* del dominio de la aplicación	Todo el código administrado en el dominio de la aplicación del *host*

Tabla 3.3. Descripción de Políticas

Los administradores pueden configurar la política de seguridad de tal manera que los sitios y editores individuales puedan tener más o menos permisos que lo permitido por la política predeterminada. Por ejemplo, un administrador puede especificar que todo el código descargado del sitio Web de la Empresa XYZ, un socio de negocios confiable, pueda tener todos los permisos. El mismo administrador puede especificar que todos los demás códigos de Internet tengan un conjunto de permisos más restringido, como acceso limitado al almacenamiento aislado y uso de una funcionalidad segura de la interfaz.

Con base en la política de seguridad, el tiempo de ejecución otorga permisos tanto a los ensambles como a los dominios de la aplicación. Durante la ejecución, el tiempo de ejecución asegura que el código acceda sólo a los recursos para los que tiene permiso de acceso. Cuando se hace un intento para cargar un ensamble, el tiempo de ejecución utiliza la política de seguridad para determinar qué permisos otorgarle al ensamble. Después de examinar la información, la evidencia invocada que describe el ensamble, el tiempo de ejecución utiliza la política de seguridad para decidir qué tan confiable es el código y, por lo tanto, los permisos que se le otorgan a ese ensamble. La evidencia incluye, pero no está limitada, al editor del código, su tamaño y su zona. Así mismo, la política de seguridad determina qué permisos otorgar a los dominios de la aplicación.

Primero se evalúan los derechos de directiva de usuario, a continuación, los derechos de directiva de equipo y finalmente los derechos de directivas de empresa. En cualquier caso, una anulación de derechos, la prohibición de realizar alguna tarea, siempre queda marcada como "literalmente definida". Si para la directiva de usuario se permite cierta tarea pero no es permitida para la directiva de equipo, el código no puede realizar esta opción, si en la directiva usuario o directiva de equipo, no define ninguna característica particular para dicha tarea, la directiva de empresa es la que toma el comando y por otra parte, si la directiva de usuario y directiva de

equipo autorizan la ejecución de una tarea, pero sin embargo la directiva de empresa lo prohíbe, esta prohibición es la que prevalece.

El análisis de la evidencia pasa por las distintas etapas, que es controlar el *host*, controlar lo que dicen los *assemblies* y luego las directivas de usuario, de quipo y empresa, sin embargo, la prevalecía es de arriba hacia abajo, vale mas la directiva de empresa que la directiva de equipo, valen más estas dos, que la directiva de usuario.

3.5. OBJETO PRINCIPAL

Un principal es un usuario o agente que actúa en nombre del usuario. El objeto principal, que depende de *System.Security*, representa de alguna forma la identidad de la ejecución del código y el rol del usuario y por lo tanto es el que ejecuta el código "en nombre de" este mismo usuario. El objeto principal es el mecanismo básico, a nivel de código, para poder identificar claramente, quién es el usuario, qué derechos tienen, qué pertenecías tiene, etc.

El objeto Principal, generado a través de conjuntos de clases base, puede ser de dos tipos: *Genérico o Windows*. Además es factible crear tipos *Personalizados* de Principal heredando de alguno de estos.

Principales genéricos: Los principales genéricos representan usuarios no autenticados y los roles disponibles para ellos.

Principales de Windows: Los principales de Windows representan usuarios de Windows y sus roles (o sus grupos de Windows NT o Windows 2000). Un principal de Windows puede personificar a otro usuario, lo que significa que el principal puede tener acceso a un recurso en nombre del usuario al tiempo que presenta la identidad que pertenece a ese usuario.

Principales personalizados: Los principales personalizados se pueden definir a través de una aplicación en cualquier forma que sea necesaria para esa aplicación en particular. Pueden ampliar la noción básica de la identidad y los roles del principal. Sin embargo, la aplicación debe proporcionar un módulo de autenticación y los tipos que implementan el principal.

Por ejemplo: Una aplicación Web en la cual se tiene acceso anónimo, en este caso el Principal corresponderá a un principal Genérico, por lo tanto el usuario no tiene porque estar registrado dentro del Directorio Activo o en el Dominio NT4, por el contrario si se trata de un sitio que requiere autenticación a través de los usuarios del dominio o se trata de una aplicación Win32, que se ejecuta directamente en su propio *host*, entonces este principal es de tipo "Windows Principal" y está vinculado directamente al Directorio Activo o al Dominio NT4. En el caso de ser un Windows Principal se tendrá la identificación completa del usuario, esto es, su nombre de usuario o identificador de accesos a la red, su nombre completo, su dominio en el cual fue autentificado, los grupos a los que pertenece, etc. en cambio Genérico "*Generic Principal*" no posee ninguna de estas características.

El hecho de utilizar Principal personalizados no significa que sea posible pasar por encima de estos dos grandes grupos, sencillamente, podemos definir nuestro propio principal que heredará de *Generic* o de *Windows*, lo que permitirá agregarle atributos, a este objeto Principal, para facilitar la tarea.

Un caso típico sería un sitio personalizado, donde se definen cuales son la preferencias del usuario, por ejemplo: el tipo de música en un sitio que vende CD's, el tipo de información en un sitio de noticias, etc. Esa personalización o perfil del usuario, generalmente se almacena en una base de datos y en el momento que el usuario accede debe recuperarse de la misma y guardase en alguna otra parte.

En el caso del objeto Principal Personalizado todos estas atributos se podrían agregar al mismo y dado que el motor de ejecución a través del objeto "*session*" o del objeto "*context*" nos permite identificar al usuario, podríamos entonces asignar nuestro propio objeto principal en el momento que el usuario inicia sesión y a partir de ahí disponer, en el transcurso de la aplicación a ASP .NET, de toda la información de perfiles sin tener que almacenarla en otro lado ni tener que recuperarla cada vez de la base de datos, simplemente, *context.user* nos devolvería el objeto principal personalizado con todos los atributos que le hallamos agregado.

3.6. AUTENTICACIÓN Y AUTORIZACIÓN

La autenticación es el proceso de descubrir y verificar la identidad de un principal al examinar las credenciales del usuario y validar dichas credenciales contra una autoridad. La información obtenida durante la autenticación puede ser usada directamente por su código, lo cual significa que una vez que se descubra la identidad del principal, puede usar la seguridad basada en roles de *.NET Framework* para determinar si se debe permitir a ese principal acceder a su código. Si no ocurre la autenticación, se dice que el usuario es anónimo.

La autenticación es el proceso de aceptar credenciales de un usuario y validar esas credenciales con alguna autoridad. Si las credenciales son válidas, se dice que se tiene una identidad autenticada. La autorización es el proceso de determinar si esa identidad autenticada tiene acceso a un recurso dado. La autenticación se puede llevar a cabo mediante el sistema o mediante lógica empresarial, y está disponible a través de una sola API. Las API de autenticación son totalmente ampliables, de manera que los desarrolladores pueden utilizar su propia lógica empresarial según sea necesario. Los desarrolladores pueden codificar sus necesidades de autenticación en una sola API y pueden revisar los métodos de autenticación subyacentes sin realizar cambios importantes en el código.

Específicamente referido a las aplicaciones *web*, sean Internet o intranet, que son ejecutadas por el motor de ASP .NET, existen lo que se denominan mecanismos de autenticación. Esto no es propio de ASP .NET sino que existe desde las primeras versiones de *Internet Information Server*, servidor de aplicaciones *Web*, dentro de *Windows NT*.

Básicamente la autenticación es el **mecanismo** por el cual se identifica a aquel usuario que accede a un recurso, es decir que "**llama"** una pagina *web* contra un determinado autorizante. A partir de la autenticación se puede obtener, por ejemplo, la pertenencia a roles y de esa forma acceder entonces a los derechos, la evidencia, etc., que un determinado usuario tenga. Dicha información está disponible a nivel de código.

Actualmente se utilizan varios mecanismos de autenticación, muchos de los cuales se pueden utilizar en la seguridad basada en roles de *.NET Framework*. Los mecanismos de autenticación que se utilizan con mayor frecuencia son:

Básica: Para cumplir con las especificaciones HTTP, la mayor parte de los exploradores soportan la autenticación Básica. IIS solicita a los usuarios una cuenta y contraseña Windows válida. Este esquema no se considera como un método seguro de autenticación de usuarios ya que el nombre y contraseña de un usuario se pasan sobre la red como texto claro. Este se debe utilizar junto con un sistema de seguridad externo, como el de Capa de sockets segura (SSL).

Resumen: La autenticación de resumen encripta la información de contraseña del usuario y proporciona un mecanismo para ayudar en la prevención de algunos ataques comunes al servidor. Es un proveedor de soporte de seguridad (SSP) disponible en la versión beta del sistema operativo Windows 2000.

Microsoft Passport: es un conjunto de servicios para negocios electrónicos que pueden utilizar los desarrolladores del Web en su sitio. *Microsoft Passport* ofrece los siguientes servicios:

Ingreso *Passport* único - ofrece un servicio de autenticación que permite que los usuarios de *Passport* utilicen un solo nombre y contraseña de usuario para acceso autenticado a cualquier sitio Web que utilice la autenticación *Passport*.

Compras express de *Passport* - ofrece un servicio de monedero que permite a los usuarios *Passport* guardar su información de tarjetas de crédito y domicilio con *Passport* y después poner a disposición esta información en cualquier sitio Web que utilice el monedero *Passport*.

Windows NT LAN Manager (NTLM): NTLM es un protocolo de autenticación que se utiliza en las redes y que incluye sistemas que ejecutan versiones del sistema operativo Microsoft Windows NT anteriores a Windows 2000, y en sistemas independientes.

Kerberos: El protocolo *Kerberos* es un sistema de autenticación que permite a un cliente probar su identidad en el servidor sin tener que pasar ninguna información que pudiera utilizarse posteriormente para personificar al cliente. Esta opción generalmente es más fácil de implementar en sitios de intranet y de empresa a empresa (B2B) que en sitios de empresa a consumidor (B2C) debido a la complejidad de administrar la seguridad pública distribuida.

Los **mecanismos** que se utilizan en el motor de ASP .NET para autenticación pueden ser la autenticación **Básica** establecida por *http* directo, sin inclusión de nombre de dominio, la autorización de tipo **"*Digest*"**, autorización directamente definida por el **Sistema Operativo** cuando un sitio no tiene acceso anónimo se requiere que exista una autenticación a través de *NT Land Manager* [7] (NTLM) o autorizaciones mas complejas como *Kerberos* [8]. Un método externo de autenticación es posible utilizar directamente *Passport*, a través de este mecanismo la entidad autorizante se encuentra fuera de los mecanismos de seguridad o definición de usuarios propios de la compañía y se identifican directamente a través de un *token*, número de identificación único generalmente una larga cadena hexadecimal, que permite conocer unívocamente quién es la persona que está accediendo. Este mecanismo de *Passport* es el que utilizan elementos de comunicación como *Hotmail*, *Messenger*, *Calendar*, etc.

[7] Mecanismo de autenticación básico que tiene Windows NT y que comparte con Windows 2000

[8] Mecanismo por el cual la autenticación requiere tres elementos que identifiquen fehacientemente al llamador para poder darle derechos de acceso

Finalmente existe un mecanismo personalizado para realizar la autenticación en este caso es la aplicación directamente la que va a reconocer quien es el llamador y asignar los derechos específicos.

El ejemplo acerca de un sitio con personalización es un caso típico en el cual dentro del mecanismo de autenticación es necesario que existan etapas definidas específicamente por la aplicación, por ejemplo, para poder obtener el perfil del usuario y por lo tanto presentar el sitio de la manera en que el usuario haya decidido que le resulta mas cómodo. Los mecanismos definidos por la aplicación se utilizan habitualmente definiendo lo que se llama mecanismos de autenticación, *forms*, dentro de ASP .NET.

En la autenticación con formularios de ASP.NET, el usuario proporciona credenciales y envía los formularios. Si la aplicación autentica la solicitud, el sistema emite una *cookie* que contiene las credenciales en algún formulario o una clave para obtener de nuevo la identidad. Las solicitudes subsecuentes se emiten con la *cookie* en los encabezados de las solicitudes y se autentican y autorizan con un controlador de ASP.NET utilizando cualquier método de validación que desee la aplicación. Si no se autentica una solicitud, se utiliza el redireccionamiento de cliente de HTTP para enviar la solicitud a un formulario de autenticación, donde el usuario puede proporcionar credenciales. La autenticación con formularios se utiliza algunas veces para la personalización de contenido para un usuario conocido. En algunos de estos casos, la cuestión es la identificación en lugar de la autenticación; por tanto, la información de personalización de un usuario se puede obtener con solo tener acceso al nombre de usuario.

La autorización es el proceso de determinar si se permite a un principal realizar una acción solicitada. La autorización ocurre generalmente después de la autenticación, y utiliza información acerca de la identidad del principal para determinar a qué recursos puede acceder el principal. La seguridad basada en roles de Microsoft *.NET Framework* se puede usar para implementar la autorización

El propósito de la autorización es determinar si se concede acceso a una identidad solicitante para un recurso dado. ASP.NET ofrece dos tipos de servicios de autorización: autorización de archivos y autorización de URL. La autorización de archivos determina las listas de control de acceso que se consultan en función del método HTTP utilizado y la identidad que realiza la solicitud. La autorización de URL es una asignación lógica entre las partes del espacio de nombres URI y varios usuarios y funciones.

3.7. EVIDENCIA

La evidencia es la entrada al mecanismo de tiempo de ejecución para tomar decisiones con base en la política de seguridad. La evidencia indica al tiempo de ejecución que el código tiene una característica particular. Las formas comunes de evidencia incluyen firmas digitales y la ubicación de la que se origina el código, pero la evidencia también se puede diseñar de manera personalizada para representar otra información que sea significativa para la aplicación.

Los *hosts* confiables de dominios de aplicaciones pueden presentar evidencias sobre un dominio de aplicación que permita al tiempo de ejecución decidir qué permisos otorgar al dominio de la aplicación. Esta información permite al tiempo de ejecución evaluar las políticas de

la máquina y del usuario y devolver el conjunto de permisos que se otorgarán al dominio de la aplicación. Si el *host* no cuenta con permiso para proporcionar evidencia, el dominio de la aplicación obtendrá los permisos que se han otorgado al *host*.

El tiempo de ejecución obtiene evidencia sobre los ensambles de *host* confiables de dominios de aplicaciones o directamente del cargador. Algunas evidencias, como dónde se origina el código, necesitan provenir del *host* de dominio de la aplicación porque sólo el *host* conoce esta información. Otra evidencia, como la firma digital de un ensamble, es inherente al código mismo, de tal manera que puede provenir del cargador o de un *host* confiable. Típicamente, cuando se carga código, el tiempo de ejecución valida cada una de las firmas digitales del ensamble. Si la firma digital es válida, el *host* pasa la información de firma como evidencia al mecanismo de la política de tiempo de ejecución. Además, un ensamble o un *host* puede proporcionar evidencia personalizada como un recurso que es parte del ensamble. Los administradores y desarrolladores pueden definir estos tipos nuevos de evidencia y ampliar la política de seguridad para reconocerla y usarla. La tabla 3.4. se enumeran los tipos de evidencias que pueden presentar al *host*.

EVIDENCIA	DESCRIPCIÓN
Editor	Firma del editor de software, esto es, el firmante *AuthentiCode* del código
Sitio	Sitio de origen, como www.microsoft.com
Nombre fijo	Nombre criptográficamente fijo del ensamble
URL	URL de origen
Zona	Zona de origen, como la Zona de Internet
Análisis	Análisis criptográfico: MD5 ó SHA1
Directorio de la aplicación	El directorio en el que se instala la aplicación

Tabla 3.4. Tipos de evidencias

El mecanismo de la política de tiempo de ejecución combina la evidencia del *host* y el ensamble, y luego utiliza la evidencia para determinar de qué grupos de código es miembro el código. En última instancia, la membresía del grupo de código junto con los permisos solicitados determinan los permisos que se otorgan al código.

3.7.1. SEGURIDAD BASADA EN EVIDENCIAS

.NET Framework crea el concepto de seguridad basada en evidencias. La evidencia sólo hace referencia a entradas en la directiva de seguridad sobre código. Básicamente es el conjunto de respuestas a preguntas formuladas por la directiva de seguridad:

- ¿De qué sitio se obtuvo el ensamblado?

Los ensamblados son las unidades de creación de las aplicaciones de *.NET Framework*. Constituyen la unidad fundamental de desarrollo, el control de versiones, la reutilización, el ámbito de activación y la autorización de seguridad. Los ensamblados de una aplicación se descargan en el cliente desde un sitio Web.

- ¿De qué URL se obtuvo el ensamblado?

La directiva de seguridad requiere la dirección específica desde donde se descargó el ensamblado.

- ¿De qué zona se obtuvo el ensamblado?

Las zonas son descripciones de criterios de seguridad, como Internet, intranet, equipo local, etc., en función de la ubicación del código.

- ¿Cuál es el nombre seguro del ensamblado?

El nombre seguro es un identificador criptográficamente seguro proporcionado por el autor del ensamblado. Si bien no proporciona autenticación del autor, sí identifica exclusivamente el ensamblado y garantiza que no se haya alterado.

En función de las respuestas a estas preguntas y otros datos de evidencia, la directiva de seguridad puede calcular un conjunto de permisos apropiado para concedérselos al ensamblado. Se puede obtener evidencia de varios orígenes, incluidos CLR, el explorador, Microsoft ASP.NET y el *shell*, dependiendo del origen del código.

3.7.2. MODELO DE CONFIANZA

Controlado por directivas utilizando evidencia de código

Cuando se carga un ensamblado, el sistema de directivas de CLR determina los permisos que se concederán recopilando pruebas y evaluando esa evidencia en el contexto de la directiva de seguridad. A continuación, el sistema de directivas de CLR concede un conjunto de permisos al ensamblado de acuerdo con las pruebas evaluadas y las solicitudes de permisos que haya realizado el ensamblado. Un autor de ensamblados puede saber que el ensamblado funcionará correctamente sólo cuando se le concedan una serie de permisos o que el ensamblado no necesitará nunca determinados permisos. Estos requisitos adicionales se pueden pasar al sistema de directivas a través de una o más solicitudes de permisos específicos.

Dependiendo del tipo de solicitud de permisos, el sistema de directivas puede limitar más aún las concesiones al ensamblado (quitando los permisos innecesarios) o incluso rechazar la carga del ensamblado por completo (si la directiva no garantiza los permisos mínimos necesarios para ejecutar el ensamblado). A un ensamblado no se le pueden asignar nunca más permisos de los que le concedería el sistema de directivas si no hubiese solicitudes de permisos; las solicitudes sólo sirven para limitar más aún las concesiones.

La directiva de seguridad consta de una serie de grupos de código que contienen los permisos que se concederán en función de la evidencia. Los grupos de código pueden describir los permisos disponibles para ensamblados adquiridos de una zona de seguridad específica, o aquellos que están firmados por un publicador específico, etc. Aunque CLR se distribuye con un conjunto predeterminado de grupos de código (y permisos asociados), los administradores pueden personalizar este aspecto de seguridad de CLR para ajustarlo a sus necesidades particulares. Es importante recordar que cualquier cosa se puede presentar como evidencia, siempre y cuando la directiva de seguridad tenga un uso para ella, definiendo un grupo de código relacionado con esa evidencia.

El procedimiento para crear concesiones de permisos implica la evaluación de la evidencia para determinar los grupos de código que son aplicables a tres niveles diferentes: empresa, equipo y usuario. La directiva evalúa estos tres niveles en ese orden y crea un conjunto de permisos que es la intersección de los tres. Los administradores pueden marcar cualquier nivel como final, lo que evita la evaluación de las directivas de otros niveles. Esto permite, por ejemplo, que un administrador finalice la directiva para un ensamblado en el nivel de equipo y evite la aplicación de directivas del nivel de usuario a ese ensamblado.

Una vez que la directiva ha terminado, se crea un conjunto inicial de permisos. Los ensamblados pueden ajustar con precisión estas concesiones realizando solicitudes específicas en tres áreas:

- La primera es especificar un conjunto mínimo de permisos que el ensamblado debe tener para funcionar. Si estos permisos no están presentes, el ensamblado no se cargará y se iniciará una excepción.
- A continuación, se puede especificar un conjunto opcional de permisos. Aunque al ensamblado le parezca bien alguno de estos permisos, se cargaría si no están disponibles.
- Finalmente, ensamblados con un comportamiento especialmente correcto podrían rechazar permisos arriesgados que no necesitan. Estas tres opciones de ajuste se llevan a cabo como instrucciones declarativas en el momento de la carga.

En tiempo de ejecución, los permisos se evalúan en función de la ejecución del código. La figura de la derecha resume el proceso. Un ensamblado, "A3," proporciona su evidencia, junto con evidencia del *host*, al evaluador de la directiva. El evaluador considera también las solicitudes de permisos del ensamblado para crear una concesión de permisos, "G3." El ensamblado A2, a quien ha llamado el ensamblado A1, llama al ensamblado A3. Cuando el ensamblado A3 realiza una operación que desencadena una comprobación de seguridad, se examinan también las concesiones de permisos de A2 y A1 para garantizar que tienen los permisos solicitados por A3. En este proceso, denominado recorrido de la pila, se inspeccionan las concesiones de permisos de cada ensamblado de la pila para ver si el conjunto concedido contiene el permiso solicitado por la comprobación de seguridad. Si a cada ensamblado de la pila se le ha concedido el permiso solicitado por la comprobación de seguridad, la llamada progresa. Si algún ensamblado no obtiene el permiso solicitado, falla el recorrido de la pila y se inicia una excepción de seguridad.

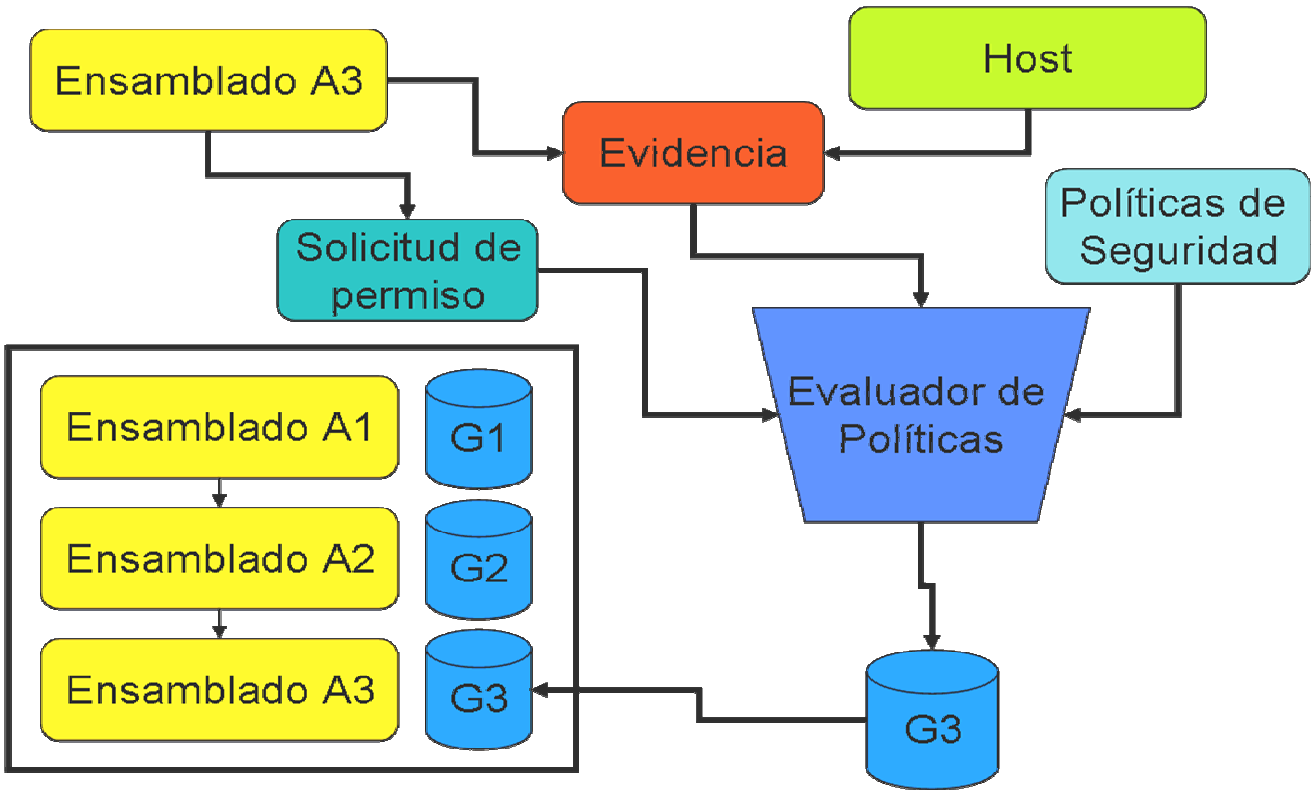


Figura 3.1 Manejo de ensamblados

En la Figura 3.1. El *host* y el ensamblado proporcionan evidencia al evaluador de la directiva, que utiliza la directiva de seguridad y las solicitudes de permisos para determinar las concesiones del ensamblado. A continuación, se utilizan las concesiones de permisos de varios componentes en ejecución para tomar decisiones de autorización.

El recorrido de la pila para la seguridad de acceso a código protege el código frente a ataques *luring*. En este ataque común, código malicioso engaña a código confiable para que haga algo que no puede hacer sólo, aprovechando los permisos del código bueno para fines malignos. Este tipo de ataque es extremadamente difícil de evitar para los desarrolladores, pero el recorrido de la pila garantiza que, si hay código de menor confianza implicado, los permisos disponibles se reducen a los del código de menor confianza.

El resultado es que se puede adquirir código de orígenes con distintos grados de confianza y se puede ejecutar con limitaciones apropiadas para cada contexto particular de la ejecución de ese código.

3.7.3. SEGURIDAD BASADA EN FUNCIONES

Algunas veces, es recomendable basarse en una identidad autenticada o una función asociada con el contexto de ejecución del código para las decisiones de autorización. Por ejemplo, el software financiero o empresarial puede hacer cumplir la directiva a través de lógica empresarial que evalúe información de funciones. El importe de una transacción financiera puede limitarse de acuerdo con la función del usuario que realiza la solicitud. Se puede permitir que los cajeros automáticos procesen solicitudes hasta un determinado importe y que importes superiores

requieran la función de un supervisor. *.NET Framework* proporciona servicios que permiten a las aplicaciones incorporar esta lógica fácilmente, creándola en torno al concepto de objetos *Identity* y Principal.

Se pueden asignar identidades al usuario que ha iniciado una sesión en el sistema operativo; o bien, las puede definir la aplicación. El objeto Principal correspondiente encapsula la identidad, junto con la información de función relacionada (por ejemplo, el "grupo" del usuario tal y como lo define el sistema operativo).

3.8. ADMINISTRADOR DE SEGURIDAD

El Administrador puede asignar derechos, definir políticas, etc. dado que en cualquier equipo donde se instala el motor de ejecución, el *.NET Framework*, también se instalan ciertas herramientas que admiten este tipo de seguridad. Lo que se muestra en la figura 3.2., es el árbol estructurado del agregado al *management console* (*snap in*) que corresponde al administrador de seguridad del *.NET Framework*.

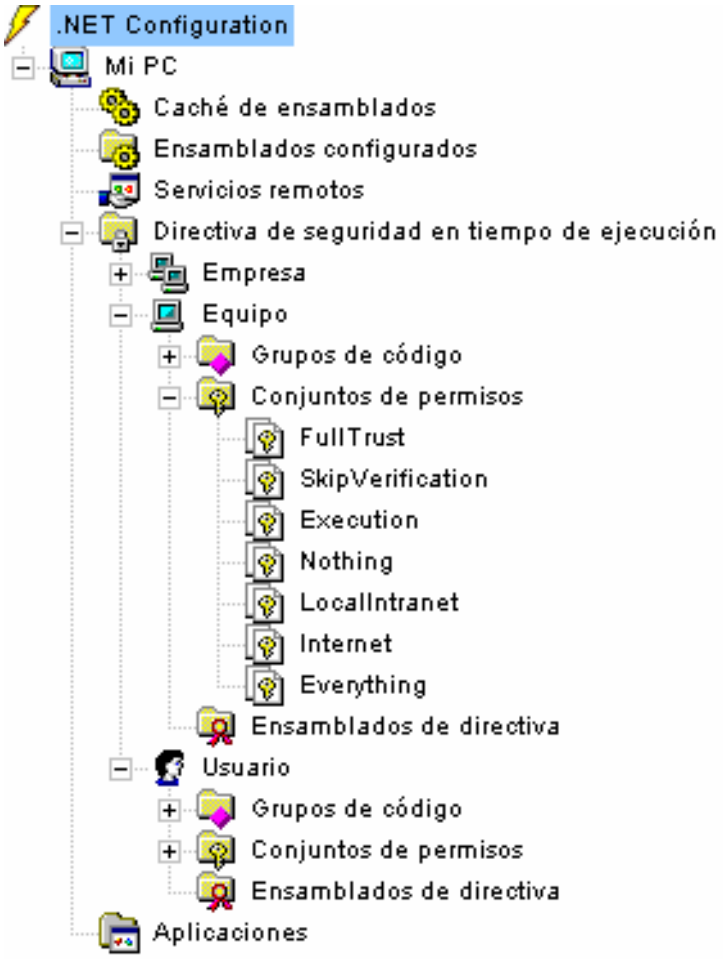


Figura 3.2 Árbol de Configuración

Existen una serie de elementos que un administrador pude definir estrictamente, pasando por configurar niveles de acceso, un grupo de código, etc., es posible apreciarlo en la figura 3.2.

El caché de ensamblado, conocido como *global assembly cache (GAC)* es donde se registran todos los componentes que se encuentran anotados dentro de la lista que generalmente se ubica en el subdirectorio *assembly* debajo del directorio de inhalación del sistema operativo, *windows* o *winnt*. El GAC tiene ciertas características, por ejemplo tiene un "nombre firme" (*strong name*), dado por una clave única de identificación, registro de su *hash code*, del número de versión y del autor de dicho código.

Solamente los elementos que cumplen estas condiciones pueden ser publicados dentro de la GAC o del caché de ensamblado. Este conjunto, cuando se termina la instalación del *.NET Framework*, esta dado por los elementos que configuran el conjunto de clases bases (***System.systemData***, etc.). Además un administrador puede configurar ensamblados de otras compañías u otros orígenes, en

la carpeta de ensamblados configurados. Los servicios remotos permiten configurar el acceso a este equipo por parte de otros a través de los canales de comunicación de *remounting*, etc.

En la carpeta de Directiva de seguridad en tiempo de ejecución, existen tres grupos: por Empresa, por Equipo y por usuario, los mismos que ya se han definido. Para cada uno de estos, por ejemplo equipo o usuario, que en la figura están abiertas, e tiene la oportunidad de definir grupos de código, conjuntos de permisos y ensamblados de directiva. En grupo de código se puede definir por ejemplo que los *assemblys* generados por una determinada empresa de desarrollo tienen ciertas características, es precisamente en grupo de código, donde un administrador autoriza o deniega por compañía que produce el código. Como segundo elemento, existen los conjuntos predefinidos de permisos donde es posible observar que existen un conjunto de permisos que autoriza absolutamente todo (*full trust*), un conjunto de permisos que escapa a la verificación, es decir que no realiza la verificación completa de la evidencia, solo da derechos de ejecución, no permite ningún derecho, solamente está autorizado lo que corresponde a la zona intranet, los derechos que lo definen para Internet, los derechos que se pueden definir para cualquier cosa.

De esa forma para cualquiera de los tres niveles, tanto en empresa, equipo o en usuario, un administrador puede asignar adecuadamente cada conjunto de permisos. Por ejemplo: MSDN una empresa que desarrolla código y el administrador de la compañía se entera que debe instalar un conjunto de *assemblies* de esta compañía, lo que hace es agregar en grupo de código ese conjunto y asignarle alguno de los conjuntos de permisos de acuerdo a lo que el considere que es necesario asignarle y además lo que el considere que no se le debe permitir a este código, si el es de código de confianza absoluta podría asignarle *full trust*.

Por definición en un equipo en el cual no solo se instala el *.Net Framework* sino además la herramienta de desarrollo, Visual Studio .NET, todos los *assemblie* que corresponden a la GAC y además el propio entorno de desarrollo, el compilador, el motor de ejecución de ASP .NET, etc., reciben como derechos *full trust*, si se modifica esta característica inmediatamente la herramienta de desarrollo deja de funcionar. Siempre que se haga una prueba de este tipo, acerca de seguridad, se debe tratar de hacer en un equipo que no tenga la herramienta de desarrollo configurada.

3.8.1. GRUPOS DE CÓDIGO

Un grupo de código es un agrupamiento lógico de código, en el cual se debe cumplir con una condición específica para que el código sea un miembro del grupo. Cualquier código que cumpla con la condición de membresía se incluye en el grupo. Cada grupo de código también está asociado con un conjunto de permisos nombrado. Los grupos de código pueden tener atributos, que afectan la forma en que se utiliza un grupo de código para definir las políticas. Los administradores configuran las políticas de seguridad al administrar los grupos de códigos.

El tiempo de ejecución utiliza evidencias que describen el código para determinar si se ha cumplido o no la condición de membresía de un grupo. Por ejemplo, si la condición de membresía del grupo de código es "Código que es del sitio Web www.microsoft.com", el tiempo de ejecución examina la evidencia para determinar si el código, de hecho, se origina de

www.microsoft.com. El tiempo de ejecución decide los permisos que se pueden otorgar al código al determinar a qué grupos de código pertenece el mismo y luego agregar los conjuntos de permisos asociados con los grupos de los que es miembro el código.

Cada tipo de política (máquina, usuario y dominio de la aplicación) se representa en una jerarquía de grupos de código. La raíz de cada jerarquía es el grupo que contiene todos los códigos. El grupo de "todos los códigos" tiene nodos hijo y los nodos hijo tienen hijos, etc. Si el código es miembro de un grupo de código padre, entonces el código puede ser un miembro de uno o más grupos de códigos hijos de ese grupo. Si el código no es miembro del grupo de códigos padre, no puede ser un miembro de cualquiera de los grupos de códigos que descienden de ese padre. Los grupos de código se pueden definir para que tengan los siguientes tipos de condiciones de membresía:

- Editor de software: Con base en claves públicas de una firma válida *Authenticode*
- Zona: Origen del código
- Nombre fijo: Basado en una firma criptográficamente fija
- Sitio Web: Origen del código, por ejemplo, www.microsoft.com o *.microsoft.com
- URL: Origen del código, incluyendo el comodín final, esto es "http://site/app/*"
- Análisis criptográfico: Basado en el análisis criptográfico MD5 ó SHA1
- Directorio de la aplicación: Basado en el directorio en el que se instala la aplicación
- Personalizado: Definido por una aplicación o un sistema

Los grupos de código pueden tener los siguientes atributos, que afectan la forma en que se determinan los conjuntos de permisos permitidos:

- Exclusivo: Si el código es un miembro de un grupo de código que está marcado como Exclusivo, el conjunto de permiso permitido para el nivel de política será el conjunto de permisos asociado con ese grupo de código. Cuando se toman en cuenta todos los niveles de políticas, no se permitirán más permisos en el código que los asociados con el grupo de código Exclusivo. Dentro de un nivel dado de política, el código puede ser un miembro de no más de un grupo de código que tenga un atributo Exclusivo.
- *LevelFinal*: No se consideran niveles de políticas por debajo del que contiene este grupo de código cuando se revisa la membresía del grupo de código y el otorgamiento de permisos; la política de la máquina es el nivel más alto de las políticas, seguido por las políticas del usuario y luego las políticas de dominio de la aplicación. Por ejemplo, si el atributo *LevelFinal* se establece para un grupo de código en las políticas de la máquina y algunos códigos corresponden con la condición de membresía de ese grupo de código, ni las políticas a nivel del usuario ni las políticas de dominio de la aplicación se aplicarían a ese código.

Un grupo de código se puede marcar tanto con el atributo Exclusivo como con el *LevelFinal*.

3.8.2. CONJUNTO DE PERMISOS

Un conjunto de permisos nombrados es un conjunto de permisos que los administradores pueden asociar con grupos de código. Un conjunto de permisos nombrados consiste en al menos un permiso y un nombre para una descripción del conjunto de permisos. Al asociar los conjuntos de permisos nombrados con los grupos de código, los administradores pueden establecer o modificar la política que determina los permisos que el tiempo de ejecución permite tener al código en esos grupos. Se puede asociar más de un grupo de código a un conjunto de permisos nombrados.

El tiempo de ejecución proporciona varios conjuntos de permisos nombrados integrados:

☑ *Nothing*

Sin permisos (no se puede ejecutar)

☑ *Execution*

Sólo permisos para ejecutar, pero no hay permisos que permitan el uso de recursos protegidos

☑ Internet

El conjunto de permisos de políticas predeterminado adecuado para contenidos de origen desconocido

☑ LocalIntranet

El conjunto de permisos de políticas predeterminado dentro de una empresa

☑ *Everything*

Todos los permisos estándar (integrados) a excepción de los permisos para saltar la verificación

☑ *FullTrust*

Acceso total a todos los recursos que están protegidos por permisos y pueden estar sin restricción

De estos conjuntos integrados de permisos nombrados, sólo Internet, LocalInternet y *Everything* se pueden modificar. Modificar un conjunto de permisos nombrados existente afecta los permisos que puede otorgar el tiempo de ejecución a todos los códigos que pertenecen a un grupo de código asociado con ese conjunto de permisos.

Los administradores pueden definir conjuntos personalizados de permisos nombrados siempre que los nombres no entren en conflicto con los nombres de los conjuntos integrados de permisos nombrados. Los conjuntos de permisos nombrados no pueden contener permisos de identidad.

3.9. MEDIDAS ADICIONALES DE SEGURIDAD

Existen algunas medidas de seguridad adicionales proporcionadas por .NET:

Seguridad de tipo: La seguridad de tipo contribuye para lograr una confiabilidad total del sistema.

Servicios de criptografía: Están disponibles Servicios de criptografía para los desarrolladores .NET para su uso en sus aplicaciones así como para descubrir y establecer configuraciones de seguridad junto con el código de ejecución.

Confusión de código: La Confusión de código protege la propiedad intelectual de los desarrolladores.

3.9.1. CRIPTOGRAFÍA

.NET Framework proporciona un conjunto de objetos criptográficos que admiten cifrado, firmas digitales, algoritmos *hash* y generación de números aleatorios, implementados a través de algoritmos bien conocidos, como RSA, DSA, Rijndael/AES, Triple DES, DES y RC2, así como los algoritmos *hash* MD5, SHA1, SHA-256, SHA-384 y SHA-512. Se admite también la especificación de firma digital XML (*XML Digital Signature*), que desarrollan Internet *Engineering Task Force* (IETF) y *World Wide Web Consortium* (W3C). *.NET Framework* utiliza objetos criptográficos para admitir servicios internos. Los objetos están disponibles también como código administrado para los desarrolladores que requieran compatibilidad criptográfica.

.NET Framework proporciona un conjunto rico de clases que se puede usar para proteger y firmar datos sensibles usando criptografía. Estas clases están en el espacio de nombre *System.Security.Cryptography*. Las funciones de las clases incluyen las siguientes cuatro áreas:

Encriptación y desencriptación

Permite la encriptación y desencriptación de datos al utilizar un algoritmo simétrico o asimétrico específico. Los algoritmos simétricos soportados son DES, Triple DES, RC2 y Rijndael. Los algoritmos asimétricos soportados son RSA y DSA.

Generación y verificación de firmas digitales

Permiten la generación y verificación de firmas digitales para datos basados en un algoritmo específico. Los algoritmos soportados incluyen RSA y DSA.

Generación de análisis

Permite la generación de valores de análisis para datos. Los algoritmos de análisis soportados son MD5, SHA1, SHA256, SHA384 y SHA512.

Firmas XML

Realmente un subconjunto de las firmas digitales, las clases de Firma XML implementan las recomendaciones propuestas ante el W3C sobre Firmas XML. Estas clases residen en el espacio de nombre *System.Security.Cryptography.Xml.*

Existen dos tipos principales de algoritmos criptográficos: simétrico y asimétrico.

Simétrico: Los algoritmos criptográficos simétricos tienen una sola clave secreta que se usa tanto para la encriptación como para la desencriptación. El uso de un algoritmo simétrico requiere que sólo el emisor y el receptor conozcan la clave secreta. DES_CSP, RC2_CSP y TripleDES_CSP son implementaciones de algoritmos simétricos.

Asimétrico: Los algoritmos criptográficos asimétricos, también conocidos como algoritmos de clave pública, requieren que cada entidad mantenga un par de claves relacionadas: una clave privada y una pública. Ambas claves son únicas para cada entidad.

La clave pública puede estar disponible a todos y se usa para codificar datos que se enviarán al receptor.

El receptor deberá mantener privada la clave privada y se usa para descodificar mensajes codificados utilizando la clave pública del receptor. RSA_CP es una implementación de un algoritmo de clave pública.

3.9.2. OFUSCACIÓN

Tener metadatos ricos y un lenguaje intermedio de alto nivel es clave para la plataforma .NET. Esto hace que la información esté disponible en cualquier momento, en cualquier lugar o en cualquier dispositivo. Esto también facilita la implementación y la aplicación del tiempo de ejecución de la seguridad de tipo, así como una interoperabilidad verdadera entre lenguajes. Lo que es más, forma la base del mecanismo de seguridad .NET.

Por otro lado, también fomenta dos problemas básicos de privacidad del código escrito por los desarrolladores: los códigos que se roban o que se alteran. El uso de las técnicas de confusión de código protege al código contra alteración o robo. La confusión de código significa esencialmente alterar el código para hacerlo ilegible y difícil de entender. La herramienta ofuscadora usa ILDASM para transformar MSIL en algo no legible por humanos.

3.9.3. CÓDIGO DE TIPO SEGURO

El código con tipo seguro accede sólo a las ubicaciones de memoria a las que tiene autorización. Por ejemplo, no puede leer valores de otros campos privados del objeto. El código de tipo seguro accede a tipos sólo en maneras bien definidas y permisibles.

Durante la compilación Justo a tiempo (JIT), un proceso opcional de verificación examina *Microsoft Intermediate Language* (MSIL) en un intento por verificar que MSIL sea seguro en su tipo. Este proceso se salta si el código tiene permiso para pasar la verificación.

Aunque la verificación de seguridad del tipo no es obligatoria, juega un rol crucial en el aislamiento de ensambles y en la aplicación de la seguridad. Cuando los códigos son seguros en su tipo, el tiempo de ejecución puede aislar completamente los ensambles. Este aislamiento ayuda a asegurar que los ensambles no puedan afectarse adversamente entre sí y aumentan la confiabilidad de la aplicación. Los componentes seguros en su tipo se pueden ejecutar de manera segura en el mismo proceso aún cuando sean confiables en distintos niveles. Cuando el código no es seguro en su tipo, el tiempo de ejecución no puede evitar invocar al código nativo (no

administrado) para realizar operaciones maliciosas. Sin embargo, si el código es seguro en su tipo, el mecanismo de aplicación de la seguridad de tiempo de ejecución puede asegurar que no acceda a código nativo a menos que tenga permiso para hacerlo.

3.10. CONCLUSIÓN

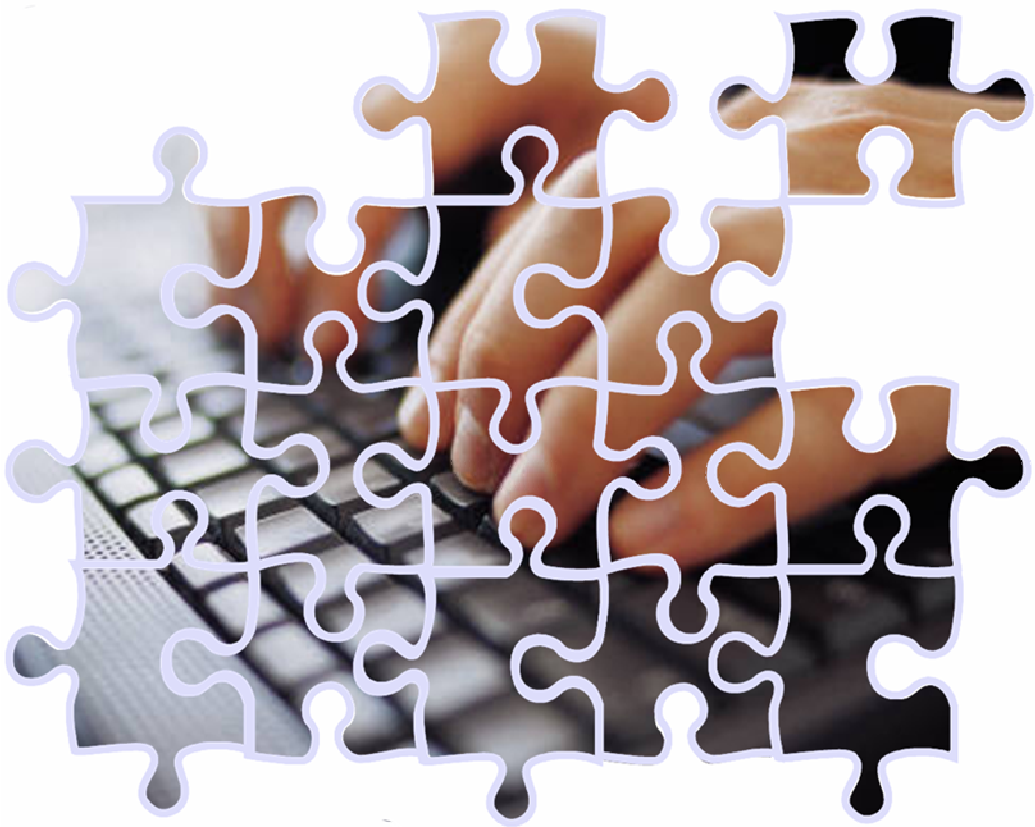
En este tercer capítulo, se aprenden los mecanismos se que proporcionan como parte de la seguridad en el .NET Framework para proteger los recursos contra fuentes y ataques no autorizados. Es posible identificar diversos conceptos centrales de la seguridad en .NET. También se aprende sobre las distintas herramientas de seguridad, considerando que dicho tema será retomado en un capítulo futuro. Se proporcionan conocimientos de las distintas medidas de seguridad proporcionadas por .NET Framework.

El capítulo permite formar un criterio respecto a lo que se tiene y lo que se necesita cuando se habla de implementar seguridad en las aplicaciones. Muchas personas que lean este capítulo pueden cuestionarse sobre el contenido orientado a los administradores y no para los programadores, pero como lo digo en la introducción de este trabajo, la tesis está orientada a los programadores pero se apoya de los beneficios que proporcionan los administradores. Desde este punto de vista es necesario comenzar por el servidor de la aplicación para establecer los permisos y reglas necesarios que eviten problemas administrativos que afecten el código y en consecuencia el óptimo proceso y desempeño del sistema.

Como Administrador, del Servidor, sistema o la aplicación en .NET, este capítulo es de utilidad para crear juicios para ejercer la autoridad o mandato sobre los sistemas y sobre las personas que lo utilicen, proporcionando opciones para ordenar, disponer, organizar o distribuir un servicio dentro de los sistemas y aplicaciones. Como desarrollador, toda esta información ayuda a proporcionar la seguridad necesaria que el sistema requiere para proteger las aplicaciones basadas en .NET contribuyendo establecer hasta donde se encuentra los límite de la seguridad del sistema operativo y los limites del desarrollo y programación segura, así se podrán complementar los sistemas en la parte administrativa y en la programación en aspectos de seguridad informática.

Capítulo 4

Características y elementos del entorno de desarrollo con Visual Basic .NET



4. CARACTERÍSTICAS Y ELEMENTOS DEL ENTORNO DE DESARROLLO CON VISUAL BASIC .NET

El papel de *VB* dentro de *Windows* DNA[9] (*Distributed interNet Architecture*) ha sido fundamentalmente, el de la escritura de componentes para su uso por parte de las páginas ASP de una aplicación *web*; de hecho, es el lenguaje preferido para el desarrollo de componentes debido a su ya larga tradición como lenguaje sencillo y de fácil manejo.

Microsoft hizo un intento de dotar de un mayor número de características a *Visual Basic* para que pudiera convertirse en una herramienta de desarrollo integral para Internet; para ello, incorporó las *Web Classes*, los documentos y controles *ActiveX*, aunque ninguno de ellos obtuvo plena aceptación.

Por un lado, las *Web Classes* tenían el complejo modelo de programación, mientras que los documentos *ActiveX* arrojaban unos pobres rendimientos de ejecución. Con respecto a los controles *ActiveX*, necesitaban de cierto proceso de instalación por parte del servidor, lo que los hacía en muchas situaciones poco operativas. Estas circunstancias han impedido que *VB* pudiera convertirse en la herramienta de desarrollo para Internet de *Microsoft*. Otros factores decisivos que han limitado la plena entrada de *VB* en la programación *web* han sido la falta de capacidades multihebra, inexistencia de un interfaz de usuario específico para aplicaciones *web*, falta de herencia y otras características orientadas a objeto, escasa integración con otros lenguajes, deficiente gestión de errores, etc., aspectos todos, solucionados en *VB.NET*.

VB.NET aporta un buen número de características que muchos programadores de *VB* demandan desde hace largo tiempo. En cierto modo, algunas de estas incorporaciones es posible encontrarlas en la plataforma *.NET*, ya que al integrar *VB* dentro del conjunto de lenguajes de *.NET Framework*, dichos cambios han sido necesarios, no ya porque los necesitara *VB*, sino porque eran requisitos derivados de la propia arquitectura de *.NET*.

Entre las novedades aportadas por *VB.NET* se tienen plenas capacidades de orientación a objetos (*Full-OOP*), incluyendo herencia; *Windows Forms* o la nueva generación de formularios para aplicaciones *Windows*; soporte nativo de *XML*; gestión de errores estructurada; un modelo de objetos para acceso a datos más potente con *ADO.NET*; posibilidad de crear aplicaciones de consola (ventana *MS-DOS*); programación para Internet mediante *Web Forms*; un entorno de desarrollo común a todas las herramientas de *.NET*, etc.

Pero todas las mejoras efectuadas en *VB.NET*, han hecho que esta herramienta sufra una renovación tan profunda, que marcan un punto de inflexión importante, haciendo que muchos programadores opinen que es un nuevo lenguaje, más que una nueva versión.

---

[9] Los herméticos y poco flexibles modelos de programación actuales, han evolucionado desde un esquema que integra diversas tecnologías como COM, ASP, ADO, etc., la mayor parte de ellas no pensadas inicialmente para ser ejecutadas en la Red, o que en el caso de ser diseñadas para Internet, arrastran elementos que no estaban pensados para funcionar en la web. Todos estos elementos, conforman la arquitectura Windows DNA (Distributed interNet Architecture), que hasta la actualidad ha sido el modelo de programación para Internet propugnado por Microsoft.
Este modelo ha sido dejado a un lado para dar paso a .NET; lo que no supone una evolución de la actual arquitectura Windows DNA, sino que por el contrario, significa el nuevo comienzo de una arquitectura pensada para la Red.

VB.NET proporciona una utilidad de migración de aplicaciones creadas con versiones anteriores de *VB* que según las pruebas realizadas es capaz de migrar hasta el 95% del código de una aplicación creada en *VB6*.

Y lo que es más importante, no es obligatoria la migración de una aplicación escrita por ejemplo en *VB6*; se pueden seguir ejecutando tales programas dentro de *.NET Framework*, con el inconveniente de que al no ser código gestionado por el entorno de *.NET* no podrá aprovecharse de sus ventajas.

Para la programación en Internet y todo el nuevo espectro de servicios, fue necesario integrar *VB* como lenguaje del entorno *.NET*, pero los lenguajes que formen parte de esta plataforma están obligados a cumplir una serie de requisitos, no porque lo pueda necesitar el lenguaje, sino porque es la plataforma la que obliga a ello para poder sacar partido de todas las ventajas de *.NET*.

Mirando hacia anteriores cambios de versiones, se puede comprobar que desde *VB4*, todos los cambios han sido en buena medida profundos, para poder adaptarse a las necesidades de los programas en cada momento. Bien es cierto, que esta versión incorpora un cambio más traumático que las otras, pero si se comparan las nuevas funcionalidades y potencia que obtendrán las aplicaciones, suponen que la inversión efectuada en adaptarse merecerá la pena.

4.1. EL ENTORNO DE DESARROLLO INTEGRADO (IDE), DE VISUAL STUDIO .NET

El disponer en la actualidad de un entorno de desarrollo (*Integrated Development Environment* o *IDE*) para una herramienta de programación, puede parecer algo natural o incluso elemental para el propio lenguaje. Ello hace que en ocasiones no se conceda la importancia que realmente tiene a este aspecto del desarrollo.

Las cosas no han sido siempre tan fáciles en este sentido, ya que en tiempos no muy lejanos, los programadores debían de realizar de una forma artesanal, todos los pasos en la creación de una aplicación.

Cuando utilizo el término artesanal, me refiero a que el programador debía escribir el código fuente en un editor y diseñar el interfaz con un programa generador de pantallas. Después ejecutaba otro programa que contenía el compilador, sobre el código fuente, para obtener los módulos compilados. Finalmente, debía de enlazar los módulos compilados con las librerías del lenguaje y terceras librerías de utilidades si era necesario, para obtener el ejecutable final de la aplicación. Todo se hacía a base de pasos independientes.

Tal dispersión de elementos a la hora de desarrollar resultaba incómoda en muchas ocasiones, por lo que los fabricantes de lenguajes de programación, comenzaron a incluir en sus productos, además del propio compilador, enlazador y librerías, una aplicación que permitía al programador realizar todas las fases del desarrollo: los editores de código, diseñadores visuales, compilador, etc., estaban incluidos en el mismo entorno a modo de escritorio de trabajo o taller de programación, se trataba de los primeros *IDE*.

En lo que respecta a los lenguajes de *Microsoft*, los programadores disponen desde hace ya tiempo del *IDE* de *Visual Studio*.

Los diseñadores del entorno de programación de *Microsoft*, sobre todo desde su versión 5 han tenido un objetivo principal: hacer que el *IDE* de cada uno de los lenguajes sea lo más similar posible al resto, de forma que el desarrollo con varios lenguajes no suponga un cambio traumático en el entorno de trabajo.

Esto quiere decir que, por ejemplo, un programador que debe desarrollar aplicaciones tanto en *Visual Basic* como en *Visual C++,* cada vez que abra el entorno de trabajo de alguna de estas herramientas, va a encontrar, salvo las particularidades impuestas por el lenguaje, un *IDE* casi igual, lo que evita un entrenamiento por separado para cada lenguaje y una mayor productividad, al acceder a los aspectos comunes de igual manera.

A pesar de todas las similitudes, y hasta la versión 6, cada lenguaje seguía teniendo su propio *IDE*.

Visual Studio .NET, es el primer paso de la total integración con la llegada de la tecnología *.NET*, el panorama ha cambiado sustancialmente, ya que al estar todos los lenguajes bajo el abrigo de un entorno de ejecución común, se ha podido desarrollar también un *IDE* común.

Ya no debe elegirse en primer lugar el lenguaje y abrir su IDE particular. Todo lo contrario, ahora debe iniciarse el *IDE* de *Visual Studio .NET* y después, elegir el lenguaje con el que trabajara. Esto materializa la idea de disponer de un *IDE* único para diversos lenguajes. Este concepto es además extensible, ya que al ser *.NET Framework* una plataforma multilenguaje, los lenguajes desarrollados por terceros fabricantes también podrán engrosar la lista de los disponibles a través del *IDE*.

La descripción del *IDE* se abordará a lo largo de los distintos temas del proyecto que así lo requieran, ya que ciertos aspectos específicos del *IDE* es recomendable describirlos en el tema con el que guardan una mayor relación.

4.2. PÁGINA DE INICIO VS .NET

La página de inicio de *VS.NET*, se muestra en la página de inicio del *IDE*. Ver Figura 4.1.

Desde esta página se realiza una primera configuración del entorno, ya que al hacer clic en el vínculo *Mi Perfil (Mi Profile)*, situado en pestaña derecha, accede a una pantalla en la que es posible establecer una configuración adaptada al lenguaje con el que se programará. Ver Figura 4.2.

Es posible configurar el perfil general para adaptar a nuestra comodidad la totalidad del *IDE*, o bien hacerlo sólo sobre ciertos elementos como el teclado, diseño de ventana, etc. de teclado adaptado a un perfil de programador de *Visual Basic 6*. El resto de elementos se dejarán como estaban por defecto, ya que si se adaptaran la totalidad del *IDE* al perfil de *VB*, se expandirán muchas de las ventanas ocultas adicionales, dejando poco espacio en la ventana principal de trabajo. Para la configuración del perfil del programador, se hará clic en la pestaña superior, *Mi Profile* para volver al punto inicial, en el que se creará un nuevo proyecto de *VB.NET*, de la forma

explicada en el tema La primera aplicación, que nos servirá para hacer las pruebas sobre los diferentes aspectos del *IDE*.

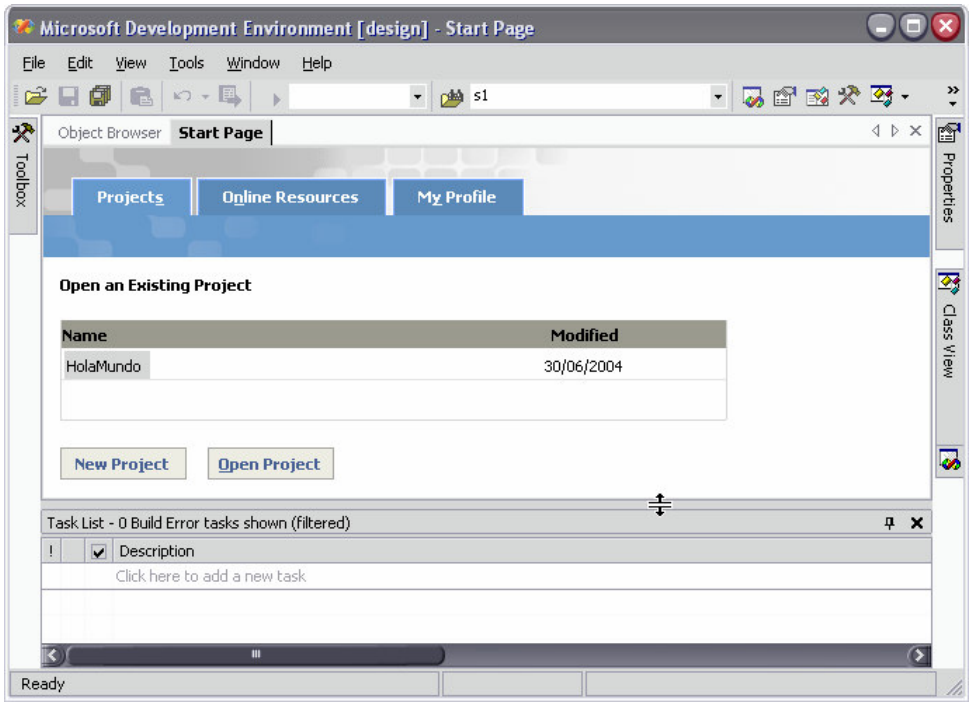


Figura 4.1. Página de inicio de VS.NET.

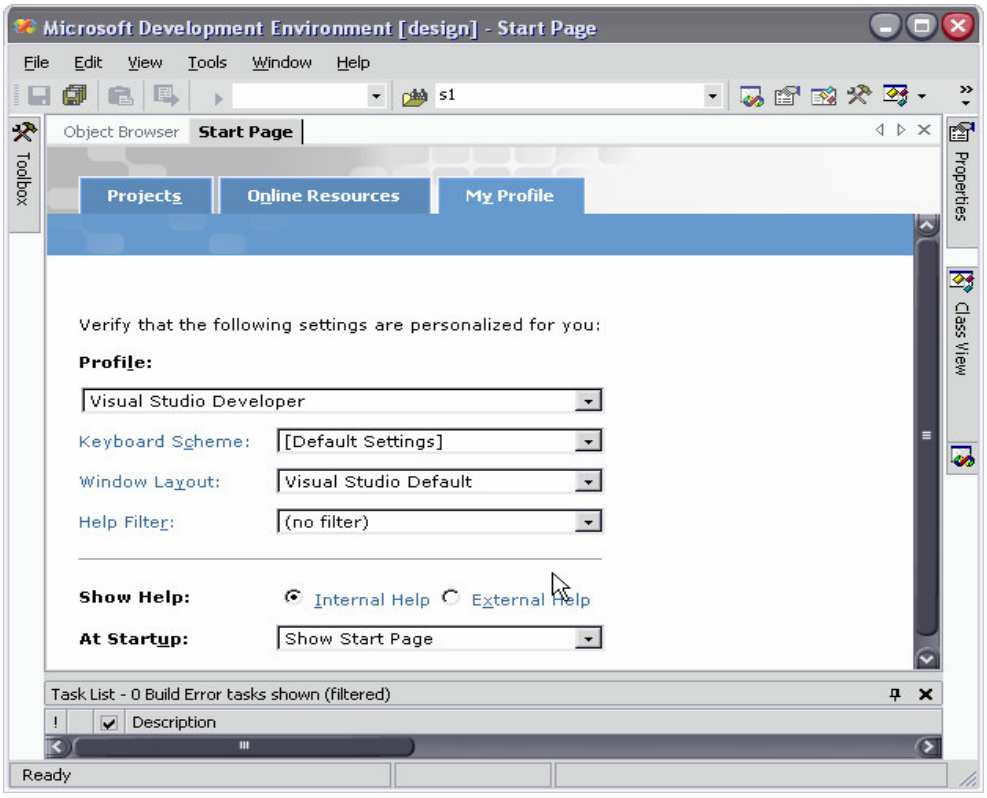


Figura 4.2. Estableciendo un perfil de programador.

4.3. PRINCIPALES ELEMENTOS DEL ENTORNO DE TRABAJO

Una vez abierto un proyecto en el *IDE*, los elementos básicos para nuestra tarea habitual de desarrollo se muestran en la Figura 4.3.

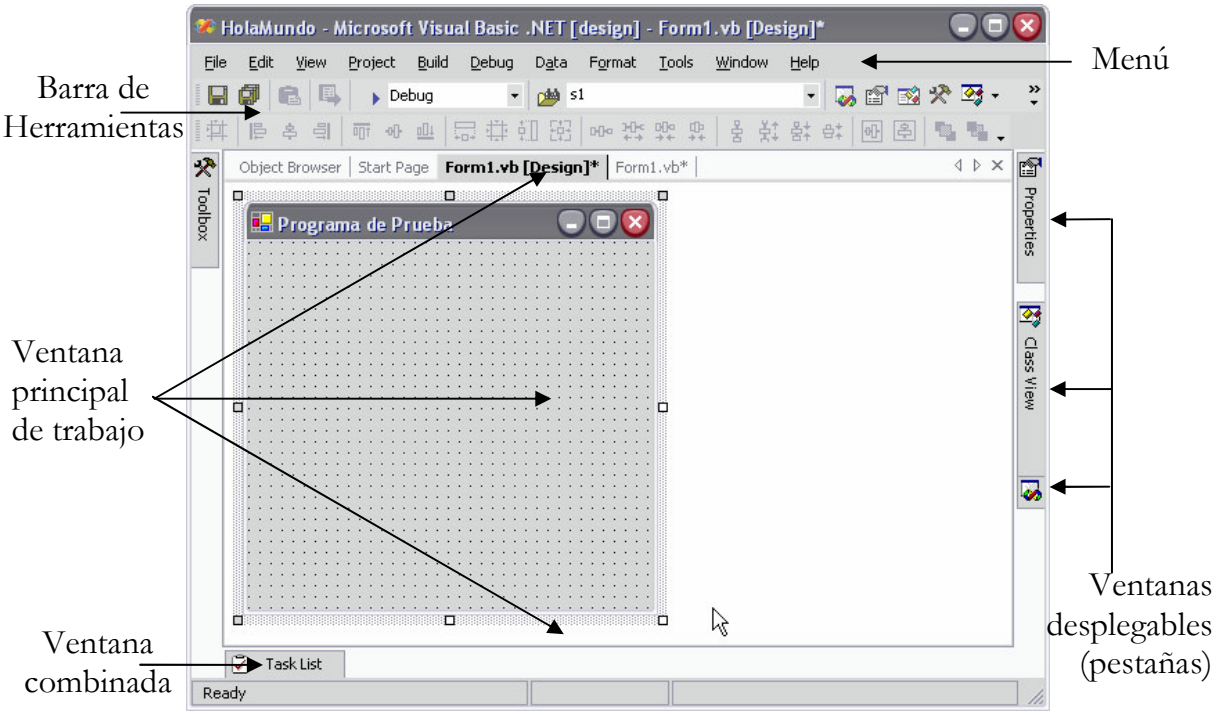


Figura 4.3. Elementos principales del IDE.

4.3.1. VENTANA PRINCIPAL DE TRABAJO

De forma predominante y ocupando la mayor parte del *IDE*, está la ventana o zona principal de trabajo. Esta ventana principal contiene todos los editores de código y diseñadores que se van abriendo, organizados con base en unas fichas o pestañas, que nos permiten trasladarnos de uno a otro cómodamente.

Para comprobarlo, añadir un nuevo formulario al proyecto mediante la opción de menú *Proyecto + Agregar formulario de Windows*, y un módulo de código con la opción *Proyecto + Agregar módulo*. En ambos casos, dejando a estos elementos los nombres que asigna por defecto el *IDE*. Si además, en los dos diseñadores de formulario que se debe tener actualmente, seleccionando la opción Ver + Código, se añadirán los correspondientes editores de código a la ventana, que mostrará un aspecto similar al de la Figura 4.4.

Es posible cambiar de diseñador con un simple clic sobre su ficha correspondiente o la combinación de teclas [*CTRL.+TAB*]. Cuando la ventana se llene totalmente de fichas, es posible desplazarse entre las mismas mediante los dos iconos de la parte superior derecha que muestra unas flechas de dirección. Si se quiere cerrar alguna de las fichas, puede hacerse igualmente pulsando el icono de cierre de esta ventana o la combinación [*CTRL.+F4*].

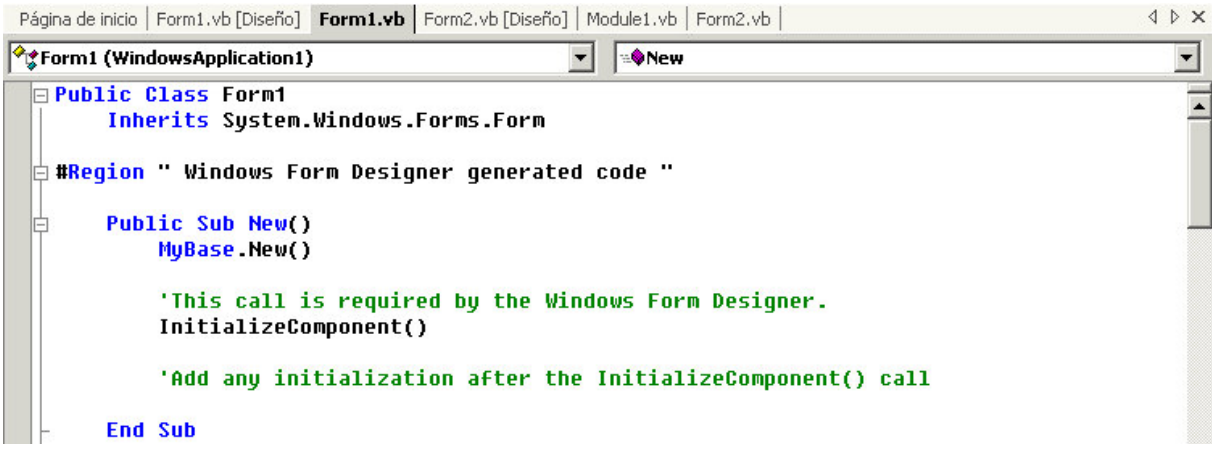


Figura 4.4. Ventana principal de trabajo mostrando varias fichas.

Para cambiar la posición de las fichas, debe hacer clic sobre la ficha que quiera cambiar y arrastrar hacia una nueva posición.

La organización en fichas, supone un importante cambio en el modo de trabajo respecto a *VB6*, que aporta una mayor comodidad a la hora de tener abiertos simultáneamente diversos editores y diseñadores. Sin embargo, si el programador se siente más confortable con la antigua organización basada en ventanas, puede cambiar a dicha configuración seleccionando la opción de menú *Herramientas + Opciones*, que mostrará la ventana de opciones de configuración del *IDE*. Ver Figura 4.5.

En el caso de no estar posicionado inicialmente, debe seleccionarse en la parte izquierda de esta ventana, la carpeta Entorno y su apartado General. A continuación pulsar sobre la opción Entorno *MDI* y pulsar *Aceptar*, ver Figura 4.6. Se debe tener en cuenta, que los cambios no se reflejarán hasta la próxima vez que se inicie *VS.NET*.

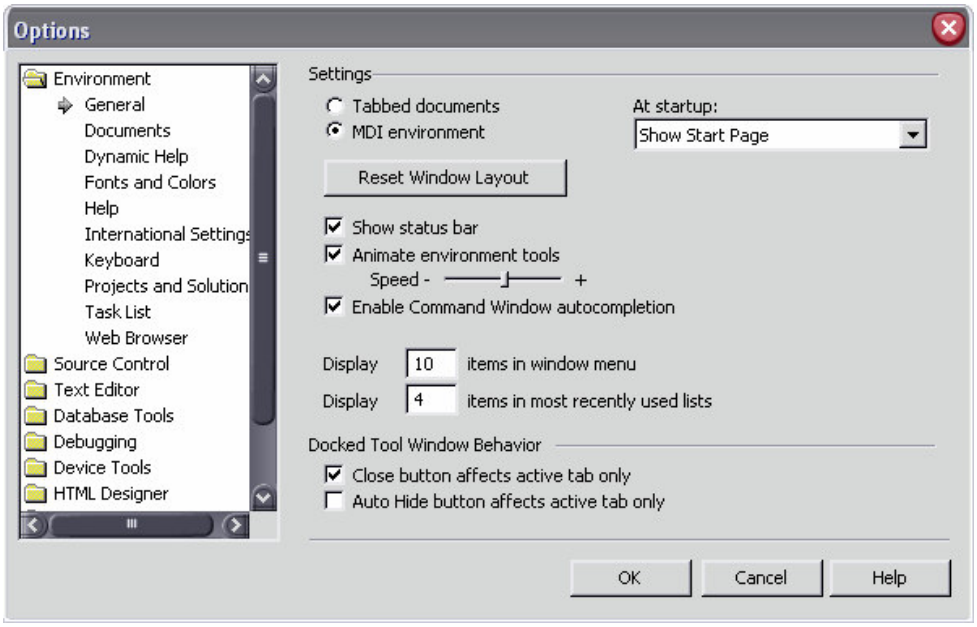


Figura 4.5. Opciones de configuración del IDE.

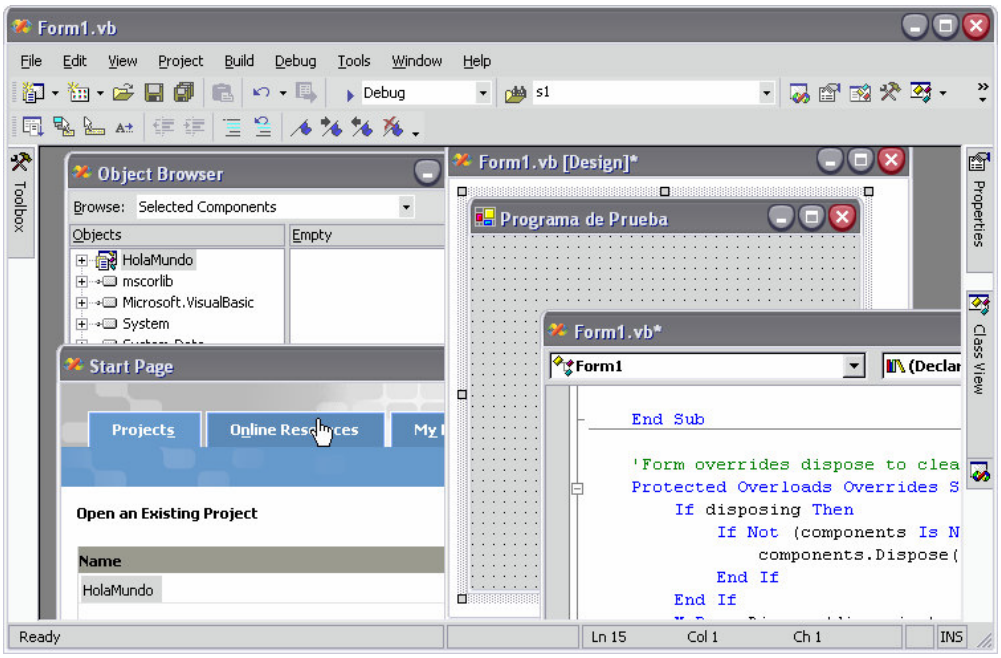


Figura 4.6. Organización del IDE en ventanas.

Es posible crear una nueva ventana para albergar fichas, usando la opción de menú Ventana + Nuevo grupo de fichas horizontal, o la opción Ventana + Nuevo grupo de fichas vertical, a la que se podrán mover fichas desde la ventana original con sólo arrastrar y soltar. Ver Figura 4.7.

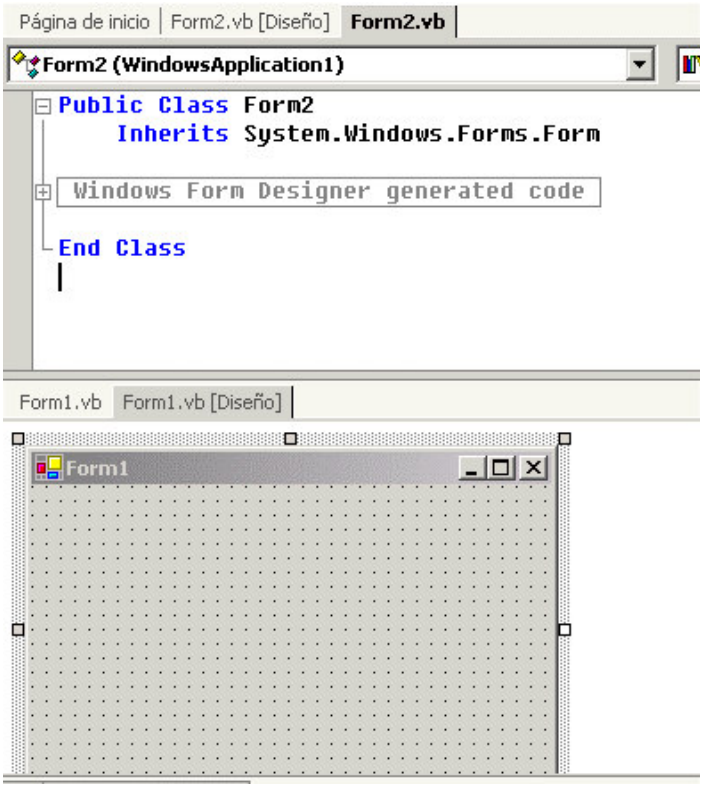


Figura 4.7. Fichas organizadas en grupos diferentes.

4.3.2. VENTANA DE PROPIEDADES

Cuando se diseña un formulario, esta ventana muestra las propiedades del objeto seleccionado en el diseñador: bien un control o el propio formulario. La Figura 4.8 muestra esta ventana indicando sus elementos principales.

Las propiedades se organizan en dos columnas: una contiene los nombres de las propiedades y otra sus valores. Las propiedades compuestas de varios miembros, incluyen en el lateral izquierdo un signo + para expandirlos.

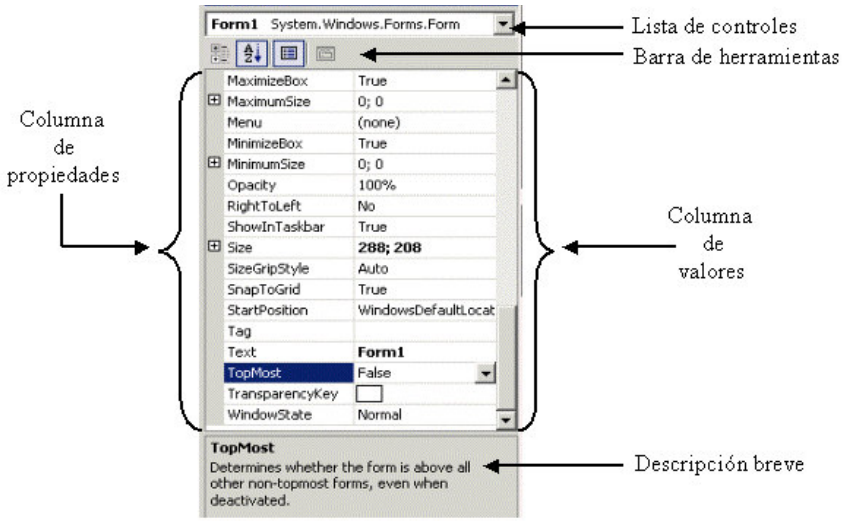


Figura 4.8. Ventana de propiedades de VS.NET.

Ciertas propiedades contienen una lista de valores, que pode abrirse con el botón que figura en el valor de la propiedad. Ver Figura 4.9.

Figura 4.9. Propiedad con lista de valores

Existen otras propiedades cuyo valor es seleccionado mediante una caja de diálogo. En esta propiedad, se muestra en su valor, un botón con puntos suspensivos indicando que debe pulsarse para modificar su valor. Ver Figura 4.10.

Figura 4.10. Propiedad modificable mediante caja de diálogo.

Al hacer clic sobre un control del formulario pasa a continuación a ver sus propiedades, o bien pude elegirse el control de la lista desplegable de controles. La Figura 4.11 muestra esta lista con el propio formulario y varios controles adicionales.

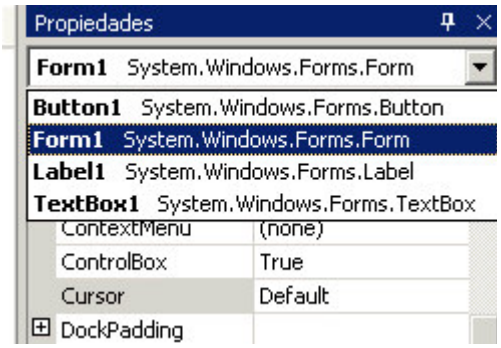


Figura 4.11. Lista de controles de la ventana de propiedades.

Los dos primeros botones de la barra de herramientas de esta ventana, permiten, ordenar las propiedades por categoría o en forma alfabética, respectivamente. Mientras que en la parte inferior, se visualiza una descripción resumida de la propiedad que se tenga seleccionada. Si no se desea ver dicha descripción, se puede hacer clic derecho sobre la ventana, seleccionando la opción de menú *Descripción*.

4.4. EL LENGUAJE DE VISUAL BASIC .NET

Desde la llegada de interfaces visuales basados en ventanas, los productos para el desarrollo de aplicaciones, han ido incorporando paulatinamente un mayor número de ayudas y asistentes que hacen a veces olvidar el verdadero soporte de aquello que se utiliza para programar: el lenguaje.

Cualquier persona con un nivel de usuario medio/avanzado, y sin una base sólida de programación, conociendo un pequeño conjunto de instrucciones del lenguaje, podía escribir programas, dada la gran cantidad de utilidades y apoyo proporcionados por el entorno de programación.

Esto es claramente contraproducente, puesto que no se aprovecha todo el potencial que ofrece la herramienta al desconocer su elemento más importante, el lenguaje, del cual parten el resto de aspectos del producto.

En el caso de *VB.NET* este aspecto se acentúa, debido al gran trabajo realizado en dotar al lenguaje de un elevado número de características que estaban siendo reclamadas desde hace ya tiempo por la comunidad de programadores.

Estas mejoras no han sido realizadas exclusivamente para *VB.NET*, ya que este lenguaje se ha beneficiado indirectamente de ellas, puesto que al compartir ahora todos los lenguajes de *.NET Framework* una especificación común, *Visual Basic* como lenguaje, ha necesitado ser adaptado para cumplir con dicha normativa, beneficiándose así de la potencia incluida en todas las características de la plataforma .NET.

Dado el vasto número de características disponibles, se realizará una exposición elemental de cada una, ampliando donde sea necesario en posteriores capítulos.

4.4.1. ESTRUCTURA DE UN PROGRAMA VB.NET

La Figura 4.12 muestra de una manera gráfica la estructura de una aplicación, en forma de niveles.

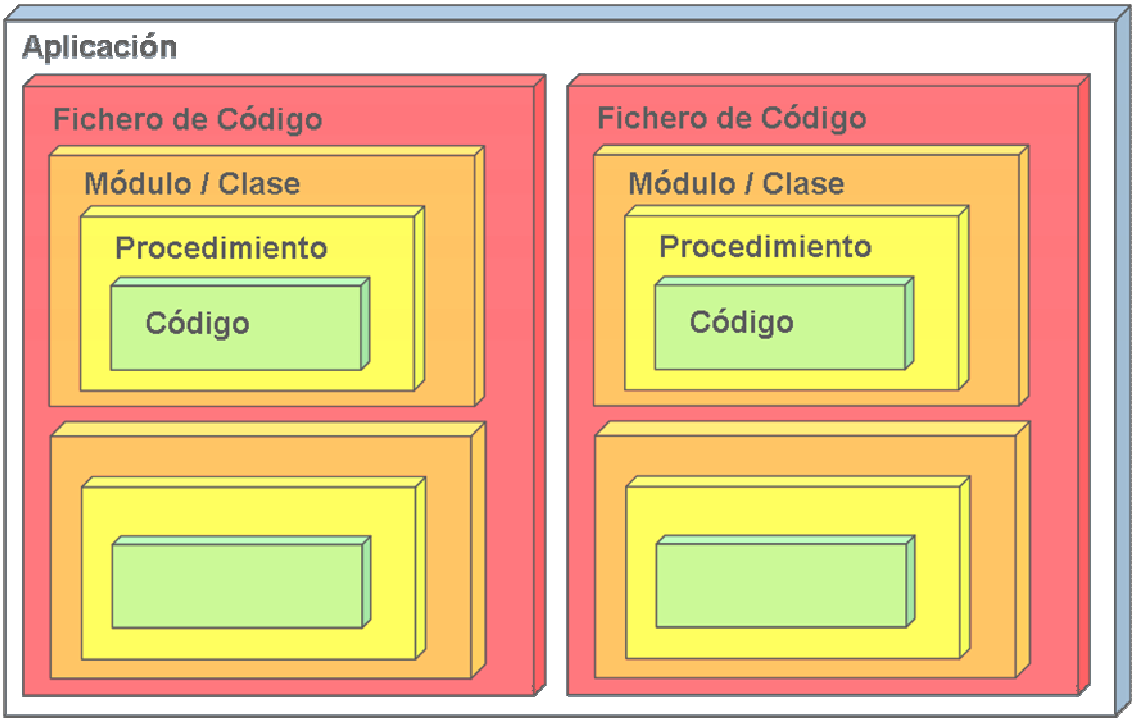


Figura 4.12. Estructura en niveles de una aplicación *VB.NET*.

Como muestra el diagrama, una aplicación está formada por uno o más ficheros de código, que a su vez contienen módulos de código o clases, dentro de los que se escriben procedimientos que son los elementos que contienen el código base.

Cuando se crea una aplicación usando *VS.NET*, es el propio *IDE* se encarga de crear la estructura básica del programa, crea un fichero de código conteniendo un módulo que tiene el procedimiento de entrada, sólo falta el código del programador.

Todos los elementos que componen una aplicación *VB.NET*, son organizados por *VS.NET* bajo el concepto de proyecto. Un proyecto aglutina los ficheros de código de la aplicación, recursos, referencias a clases globales de la plataforma *.NET*, etc.

De manera implícita, cada vez que se crea un nuevo proyecto utilizando el *IDE*, dicho proyecto es al mismo tiempo un ensamblado de ámbito privado, por lo que también puede referirse a una aplicación utilizando ambos términos: proyecto o ensamblado.

4.4.2. MAIN( ) COMO PROCEDIMIENTO DE ENTRADA AL PROGRAMA

Todo programa necesita una rutina o procedimiento de entrada, que sea el primero que se ejecute. En *VB.NET* ese procedimiento recibe el nombre especial ***Main ( )***, y debe estar contenido dentro de un módulo de código, como muestra el Código fuente 4.A.

```
Module Module1
    Sub Main()
    End Sub
End Module
```

Código fuente 4.A.

En el caso de una aplicación de consola creada desde *VS.NET*, se crea un módulo de forma automática que contiene un procedimiento *Main ( )* vacío. Dentro de este procedimiento se escribirá el código de los próximos ejemplos.

4.4.3. VARIABLES

Una variable es un identificador del programa que guarda un valor que puede ser modificado durante el transcurso de dicha aplicación.

4.4.3.1. DECLARACIÓN Y DENOMINACIÓN

La **declaración** de una variable es el proceso por el cual se comunica al compilador que se creará una nueva variable en el programa.

Para declarar una variable se utiliza la palabra clave ***Dim***, seguida del identificador o nombre que se dará a dicha variable. Ver Código fuente 4.B.

```
Sub Main()
    Dim MiValor
End Sub
```

Código fuente 4.B.

Respecto al nombre de la variable, es decir la **denominación**, debe empezar por letra, y no puede ser ninguna de las palabras reservadas del lenguaje, ni contener caracteres como operadores u otros símbolos especiales. Ver Código fuente 4.C.

```
Sub Main()
    Dim MiValor ' nombre correcto
    Dim Total2 ' nombre correcto
    Dim Mis_Datos ' nombre correcto
Dim 7Datos ' nombre incorrecto
Dim Nombre+Grande ' nombre incorrecto
Dim End ' nombre incorrecto
End Sub
```

Código fuente 4.C

Es posible comprobar en este fuente, el incluir comentarios en el código usando la comilla simple (') seguida del comentario correspondiente.

4.4.3.2. AVISOS DEL IDE SOBRE ERRORES EN EL CÓDIGO

Al declarar una variable con un nombre incorrecto, o si se produce otro tipo de error en la escritura del código, el propio IDE se encarga de avisar que existe un problema, subrayando el fragmento de código conflictivo y mostrando una viñeta informativa al situar sobre dicho código el cursor. Ver Figura 4.13

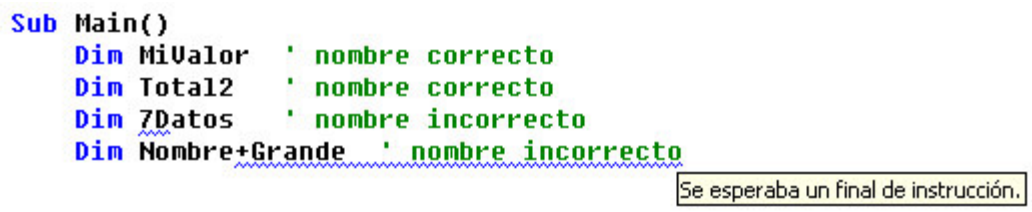


Figura 4.13. Código con errores subrayado por el *IDE*.

Estos avisos constituyen una gran ayuda, ya que permiten al programador observar problemas en la escritura del código, antes incluso de ejecutar el programa.

Existen multitud de avisos de muy diversa naturaleza, teniendo en cuenta que la tónica general consiste en que el código problemático quedará subrayado por el *IDE* hasta que se modifique la línea en cuestión y se escriba correctamente.

4.4.3.3. LUGAR DE LA DECLARACIÓN

Es posible declarar variables en muy diversos lugares del código. El punto en el que se declara una variable será determinante a la hora del ámbito o accesibilidad a esa variable desde otros puntos del programa. Es recomendable declarar todas las variables en la cabecera o comienzo del procedimiento, para dar una mayor claridad al mismo. Después de la declaración, escribir el resto de instrucciones del procedimiento.

4.4.3.4. TIPIFICACIÓN

La tipificación de una variable es la operación por la cual, al declarar una variable, se especifica qué clase de valores o tipo de datos pueden almacenase en dicha variable.

VB.NET utiliza la palabra clave ***As*** seguida del nombre del tipo de datos, para establecer el tipo de una variable. Ver Código fuente 4.D.

```
Sub Main()
    Dim Valor As String ' cadena de caracteres
    Dim Cuenta As Integer ' numérico entero
    Dim FhActual As Date ' fecha
End Sub
```

Código fuente 4.D

La Tabla 4.1 muestra la relación de tipos de datos disponibles en el lenguaje.

TIPO DE DATO EN VB.NET	TIPO CORRESPONDIENTE EN EL .NET FRAMEWORK	TAMAÑO	RANGO DE VALORES
Boolean	System.Boolean	1 byte	True o False (Verdadero o Falso)
Byte	System.Byte	1 byte	0 a 255 (sin signo)
Char	System.Char	2 bytes	0 a 65535 (sin signo)
Date	System.DateTime	8 bytes	1 de enero de 1 a 31 de diciembre de 9999
Decimal	System.Decimal	12 bytes	+/-79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/-0,0000000000000000000000000001
Double (punto flotante con precisión doble)	System.Double	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores +
Integer	System.Int32	4 bytes	-2.147.483.648 a 2.147.483.647
Long (entero largo)	System.Int64	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
Short	System.Int16	2 bytes	-32.768 a 32.767
Single (punto flotante con precisión simple)	System.Single	4 bytes	-3,402823E38 a –1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Object	System.Object	4 bytes	Cualquier tipo
String (cadena de longitud variable)	System.String	10 bytes + (2 * longitud de la cadena)	Desde 0 a unos 2.000 millones de caracteres Unicote
Estructura (tipo de dato definido por el usuario)	Hereda de System.ValueType	Suma de los tamaños de los miembros de la estructura correspondientes a los demás miembros	Cada miembro de la estructura tiene un intervalo de valores determinado por su tipo de datos e independiente de los intervalos de valores

Tabla 4.1. Tipos de datos en *VB.NET*.

Si al declarar una variable no se indica el tipo, por defecto tomará *Object*, que corresponde al tipo de datos genérico en el entorno del *CLR*, y admite cualquier valor.

Si se declara una variable de tipo *Byte* e intentase asignarle el valor 5899 se va a producir un error, ya que no se encuentra en el intervalo de valores permitidos para esa variable. Esto puede llevar al lector a preguntar: "¿por qué no utilizar siempre *Object* y poder usar cualquier valor?, o mejor ¿para qué se necesita asignar tipo a las variables?".

El motivo de tipificar las variables reside en que cuando se realiza una declaración, el *CLR* debe reservar espacio en la memoria para los valores que pueda tomar la variable, como puede ver el lector en la tabla anterior, no requiere el mismo espacio en memoria una variable *Byte* que una Date. Si además, se declaran todas las variables como *Object*, los gastos de recursos del sistema serán mayores que si se establece el tipo adecuado para cada una, ya que como el *CLR* no sabe el valor que puede tomar en cada ocasión la variable, debe realizar un trabajo extra de adecuación, consumiendo una mayor cantidad de recursos.

Una correcta tipificación de las variables redundará en un mejor aprovechamiento de las capacidades del sistema y en un código más veloz en ejecución. Cuantos más programas se diseñen optimizando en este sentido, el sistema operativo ganará en rendimiento beneficiándose el conjunto de aplicaciones que estén en ejecución.

VS.NET dispone de una ayuda al asignar el tipo a una variable, que nos muestra la lista de tipos disponibles para poder seleccionar uno sin tener que escribir nosotros el nombre. Al terminar de escribir la palabra As, aparecerá dicha lista, en la que pulsando las primeras letras del tipo a buscar, se irá situando en los más parecidos. Una vez encontrado, pulsar la tecla *Enter* o *Tab* para tomarlo. Ver Figura 4.14.

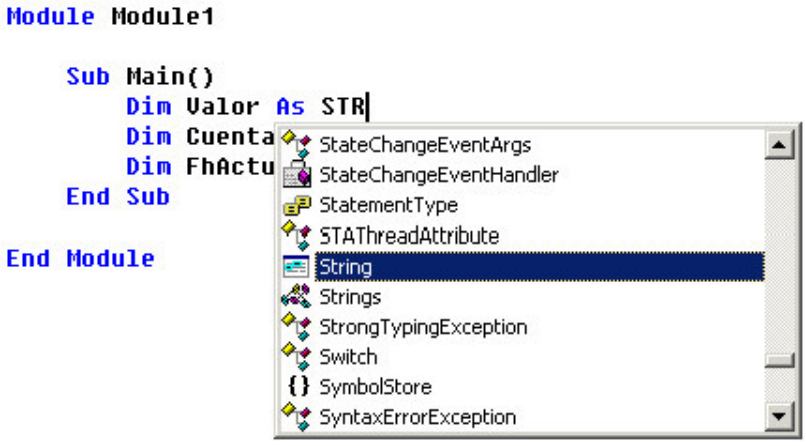


Figura 4.14. Lista de tipos de datos al declarar una variable.

4.4.3.5. DECLARACIÓN MÚLTIPLE EN LÍNEA

En el caso de tener que declarar más de una variable del mismo tipo, puede declararse todas en la misma línea, separando cada una con una coma e indicando al final de la lista el tipo de dato que van a tener, como en el Código fuente 4.E.

```
Dim Nombre, Apellidos, Ciudad As String
```

Código fuente 4.E.

Con esta técnica de declaración, todas las variables de la línea tienen el mismo tipo de dato, ya que no es posible declarar múltiples variables en la misma línea que tengan distintos tipos de dato.

4.4.3.6. ASIGNACIÓN DE VALOR

Para asignar un valor a una variable se utiliza el operador de asignación: el signo igual (=), situando a su izquierda la variable a asignar, y a su derecha el valor. Ver Código fuente 4.F.

```
Dim Cuenta As Integer
Cuenta = 875
```

Código fuente 4.F.

Según el tipo de dato de la variable, puede ser necesario el uso de delimitadores para encerrar el valor a asignar.

- ☑ Tipos numéricos. Las variables de tipos de datos numéricos no necesitan delimitadores, se asigna directamente el número correspondiente. Si se necesita especificar decimales, se utiliza el punto (.) como carácter separador para los decimales
- ☑ Cadenas de caracteres. En este caso es preciso encerrar la cadena entre comillas dobles (").
- ☑ Fechas. Al asignar una fecha a una variable de este tipo, es posible encerrar dicho valor entre el signo de almohadilla (#) o comillas dobles ("). El formato de fecha a utilizar depende del delimitador. Cuando utilice almohadilla la fecha tendrá el formato Mes/Día/Año; mientras que cuando se use comillas dobles el formato será Día/Mes/Año. Las fechas pueden contener además información horario que se especifica en el formato Hora:Minutos:Segundos FranjaHoraria. En el caso de no indicar la franja horaria (AM/PM) y si esta utilizando el signo almohadilla como separador, el entorno insertará automáticamente los caracteres de franja horaria correspondientes.
- ☑ Tipos lógicos. Las variables de este tipo sólo pueden tener el valor *True* (Verdadero) o *False* (Falso).

Además de asignar valores, se puede asignar el contenido de una variable a otra o el resultado de una expresión. El Código fuente 4.G. muestra unos ejemplos de asignación a variables.

```
Sub Main()
    Dim ImporteFac As Integer
    Dim Precio As Double
    Dim Valor As String
```

```
    Dim FhActual As Date
    Dim FhNueva As Date
    Dim FhCompletaUno As Date
    Dim FhCompletaDos As Date
    Dim FhHora As Date
    Dim Correcto As Boolean
    ImporteFac = 875
    Precio = 50.75
    Valor = "mesa"
    FhActual = #5/20/2001# ' mes/día/año
    FhNueva = "25/10/2001" ' dia/mes/año
    FhCompletaUno = #10/18/2001 9:30:00 AM#
    FhCompletaDos = "7/11/2001 14:22:00"
    FhHora = #5:40:00 PM#
    Dim NuevaCadena As String
    NuevaCadena = Valor ' asignar una variable a otra
    Correcto = True
    ' mostrar variables en la consola
    Console.WriteLine("Variable ImporteFac: {0}", ImporteFac)
    Console.WriteLine("Variable Precio: {0}", Precio)
    Console.WriteLine("Variable Valor: {0}", Valor)
    Console.WriteLine("Variable FhActual: {0}", FhActual)
    Console.WriteLine("Variable FhNueva: {0}", FhNueva)
    Console.WriteLine("Variable FhCompletaUno: {0}", FhCompletaUno)
    Console.WriteLine("Variable FhCompletaDos: {0}", FhCompletaDos)
    Console.WriteLine("Variable FhHora: {0}", FhHora)
    Console.WriteLine("Variable NuevaCadena: {0}", NuevaCadena)
    Console.WriteLine("Variable Correcto: {0}", Correcto)
    Console.ReadLine()
End Sub
```

Código fuente 4.G.

Otra cualidad destacable en este apartado de asignación de valores, reside en la posibilidad de declarar una variable y asignar valor en la misma línea de código, como en el Código fuente 4.H.

```
Dim Valor As String = "mesa"
Dim ImporteFac As Integer = 875
```

Código fuente 4.H.

4.4.3.7. VALOR INICIAL

Toda variable declarada toma un valor inicial por defecto, a no ser que se realice una asignación de valor en el mismo momento de la declaración. A continuación se muestran algunos valores de inicio en función del tipo de dato que tenga la variable:

- ☑ Numérico. Cero (0).
- ☑ Cadena de caracteres. Cadena vacía ("").
- ☑ Fecha. 01/01/0001 0:00:00.
- ☑ Lógico. Falso (False).
- ☑ Objeto. Valor nulo (Nothing).

El Código fuente 4.I. muestra un ejemplo de valores iniciales.

```
Sub Main()
    Dim ImporteFac As Integer
    Dim Valor As String
    Dim FhActual As Date
    Dim FhNueva As Date
    Dim ValorLogico As Boolean
    Dim UnObjeto As Object
    ' mostrar variables en la consola
    Console.WriteLine("Variable ImporteFac: {0}", ImporteFac)
    Console.WriteLine("Variable Valor: {0}", Valor)
    Console.WriteLine("Variable FhActual: {0}", FhActual)
    Console.WriteLine("Variable FhNueva: {0}", FhNueva)
    Console.WriteLine("Variable ValorLogico: {0}", ValorLogico)
    Console.WriteLine("Variable UnObjeto: {0}", UnObjeto)
    Console.ReadLine()
End Sub
```

Código fuente 4.I.

Se debe tener en cuenta al ejecutar estas líneas, que en los casos de las variables de tipo cadena y objeto, no se mostrará nada, ya que se considera que están inicializadas pero vacías.

Por otro lado, inversamente, se puede inicializar una variable que ya tiene valor, asignándole la palabra clave Nothing; con ello, la variable pasa a tener el valor por defecto o inicial. Ver el Código fuente 4.J.

```
Sub Main()
    Dim Valor As String
    Dim FhActual As Date
    Dim ValorLogico As Boolean
    ' asignar valores a variables
    Valor = "mesa"
    FhActual = "10/8/2001"
    ValorLogico = True
    ' inicializar variables
    Valor = Nothing
    FhActual = Nothing
    ValorLogico = Nothing
    ' mostrar variables en la consola
    Console.WriteLine("Variable Valor: {0}", Valor)
    Console.WriteLine("Variable FhActual: {0}", FhActual)
    Console.WriteLine("Variable ValorLogico: {0}", ValorLogico)
    Console.ReadLine()
End Sub
```

Código fuente 4.J.

4.4.3.8. DECLARACIÓN Y TIPIFICACIÓN OBLIGATORIA

Declaración obligatoria

Es obligatorio, por defecto, la declaración de todas las variables a utilizar en el código. En el caso de intentar utilizar una variable no declarada, se producirá un error.

La declaración de variables proporciona una mayor claridad al código, ya que de esta forma, se sabe en todo momento si un determinado identificador corresponde a una variable en el procedimiento, de un parámetro, etc.

Mediante la instrucción *Option Explicit*, y sus modificadores *On/Off*, se puede requerir o no la declaración de variables dentro del programa.

- ☑ *Option Explicit On*. Hace obligatoria la declaración de variables. Opción por defecto.
- ☑ *Option Explicit Off*. Hace que no sea obligatoria la declaración de variables.

Es posible aplicar esta instrucción para que tenga efecto a nivel de proyecto y a nivel de fichero de código.

Para establecer *Option Explicit* a nivel de proyecto, debe abrir la ventana Explorador de soluciones, hacer clic en el nombre del proyecto, y a continuación pulsar el botón de propiedades en esa misma ventana. Esto mostrará la ventana de propiedades del proyecto, en cuyo panel izquierdo se hará clic sobre el elemento Generar. Finalmente se abre la lista desplegable del elemento *Option Explicit*, se selecciona un valor (*On, Off*) y se pulsa Aplicar y Aceptar. Ver Figura 4.15.

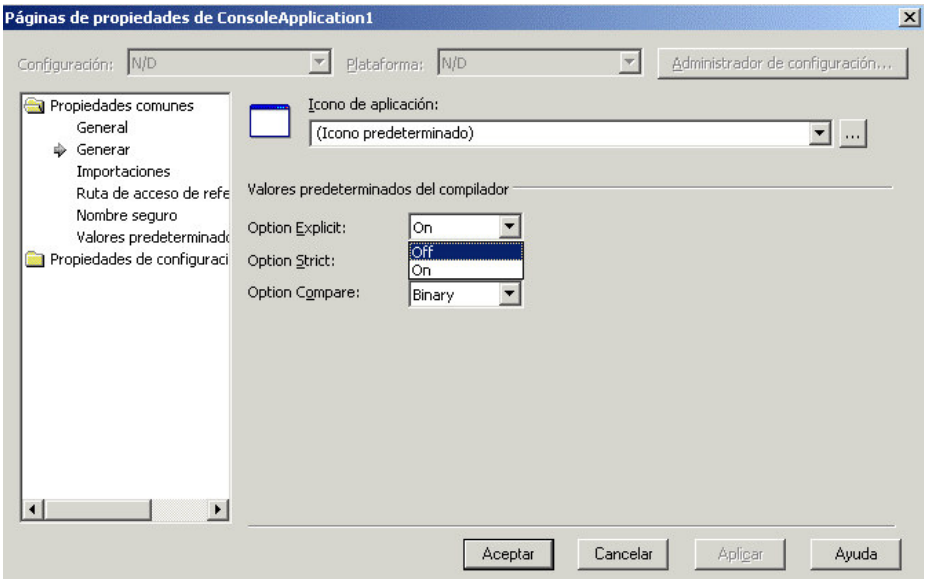


Figura 4.15. Propiedades del proyecto para modificar la declaración obligatoria de variables.

Con la declaración obligatoria desactivada puede escribirse código como el mostrado en el Código fuente 4.K.

```
    Sub Main()
        Valor = "coche"
        MiDato = 984
        Console.WriteLine("Variable Valor: {0}", Valor)
        Console.WriteLine("Variable MiDato: {0}", MiDato)
        Console.ReadLine()
    End Sub
```

Código fuente 4.K.

En el ejemplo anterior, no se han declarado las variables en Main( ). Al estar *Option Explicit Off* esto no produce error, y el *CLR* al detectar un identificador sin declarar, crea una nueva variable internamente.

Precisamente esta facilidad es uno de los graves problemas de no declarar variables. En un procedimiento de prueba con poco código, esto no supone una importante contrariedad. Sin embargo, si en lugar de un pequeño procedimiento se trata de una gran aplicación con muchas líneas de código, procedimientos y cientos de variables. Al encontrarse con una variable de esta forma, no sabrá si esa variable ya es utilizada con anterioridad en el procedimiento, si ha sido pasada como parámetro al mismo, etc. Estas circunstancias provocan que el código se vuelva complejo de interpretar, retrasando la escritura general de la aplicación. Si se vuelve a activar *Option Explicit On*, inmediatamente se sabe que algo va mal, ya que toda variable no declarada, quedará subrayada por el IDE como un error de escritura. Las ventajas son evidentes.

Option Explicit a nivel de fichero.

Para establecer la declaración obligatoria a nivel de fichero, debe situarse al comienzo del fichero de código y escribir la instrucción *Option Explicit* con el modificador correspondiente. El Código fuente 4.L. muestra un ejemplo de cómo desactivar esta característica en el fichero de código actual.

```
' desactivar declaración obligatoria de variables
' ahora es puede, dentro de este fichero de código,
' escribir todas las variables sin declarar
Option Explicit Off
Module Module1
    Sub Main()
        Valor = "coche"
        MiDato = 984
        Console.WriteLine("Variable Valor: {0}", Valor)
        Console.WriteLine("Variable MiDato: {0}", MiDato)
        Console.ReadLine()
    End Sub
End Module
```

Código fuente 4.L

Option Explicit a nivel de fichero, nos permite establecer el modo de declaración de variables sólo para ese fichero en el que lo se utiliza, independientemente del tipo de obligatoriedad en declaración de variables establecido de forma general para el proyecto. Podemos tener establecido *Option Explicit On* para todo el proyecto, mientras que para un fichero determinado puede no obligarse a declarar variables escribiendo al comienzo del mismo *Option Explicit Off*.

El hecho de tener *Option Explicit Off* no quiere decir que no pueda declararse variables, podemos, por supuesto declararlas, lo que sucede es que el compilador no generará un error al encontrar una variable sin declarar.

El otro grave problema al no declarar variables proviene por la incidencia en el rendimiento de la aplicación. Cuando se tiene *Option Explicit Off*, el *CLR* por cada identificador que encuentre sin declarar, crea una nueva variable, y ya que desconoce qué tipo de dato querría utilizar el programador, opta por asignarle el más genérico: *Object*.

Una excesiva e innecesaria proliferación de variables *Object* afectan al rendimiento del programa, ya que el *CLR* debe trabajar doblemente en la gestión de recursos utilizada por dichas variables.

Por todo lo anteriormente comentado, a pesar de la engañosa facilidad y flexibilidad de *Option Explicit Off*, la recomendación es tener configurado siempre *Option Explicit On* a nivel de aplicación, nos ahorrará una gran cantidad de problemas.

Tipificación obligatoria

Cuando es declarada una variable, no es obligatorio por defecto, establecer un tipo de dato para la misma. Igualmente, al asignar por ejemplo, una variable numérica a una de cadena, se realizan automáticamente las oportunas conversiones de tipos, para transformar el número en una cadena de caracteres. Un ejemplo en el Código fuente 4.M.

```
Sub Main()
    ' no es necesario tipificar la variable, tipificación implícita,
    ' la variable Valor se crea con el tipo Object
    Dim Valor
    ' tipificación explícita
    Dim Importe As Integer
    Dim UnaCadena As String
    ' al asignar una fecha a la variable Valor,
    ' sigue siendo de tipo Object, pero detecta que
    ' se trata de una fecha y guarda internamente
    ' esta información como un subtipo Date
    Valor = #8/20/2001#
    Importe = 590
    ' no es necesario hacer una conversión de tipos previa
    ' para asignar un número a una variable de cadena,
    ' ya que se realiza una conversión implícita,
    ' la variable UnaCadena contiene la cadena "590"
    UnaCadena = Importe
    Console.WriteLine("Variable Valor: {0}", Valor)
    Console.WriteLine("Variable Importe: {0}", Importe)
    Console.WriteLine("Variable UnaCadena: {0}", UnaCadena)
    Console.ReadLine()
End Sub
```

Código fuente 4.M.

Si no se asigna el tipo de dato adecuado al declarar una variable, el CLR le asigna el tipo *Object*, lo que afecta negativamente al rendimiento de la aplicación.

La instrucción *Option Strict*, junto a sus modificadores *On/Off*, permite establecer si en el momento de declarar variables, será obligatoria su tipificación. También supervisa la obligatoriedad de realizar una conversión de tipos al efectuar asignaciones entre variables, o de expresiones a variables.

☑ *Option Strict On*. Hace obligatoria la tipificación de variables y la conversión de tipos explícita.

☑ *Option Strict Off*. Hace que no sea obligatoria la tipificación de variables. La conversión de entre tipos distinta en asignaciones y expresiones es realizada automáticamente por el entorno.

Opción por defecto.

Puede configurarse *Option Strict* a nivel de proyecto y de fichero de código, de igual forma que con *Option Explicit*. En el caso de configurar a nivel de proyecto, debe abrirse la ventana de propiedades del proyecto, y en su apartado Generar, establecer el valor correspondiente en la lista desplegable *Option Strict*. Ver Figura 4.16.

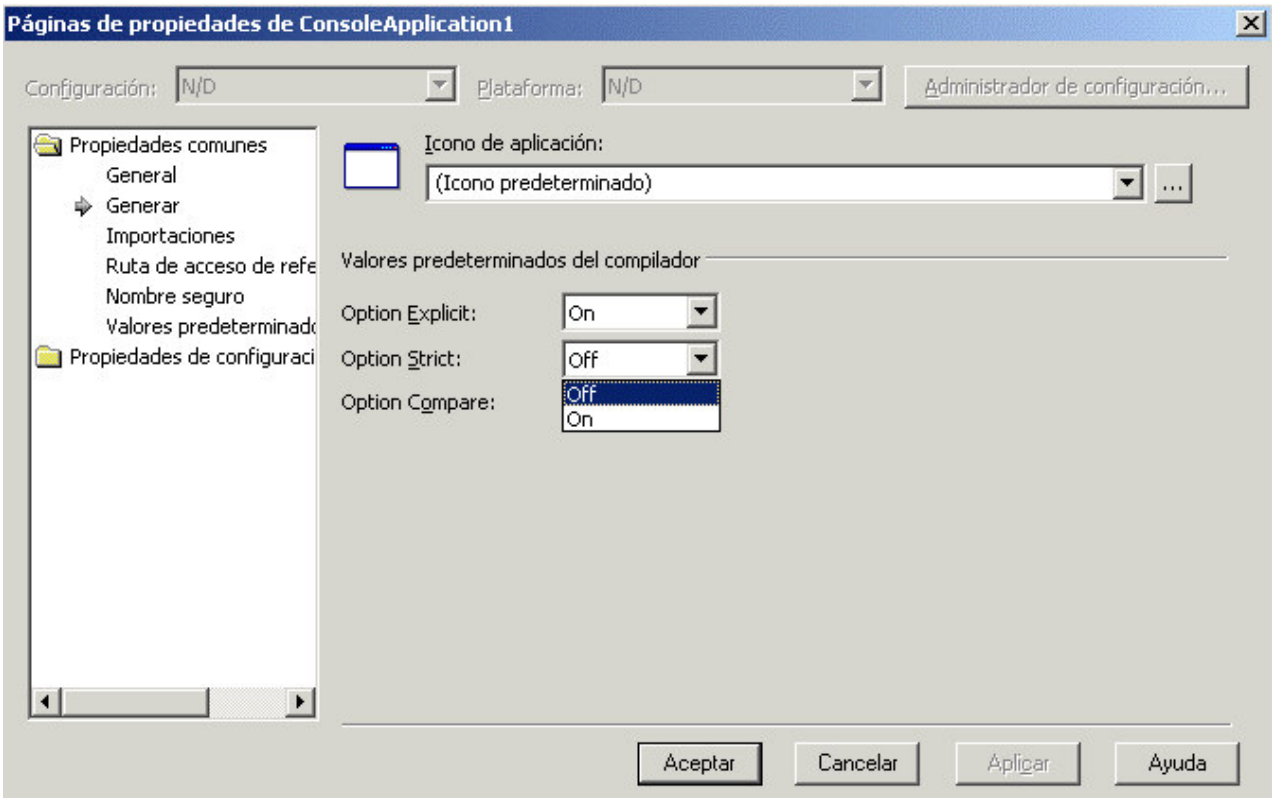


Figura 4.16. Configuración de *Option Strict* a nivel de proyecto.

Si se configura a nivel de fichero de código, se escribe esta instrucción en la cabecera del fichero con el modificador oportuno.

En el ejemplo del Código fuente 4.N, establece *Option Strict On* a nivel de fichero de código, y a partir de ese momento, no podrá asignarse un tipo de dato *Double* a un *Integer*, o un valor

numérico a una variable *String*, por exponer un par de casos de los más comunes. El código erróneo será marcado por el IDE como un error de sintaxis, e igualmente se producirá un error si se intenta ejecutar el programa.

```
Option Strict On
Module Module1
    Sub Main()
        ' ahora es obligatorio establecer
        ' el tipo de dato a todas las variables
        Dim Valor As Integer
        Dim TotalGeneral As Double
        Dim Dato As String
        TotalGeneral = 500
        Valor = TotalGeneral ' error, no se permite la conversión implícita
        Dato = TotalGeneral ' error, no se permite la conversión implícita
    End Sub
End Module
```

Código fuente 4.N.

Si se quiere que no se produzcan errores de conversión en el anterior código fuente, tiene que emplearse las funciones de conversión de tipo que proporciona el lenguaje. En este caso se utiliza *CInt( )*, a la que se pasa un valor numérico como parámetro, y devuelve un tipo numérico *Integer*; y *CStr( )*, que convierte a *String* el valor que se pase como parámetro. El resultado en el Código fuente 4.Ñ.

```
Sub Main()
    ' ahora es obligatorio establecer
    ' el tipo de dato a todas las variables
    Dim Valor As Integer
    Dim TotalGeneral As Double
    Dim Dato As String
    TotalGeneral = 500
    Valor = CInt(TotalGeneral) ' conversión de tipos
    Dato = CStr(TotalGeneral) ' conversión de tipos
End Sub
```

Código fuente 4.Ñ.

Establecer *Option Strict On* requiere un mayor trabajo por parte del programador, ya que ha de ser más cuidadoso y escribir un código más correcto y preciso, lo cual es muy conveniente. Sin embargo, ya que la opción por defecto en este sentido es *Option Strict Off*, los ejemplos realizados a lo largo de este texto se ajustarán en este particular a dicha configuración, con ello se gana en comodidad, ya que evita la obligación de realizar conversiones de tipos en muy diversas situaciones.

4.4.4. ARRAYS, CONCEPTOS BÁSICOS

Un *array* consiste en una lista de valores asociada a un identificador. Al emplear una variable para contener más de un dato, el modo de acceder a los valores se consigue a través de un índice asociado a la variable, que permite saber con qué elemento o posición de la lista se está tratando. Otros nombres para referirse a un *array* son matriz y vector, aunque en este proyecto se emplea el término *array* de forma genérica.

4.4.4.1. DECLARACIÓN

Para declarar un *array* es prácticamente igual que para declarar una variable normal, con la diferencia de que utiliza los paréntesis junto al nombre de la variable, para indicar que se trata de un *array*, y opcionalmente, dentro de los paréntesis, se indicará el número de elementos de que inicialmente va a constar el *array*. También es posible, asignar valores a los elementos en el mismo momento de su declaración.

A la hora de establecer el número de elementos, el primer índice de un *array* es el cero, por lo que al ser creado, el número real de elementos en un *array* será el especificado en la declaración más uno. La Figura 4.17 muestra la representación de un *array* en un modo gráfico.

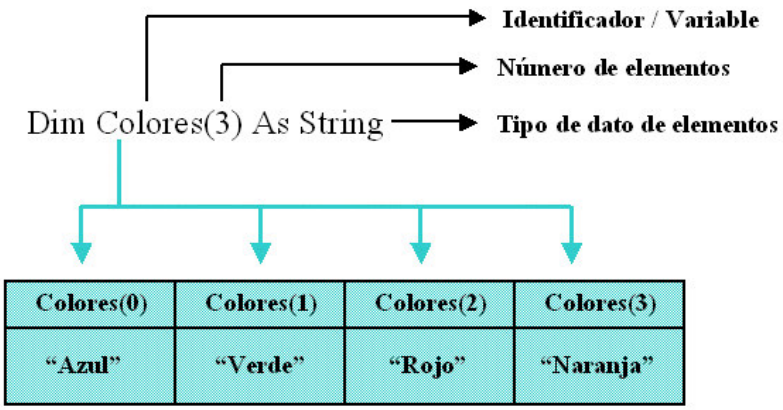


Figura 4.17. Representación gráfica de un *array*.

Ejemplos de creación de *arrays* en el Código fuente 4.O.

Código fuente 4.O.

```
Sub Main()
    ' array sin elementos
    Dim Colores() As String
    ' array con 4 elementos: de 0 a 3
    Dim Nombres(3) As String
    ' array con 3 elementos, cuyos valores son asignados
    ' en el momento de la declaración del array
    Dim Frutas() As String = {"Manzana", "Naranja", "Pera"}
End Sub
```

Al declarar un *array*, todos sus valores son del mismo tipo de dato. Si es necesario que dichos valores sean de tipos diferentes, se debe declarar el *array* como tipo *Object*, ya que al ser esté el tipo de dato genérico en el entorno de *.NET*, permitirá asignar valores de distintos tipos al *array*.

4.4.4.2. ASIGNACIÓN Y OBTENCIÓN DE VALORES

Para asignar y obtener valores de los elementos de un *array*, se actúa igual que para una variable normal, pero empleando además el índice para indicar qué posición se quiere manipular. Ver Código fuente 4.P.

```
Sub Main()
    ' array con 4 elementos: de 0 a 3
    Dim Nombres(3) As String
    ' asignar valores al array
    Nombres(0) = "Ana"
    Nombres(1) = "Pedro"
    Nombres(2) = "Antonio"
    Nombres(3) = "Laura"
    ' obtener valores de un array
    Dim ValorA As String
    Dim ValorB As String
    ValorA = Nombres(1) ' Pedro
    ValorB = Nombres(3) ' Laura
    ' mostrar los valores obtenidos del array
    Console.WriteLine("Variables: ValorA --> {0}, ValorB --> {1}", ValorA,
ValorB)
    Console.ReadLine()
End Sub
```

Código fuente 4.P.

4.4.4.3. MODIFICACIÓN DE TAMAÑO

Para modificar el tamaño o número de elementos de un *array*, se emplea la instrucción *ReDim*, seguida del *array* a modificar y el nuevo tamaño. En el Código fuente 4.Q., se modifica el tamaño de un *array*, añadiendo dos elementos.

```
' array con 4 elementos: de 0 a 3
Dim Nombres(3) As String
' asignar valores al array
Nombres(0) = "Ana"
Nombres(1) = "Pedro"
Nombres(2) = "Antonio"
Nombres(3) = "Laura"
' ampliar el array con 6 elementos: de 0 a 5
ReDim Nombres(5)
```

Código fuente 4.Q.

ReDim no toma el *array* existente y modifica su número de elementos, sino que internamente crea un nuevo *array* con el número de elementos indicado, por lo que se pierden los valores del *array* previo.

Para solucionar este inconveniente, se utiliza junto a *ReDim*, la palabra clave Preserve. Con ello, los valores existentes en el *array* a modificar son conservados. Ver Código fuente 4.R.

```
' ampliar el array con 6 elementos: de 0 a 5
' y los valores de los elementos que hubiera, son conservados
ReDim Preserve Nombres(5)
```

Código fuente 4.R.

4.4.4.4. RECORRER UN ARRAY

Para recorrer todos los elementos de un *array*, la estructura de control *For...Next*, ejecuta un bloque de código, un número determinado de veces, y la función del lenguaje *Ubound*( ), que devuelve el número de elementos del *array* pasado como parámetro. Ver Código fuente 4.S.

```
Sub Main()
    ' crear un array y rellenarlo con valores
    Dim Nombres(3) As String
    Nombres(0) = "Ana"
    Nombres(1) = "Pedro"
    Nombres(2) = "Antonio"
    Nombres(3) = "Laura"
    ' recorrer el array y mostrar el contenido
    ' de cada uno de sus elementos
    Dim Contador As Integer
    For Contador = 0 To UBound(Nombres)
        Console.WriteLine("Posición del array: {0}, valor: {1}", _
        Contador, Nombres(Contador))
    Next
    Console.ReadLine()
End Sub
```

Código fuente 4.S.

4.4.5. CONSTANTES

Al igual que las variables, una constante es un elemento del lenguaje que guarda un valor, pero que en este caso y como su propio nombre indica, dicho valor será permanente a lo largo de la ejecución del programa, no pudiendo ser modificado.

Para declarar una constante, se debe utilizar la palabra clave *Const*, debiendo al mismo tiempo establecer el tipo de dato y asignarle valor. Ver Código fuente 4.T.

```
Sub Main()
    Const Color As String = "Azul"
    Const ValorMoneda As Double = 120.48
End Sub
```

Código fuente 4.T.

La tipificación de una constante se rige, al igual que las variables, por la configuración que se tenga establecida para la instrucción *Option Strict*.

Si se intenta asignar un valor a una constante después de su asignación inicial, el *IDE* subrayará la línea con un aviso de error de escritura, y se producirá igualmente un error si se intenta ejecutar el programa. Ver Figura 4.18

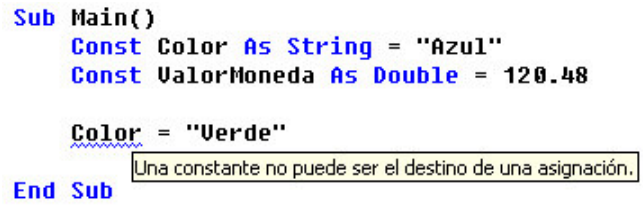


Figura 4.18. No es posible asignar valores a constantes después de su creación.

La ventaja del uso de constantes reside en que es posible tener un valor asociado a una constante, a lo largo de nuestro código para efectuar diversas operaciones. Si por cualquier circunstancia, dicho valor debe cambiarse, sólo se tiene que hacer en el lugar donde se declara la constante.

Supóngase como ejemplo, que se ha escrito un programa en el que se realiza una venta de productos y se confeccionan facturas. En ambas situaciones se debe aplicar un descuento sobre el total resultante. Ver Código fuente 4.U. Código fuente 4.U.

```
Sub Main()
    ' venta de productos
    Dim Importe As Double
    Dim TotalVenta As Double
    Console.WriteLine("Introducir importe de la venta")
    Importe = Console.ReadLine()
    ' aplicar descuento sobre la venta
    TotalVenta = Importe - 100
    Console.WriteLine("El importe de la venta es: {0}", TotalVenta)
    Console.WriteLine()
    ' ............
    ' ............
    ' ............
    ' factura de mercancías
    Dim PrecioArt As Double
    Dim TotalFactura As Double
    Console.WriteLine("Introducir precio del artículo")
    PrecioArt = Console.ReadLine()
    ' aplicar descuento a la factura
    TotalFactura = PrecioArt - 100
    Console.WriteLine("El total de la factura es: {0}", TotalFactura)
    Console.WriteLine()
    ' ............
    ' ............
    ' ............
    Console.ReadLine()
End Sub
```

En el anterior ejemplo, el descuento se realiza utilizando directamente el valor a descontar. Si en un momento dado, se necesita cambiar dicho valor de descuento, se tiene que recorrer todo el código e ir cambiando en aquellos lugares donde se realice esta operación.

Empleando una constante para el descuento, y utilizando dicha constante en todos aquellos puntos del código en donde se necesite aplicar un descuento, cuando deba modificarse el descuento, sólo se necesita hacerlo en la línea en la que es declarada la constante. Ver Código fuente 4.V.

```
Sub Main()
    ' crear constante para calcular descuento
    Const DESCUENTO As Integer = 100
    ' venta de productos
    Dim Importe As Double
    Dim TotalVenta As Double
    Console.WriteLine("Introducir importe de la venta")
    Importe = Console.ReadLine()
    ' aplicar descuento sobre la venta, atención al uso de la constante
    TotalVenta = Importe - DESCUENTO
    Console.WriteLine("El importe de la venta es: {0}", TotalVenta)
    Console.WriteLine()
    ' ............
    ' ............
    ' ............
    ' factura de mercancías
    Dim PrecioArt As Double
    Dim TotalFactura As Double
    Console.WriteLine("Introducir precio del artículo")
    PrecioArt = Console.ReadLine()
    ' aplicar descuento a la factura, atención al uso de la constante
    TotalFactura = PrecioArt - DESCUENTO
    Console.WriteLine("El total de la factura es: {0}", TotalFactura)
    Console.WriteLine()
    ' ............
    ' ............
    ' ............
    Console.ReadLine()
End Sub
```

Código fuente 4.V.

4.4.6. CONCEPTOS MÍNIMOS SOBRE DEPURACIÓN

Un depurador permite adentrarse en el código del programa durante la ejecución del mismo, para observar qué es lo que está ocurriendo: ejecutar línea a línea el programa, observar el valor de las variables, etc., aspectos todos ellos fundamentales para el seguimiento de errores y fallos en la lógica de la aplicación.

VS.NET dispone de un excelente depurador.

Para ejecutar el programa en modo de depuración se pulsa [F8], o selecciona el menú Depurar + Ir a instrucciones. Cualquiera de estas acciones iniciará el programa dentro del contexto del depurador, deteniendo la ejecución en la primera línea de código ejecutable, destacada en color amarillo. La línea marcada en amarillo indica que está a punto de ejecutarse, para ejecutarla y pasar a la siguiente línea se pulsa de nuevo [F8], y así sucesivamente hasta llegar a la última línea del programa, donde se finalizará el mismo, cerrándose el depurador.

Para ver de forma inmediata el valor de una variable simplemente se sitúa el cursor del ratón sobre ella, con lo que se mostrará una viñeta informativa de su valor. Ver Figura 4.19.

Figura 4.19. Ejecución del programa en el depurador.

Es posible también ver con detalle el valor que van adquiriendo las variables a lo largo de la ejecución, abriendo la ventana Locales del depurador, mediante el menú *Depurar* + *Ventanas* + *Locales*, o la pulsación [*CTRL* + *ALT* + *V, L*]. Ver Figura 4.20.

Locales

Nombre	Valor	Tipo
FhActual	#5/20/2001#	Date
FhCompletaDos	#12:00:00 AM#	Date
FhCompletaUno	#12:00:00 AM#	Date
FhHora	#12:00:00 AM#	Date
FhNueva	#12:00:00 AM#	Date
ImporteFac	875	Integer
Precio	50.75	Double
Valor	"mesa"	String

Figura 4.20. Ventana Local del depurador.

En el caso de *arrays*, se debe hacer clic en el signo más (+) que aparece junto al nombre de la variable, para abrir y mostrar los elementos del *array*. Ver Figura 4.21.

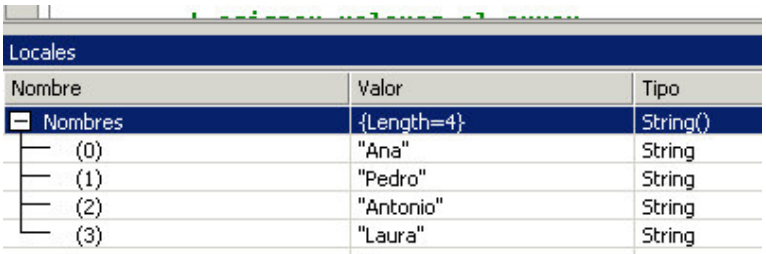


Figura 4.21. Ventana Locales del depurador, mostrando el contenido de un *array*.

Si en cualquier momento se quiere continuar la ejecución normal del programa sin seguir usando el depurador, se pulsa [F5].

4.5. FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Todo *.NET Framework* está basado en clases (u objetos). A diferencia de las versiones anteriores de *Visual Basic*, la versión *.NET* de este lenguaje basa su funcionamiento casi exclusivamente en las clases contenidas en *.NET Framework*. Debido a esta dependencia en las clases del *.NET Framework* y sobre todo a la forma "hereditaria" de usarlas, *Visual Basic .NET* tenía que ofrecer esta característica sin ningún tipo de restricciones.

La organización de una aplicación en Programación Orientada a Objetos, *(OOP)* se realiza mediante estructuras de código.

Una estructura de código contiene un conjunto de procedimientos e información que ejecutan una serie de procesos destinados a resolver un grupo de tareas con un denominador común. Una aplicación orientada a objetos tendrá tantas estructuras de código como aspectos del programa sea necesario resolver.

Un procedimiento que esté situado dentro de una de estructura de este tipo, no podrá llamar ni ser llamado por otro procedimiento situado en una estructura distinta, si no es bajo una serie de reglas. Lo mismo sucederá con los datos que contenga la estructura, permanecerán aislados del exterior, y sólo serán accesibles siguiendo ciertas normas. Una estructura de código, es lo que en *OOP* se identifica como objeto.

Al ser las estructuras de código u objetos, entidades que contienen una información precisa y un comportamiento bien definido a través del conjunto de procedimientos que incluyen, pueden ser clasificados en función de las tareas que desempeñan. Precisamente, uno de los fines perseguidos por la *OOP* es conseguir una mejor catalogación del código, con base a estructuras jerárquicas dependientes, al estilo de un árbol genealógico.

Todos los elementos que forman parte de un objeto componen la clase del objeto. Una clase consiste en el conjunto de especificaciones que permiten crear los objetos.

Las motivaciones que han llevado al desarrollo de la *OOP* son facilitar una mejor organización y clasificación del código, que la proporcionada por la programación procedural tradicional; aproximando al mismo tiempo, el modo de programar a la manera en que nuestra mente trabaja para aplicar soluciones a los problemas planteados.

4.5.1. OBJETOS

Un objeto es una agrupación de código, compuesta de propiedades y métodos, que pueden ser manipulados como una entidad independiente. Las propiedades definen los datos o información del objeto, permitiendo consultar o modificar su estado; mientras que los métodos son las rutinas que definen su comportamiento.

Un objeto es una pieza que se ocupa de desempeñar un trabajo concreto dentro de una estructura organizativa de nivel superior, formada por múltiples objetos, cada uno de los cuales ejerce la tarea particular para la que ha sido diseñado.

4.5.2. CLASES

Una clase no es otra cosa que el conjunto de especificaciones o normas que definen cómo va a ser creado un objeto de un tipo determinado; algo parecido a un manual de instrucciones conteniendo las indicaciones para crear el objeto.

Los términos objeto y clase son utilizados en *OOP* con gran profusión y en contextos muy similares, por lo que para intentar aclarar en lo posible ambos conceptos, se dice que una clase constituye la representación abstracta de algo, mientras que un objeto constituye la representación concreta de lo que una clase define.

La clase determina el conjunto de puntos clave que ha de cumplir un objeto para ser considerado perteneciente a dicha clase o categoría, ya que no es obligatorio que dos objetos creados a partir de la misma clase sean exactamente iguales, basta con que cumplan las especificaciones clave de la clase.

Mediante un ejemplo preciso: un molde para crear figuras de cerámica y las figuras obtenidas a partir del molde. En este caso, el molde representaría la clase Figura, y cada una de las figuras creadas a partir del molde, sería un objeto Figura. Cada objeto Figura tendrá una serie de propiedades comunes: tamaño y peso iguales; y otras propiedades particulares: un color distinto para cada figura.

Aunque objetos distintos de una misma clase pueden tener ciertas propiedades diferentes, deben tener el mismo comportamiento o los mismos métodos. Para explicar mejor esta circunstancia, tomemos el ejemplo de la clase Coche; podemos crear dos coches con diferentes características (color, tamaño, potencia, etc.), pero cuando aplicamos sobre ellos los métodos Arrancar, Acelerar o Frenar, ambos se comportan o responden de la misma manera.

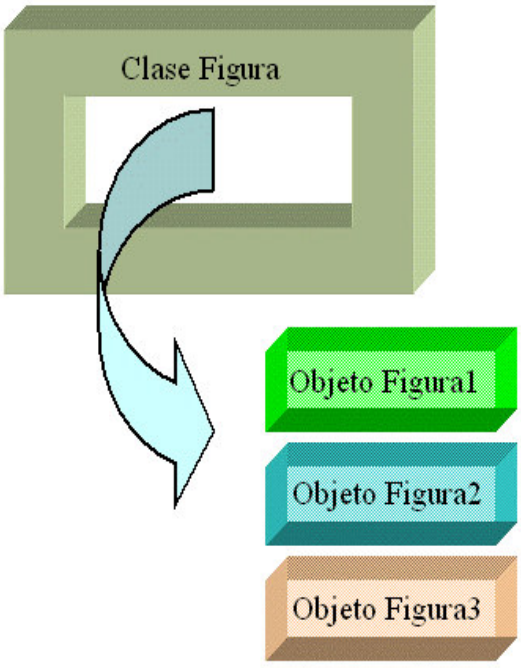


Figura 4.22. Instanciación de objetos a partir de una clase.

4.5.2.1. INSTANCIAS DE UNA CLASE

El proceso por el cual se obtiene un objeto a partir de las especificaciones de una clase se conoce como instanciación de objetos. En la Figura 4.22 volvemos al ejemplo del molde y las figuras; en dicha imagen vemos un molde para fabricar figuras rectangulares, donde la clase Figura estaría representada por el molde, y cada uno de los objetos Figura (iguales en forma pero con la propiedad Color distinta), representaría una instancia de la clase.

4.5.3. CARACTERÍSTICAS BÁSICAS DE UN SISTEMA ORIENTADO A OBJETO

Los tres pilares de la Programación Orientada a Objetos en un lenguaje considerado orientado a objeto, son: 1. Herencia 2. Encapsulación 3. Polimorfismo Nota: Algunos autores añaden un cuarto requisito: la Abstracción, pero este último está estrechamente ligado con la encapsulación.

4.5.3.1. ABSTRACCIÓN

La abstracción es aquella característica que permite identificar un objeto a través de sus aspectos conceptuales.

Las propiedades de los objetos de una misma clase, pueden hacerlos tan distintos que sea difícil reconocer que pertenecen a una clase idéntica. No obstante, es posible reconocer a qué clase pertenecen, identificando además, si se trata de la misma clase para ambos. Ello es posible gracias a la abstracción.

Del ejemplo dos objetos coche, uno deportivo y otro familiar; su aspecto exterior es muy diferente, sin embargo, cuando pensamos en cualquiera de ellos, sabemos que ambos pertenecen a la clase Coche, porque se realiza una abstracción o identificación mental de los elementos comunes que ambos tienen (ruedas, volante, motor, puertas, etc.).

Del mismo modo que al identificar objetos reales, la abstracción ayuda a la hora de desarrollar una aplicación, permitiendo identificar los objetos que van a formar parte del programa, sin necesidad de disponer aún de su implementación; basta con reconocer los aspectos conceptuales que cada objeto debe resolver.

Por ejemplo, cuando se aborda el desarrollo de un programa de gestión orientado a objetos, es realizada una abstracción de los objetos que son necesarios para resolver los procesos del programa: un objeto Empleado, para gestionar al personal de la empresa; un objeto Factura, para gestionar las ventas realizadas de productos; un objeto Usuario, para verificar las personas que utilizan la aplicación, etc.

4.5.3.2. ENCAPSULACIÓN

La encapsulación establece la separación entre el interfaz del objeto y su implementación, aportándonos dos ventajas fundamentales.

Por una parte proporciona seguridad al código de la clase, evitando accesos y modificaciones no deseadas; una clase bien encapsulada no debe permitir la modificación directa de una variable, ni ejecutar métodos que sean de uso interno para la clase.

Por otro lado la encapsulación simplifica la utilización de los objetos, ya que un programador que use un objeto, si éste está bien diseñado y su código correctamente escrito, no necesitará conocer los detalles de su implementación, se limitará a utilizarlo.

Tomando un ejemplo real, cuando se utiliza un objeto Coche, al presionar el acelerador, no necesitamos conocer la mecánica interna que hace moverse al coche, sabemos que el método Acelerar del coche es lo que tenemos que utilizar para desplazarse, y simplemente lo usamos.

Pasando a un ejemplo en programación, si se esta creando un programa de gestión y nos proporcionan un objeto Cliente que tiene el método Alta, y sirve para añadir nuevos clientes a la base de datos, no es necesario conocer el código que contiene dicho método, simplemente se ejecuta y se da de alta a los clientes en la aplicación.

4.5.3.3. POLIMORFISMO

El polimorfismo determina que el mismo nombre de método, realizará diferentes acciones según el objeto sobre el que sea aplicado. Al igual que sucedía en la encapsulación, el programador que haga uso del objeto, no necesita conocer los detalles de implementación de los métodos, se limita a utilizarlos.

Pasando a un ejemplo real, tomamos dos objetos: Pelota y VasoCristal; si se ejecuta sobre ambos el método Tirar, el resultado en ambos casos será muy diferente; mientras que el objeto Pelota rebotará al llegar al suelo, el objeto VasoCristal se romperá.

En un ejemplo aplicado a la programación, supóngase que se disponen de los objetos Ventana y Fichero; si se ejecuta sobre ambos el método Abrir, el resultado en Ventana será la visualización de una ventana en el monitor del usuario; mientras que en el objeto Fichero, se tomará un fichero en el equipo del usuario y se dejará listo para realizar sobre él operaciones de lectura o escritura.

4.5.3.4. HERENCIA

Se trata de la característica más importante de la *OOP*, y establece que partiendo de una clase a la que denominamos clase base, padre o superclase, creamos una nueva clase denominada clase derivada, hija, o subclase. En esta clase derivada se dispone de todo el código de la clase base, más el nuevo código propio de la clase hija, que se escriba para extender sus funcionalidades.

A su vez puede tomarse una clase derivada, creando una nueva subclase a partir de ella, y así sucesivamente, componiendo lo que se denomina una jerarquía de clases.

Existen dos tipos de herencia: simple y múltiple. La herencia simple es aquella en la que se crea una clase derivada a partir de una sola clase base, mientras que la herencia múltiple permite crear una clase derivada a partir de varias clases base. El entorno de *.NET Framework* sólo permite utilizar herencia simple.

Como ejemplo real de herencia, es posible usar la clase Coche como clase base; en ella se reconocen una serie de propiedades como Motor, Ruedas, Volante, etc., y unos métodos como Arrancar, Acelerar, Frenar, etc. Como clase derivada se crean CocheDeportivo, en la cuál,

además de todas las características mencionadas para la clase Coche, se encuentran propiedades y comportamiento específicos como ABS, Turbo, etc.

Un ejemplo basado en programación consiste en disponer de la clase Empleado. Esta clase se ocupa, de las operaciones de alta de empleados, pago de nóminas, etc.; pero en un momento dado, surge la necesidad de realizar pagos a empleados que no trabajan en la central de la empresa, ya que se trata de comerciales que pasan la mayor parte del tiempo desplazándose. Para realizar dichos pagos se usa Internet, necesitando el número de tarjeta de crédito y la dirección *email* del empleado. Se resuelve esta situación creando la clase derivada CiberEmpleado, que hereda de la clase Empleado, en la que sólo se tienen que añadir las nuevas propiedades y métodos para las transacciones electrónicas, puesto que las operaciones tradicionales ya las tendríamos disponibles por el mero hecho de haber heredado de Empleado.

4.5.3.5. JERARQUÍAS DE CLASES

Uno de los fines de la *OOP* consiste en la clasificación del código; para ello se emplean jerarquías o árboles de clases, en los que con base en niveles, se muestra un conjunto de clases conectadas por una relación de herencia. El esquema de la Figura 4.23, muestra un ejemplo de la jerarquía de clases de medios de transporte.

En esta representación de ejemplo, como nivel superior de la jerarquía o clase base se encuentra Medios de transporte, de la que se derivarían las clases Barco, Tren, Automóvil, y a su vez, de estas últimas, partirían nuevas clases hijas.

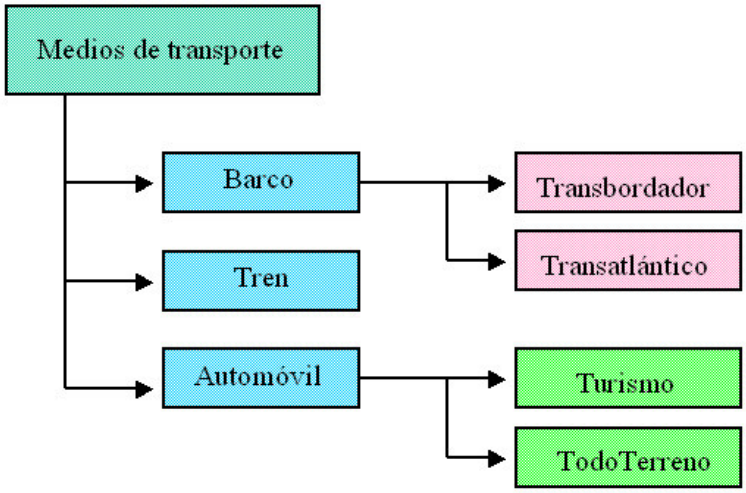


Figura 4.23. Jerarquía de clases de medios de transporte.

4.5.3.6. RELACIONES ENTRE OBJETOS

Los objetos existentes en una aplicación se comunican entre sí mediante una serie de relaciones que describimos a continuación.

Herencia: Cuando a partir de una clase existente, se crea una nueva clase derivada, esta nueva clase dispone de todas las propiedades y métodos de la clase base, más el código propio que implemente.

Para reconocer si existe esta relación entre dos objetos, debe realizarse un análisis sintáctico sobre la misma usando la partícula "es un".

Por ejemplo los objetos Empleado, *CiberEmpleado* y Factura, podemos decir que sí hay una relación de herencia entre Empleado y *CiberEmpleado*, ya que al analizar la frase "Un objeto *CiberEmpleado* es un Empleado", el resultado es verdadero.

No ocurre lo mismo entre los objetos *CiberEmpleado* y Factura, ya que el análisis de la frase "Un objeto *CiberEmpleado* es una Factura", devuelve falso.

Pertenencia: Los objetos pueden estar formados a su vez por otros objetos. Un objeto Factura puede estar compuesto por objetos *CabeceraFactura*, *LineaFactura*, etc. Se dice en este caso que hay una relación de pertenencia, puesto que existe un conjunto de objetos que pertenecen a otro objeto o se unen para formar otro objeto. A este tipo de relación se le denomina también Contenedora.

Para reconocer si existe esta relación entre dos objetos, debe realizarse un análisis sintáctico sobre la misma usando la partícula "tiene un". Así, por ejemplo, la frase "Un objeto Factura tiene un objeto *LineaFactura*" devolvería verdadero.

Utilización: Hay situaciones en que un objeto utiliza a otro para realizar una determinada tarea, sin que ello suponga la existencia de una relación de pertenencia entre dichos objetos.

Por ejemplo, un objeto Ventana puede utilizar un objeto Empleado para mostrar al usuario las propiedades del empleado, sin necesidad de que el objeto Empleado sea propiedad del objeto Ventana.

Nótese la importante diferencia entre esta relación y la anterior, ya que aquí, el objeto Ventana a través de código, creará, o le será pasado como parámetro, un objeto Empleado, para poder mostrarlo en el área de la ventana.

Para reconocer si existe esta relación entre dos objetos, debemos realizar un análisis sintáctico sobre la misma empleando la partícula "usa un". Así, por ejemplo, la frase "Un objeto Ventana usa un objeto Empleado" devolvería verdadero.

Reutilización: Un objeto bien diseñado, puede ser reutilizado en otra aplicación de modo directo o creando una clase derivada a partir de él. Este es uno de los objetivos perseguidos por la OOP, aprovechar en lo posible el código ya escrito, ahorrando un considerable tiempo en el desarrollo de programas.

4.5.4. ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Antes de comenzar la escritura del programa, se hace necesario realizar un análisis de los problemas a resolver, que nos permita identificar qué procesos debemos codificar.

Si pretendemos además, abordar la programación utilizando un enfoque orientado a objetos, debemos emplear técnicas adecuadas a este tipo de programación.

Para aunar todas las tendencias de análisis orientadas a objetos existentes, ha aparecido el Lenguaje Unificado de Modelado o UML (*Unified Modeling Language*), cuyo objetivo es proporcionar un verdadero sistema de análisis y diseño aplicado a objetos.

La descripción de UML es algo que se encuentra fuera del alcance de este texto, por lo que recomendamos al lector consultar la documentación existente al respecto, de manera que pueda familiarizarse con este aspecto de la creación de un programa.

Cuando se realiza un análisis basado en procedimientos, de los problemas planteados, se identifican los verbos como elementos de los procesos a trasladar a procedimientos y funciones. Sin embargo, cuando se trata de un análisis basado en objetos, se identifican en este caso los nombres existentes en los procesos, como elementos a trasladar a objetos.

Tomemos el siguiente planteamiento: "Crear una aplicación en la que podamos realizar sobre una base de datos, las siguientes operaciones: añadir, borrar y modificar clientes. Por otro lado, será necesario crear facturas, grabando sus datos generales y calcular su importe total".

Analizando la exposición del anterior problema, si necesitáramos resolverlo mediante una aplicación con enfoque procedural, separaríamos los verbos para crear los siguientes procedimientos:

AñadirCliente( ), BorrarCliente( ), ModificarCliente( ), GrabarFac( ), CalcularTotalFac( ).

Si por el contrario efectuamos sobre la misma exposición, un análisis orientado a objetos, extraeríamos los siguientes nombres como los objetos a crear: Cliente, Factura.

Para el objeto Cliente, definiríamos entre otras, las propiedades Nombre, Apellidos, Dirección, DNI, etc.; creando para su comportamiento, los métodos *Añadir( ), Borrar( ), Modificar( ),* etc.

Para el objeto Factura, definiríamos entre otras, las propiedades Número, Fecha, Importe, etc.; creando para su comportamiento, los métodos *Grabar( ), CalcularTotal ( )*, etc.

Una vez obtenido el correspondiente análisis, se pasará a la siguiente fase del desarrollo, la escritura de las diferentes clases que van a componer nuestro programa, y que veremos a continuación.

4.5.5. CREAR O DEFINIR UNA CLASE

Al igual que existen instrucciones para declarar o definir una variable o cualquier otro elemento de un programa de Visual Basic, existen instrucciones que permiten crear o definir una clase. Para crear una clase debe usarse la instrucción *Class* seguida del nombre que tendrá dicha clase, por ejemplo:

```
Class Cliente
```

A continuación se escribe el código necesario para implementar las propiedades y métodos de esa clase, y para que Visual Basic sepa que ya hemos terminado de definir la clase, usaremos una instrucción de cierre:

```
End Class
```

Por tanto, todo lo que esté entre *Class* <nombre> y *End Class* será la definición de dicha clase.

4.5.6. DEFINIR LOS MIEMBROS DE UNA CLASE

Para definir los miembros de una clase, se escribe dentro del "bloque" de definición de la clase, las declaraciones y procedimientos se crean convenientes. Por ejemplo:

```
Class Cliente
    Public Nombre As String
    Sub Mostrar()
        Console.WriteLine("El nombre del cliente: {0}", Nombre)
    End Sub
End Class
```

Código fuente 4.W.

En este caso, la línea *Public Nombre As String*, estaría definiendo una propiedad o "campo" público de la clase Cliente.

El procedimiento Mostrar sería un método de dicha clase, en esta caso, permitiría mostrar la información contenida en la clase Cliente.

4.5.7. CREAR UN OBJETO A PARTIR DE UNA CLASE

Las clases definen las características y la forma de acceder a los datos que contendrá, pero sólo eso: los define.

Para asignar información a una clase y usar los métodos de la misma, se crea un objeto basado en esa clase, o lo que es lo mismo: se tiene que crear una nueva instancia en la memoria de dicha clase.

Para ello, se define una variable capaz de contener un objeto del tipo de la clase, como con variable:

```
Dim cli As Cliente
```

Para poder crear un objeto basado en una clase, es necesario algo más de código que permita "crear" ese objeto en la memoria, ya que con el código usado en la línea anterior, simplemente define una variable que es capaz de contener un objeto de ese tipo, pero aún no existe ningún objeto en la memoria, para ello se utiliza el siguiente código:

```
cli = New Cliente()
```

Con esta línea decimos al Visual Basic: crea un nuevo objeto en la memoria del tipo Cliente. Estos dos pasos se pueden simplificar de la siguiente forma:

```
Dim cli As New Cliente()
```

A partir de este momento existirá en la memoria un objeto del tipo Cliente.

4.6. CONCLUSIÓN

Este capítulo ayuda a mostrar temas y conceptos básicos del sistema Visual Basic .NET, lenguaje que se utiliza para ejemplificar a lo largo de los siguientes capítulos. No es un manual, que permita aprender de forma experta el uso y aplicación de este lenguaje, puesto que carece de términos avanzados que puedan también ser utilizado por el lenguaje, en realidad solo se habla de conceptos que son usados en temas de seguridad, esto es: arreglos, cadenas, variables, etc. así como conceptos de características que permite al lenguaje destacar respecto de otros lenguajes por ejemplo herencia, programación orientado a objetos, etc. y que además permitan aplicar dichas ventajas para complementar y diseñar la seguridad en aplicaciones o sistemas.

Las metas de la programación orientada al objeto es mejorar la productividad de los programadores haciendo más fácil de rehusar, extender los programas y manejar sus complejidades. De esta forma con este tipo de programación, se reduce el costo de desarrollo y mantenimiento de los programas. En los lenguajes orientados al objeto los datos son considerados como objetos que a su vez pertenecen a alguna clase. A las operaciones que se definen sobre los objetos son llamados métodos.

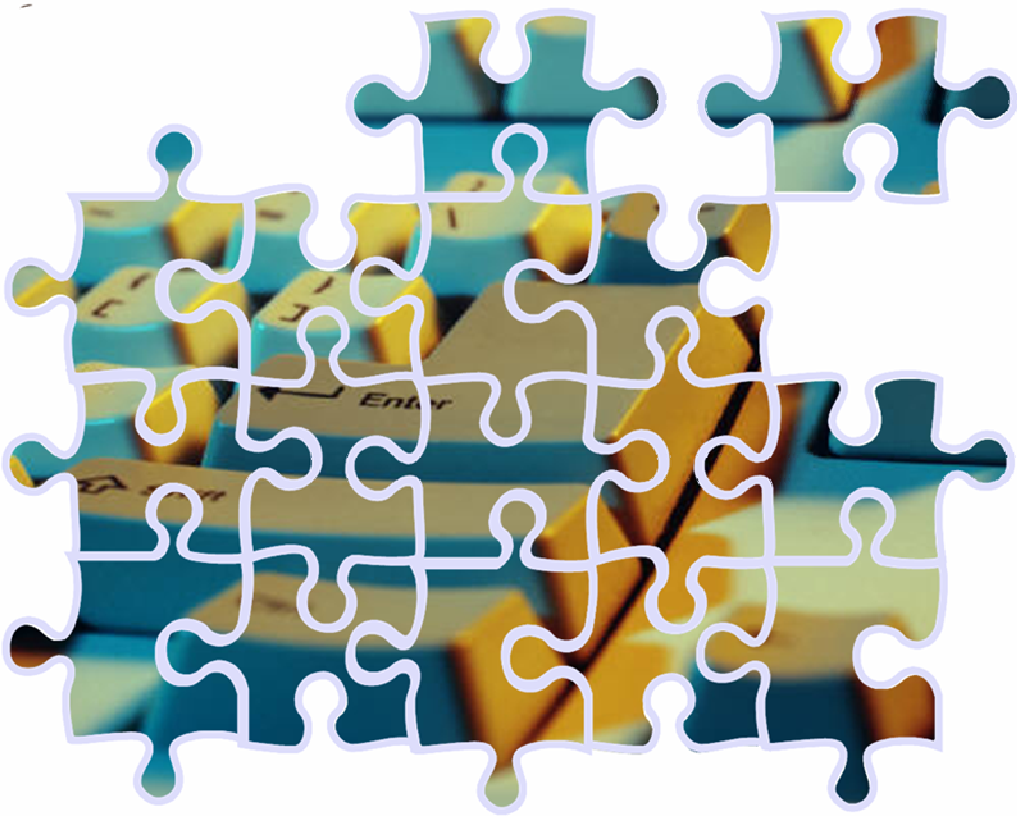
Este capítulo en particular se complementa con las prácticas que se diseñaron, dado que todas explican paso a paso el desarrollo y escritura de las aplicaciones, con la finalidad de comprender el tema que trata la practica, y además ensayar el uso y desarrollo en Visual Basic .NET.

En particular la práctica de Herencia está pensada para comprender este concepto que aun cuando no es complejo de comprender, para un desarrollador, permite reafirmar y aplicar este concepto. La práctica de Ataque DoS también se basa mucho en este capítulo, pues maneja términos como propiedad, form, text, etc. Aunado a los conocimientos que se presentan en este capítulo, en la práctica de ataque en particular se muestran nuevos conceptos, que la práctica ayudara a comprender y aplicar.

De esta forma el capítulo es la base para comprender conceptos y términos que se utilizan en temas siguientes y que se aplican en las prácticas. Todo lo anterior mencionado permite el desarrollo de aplicacionesmas potentes, versatiles,etc. y por ser más ambiciosas exigen un mayor grado de seguridad.

Capítulo 5

Problemas de seguridad para los programadores de Visual Basic .NET



5. PROBLEMAS DE SEGURIDAD PARA PROGRAMADORES DE VISUAL BASIC .NET

En el presente capítulo se muestran los principales errores y problemas de seguridad en torno al desarrollo de aplicaciones en *Visual Basic .NET*. Se analiza la importancia de los factores relativos a la seguridad que hay que tener en cuenta a la hora de diseñar una aplicación, se comentan posibles soluciones a los problemas, así como la necesidad de contemplar las herramientas y estrategias que proporcionen los modelos de seguridad que permiten a los usuarios y programadores obtener el contexto de seguridad adecuado en la aplicación.

Los datos de las aplicaciones o sistemas informáticos están en constante peligro por varias causas: errores del programador, errores de los usuarios, ataques intencionados o fortuitos, interrupción de servicios, etc.

Encuestas de Seguridad y Crimen Computacional de 2002, realizadas por el CERT, evaluaron aplicaciones informáticas. En el 90% de las aplicaciones se detectaron brechas de seguridad y en el 85% se detectaron virus dentro de las aplicaciones. Además de los resultados arrojados por dichas encuestas se mostró que el 95% de todas las brechas de seguridad encontradas en las aplicaciones, hubieran sido prevenidas con una correcta configuración.

Los sistemas informáticos pueden necesitar y requerir protección en algunos de los siguientes aspectos de la información:

Confidencialidad: La aplicación contiene información que requiere protección contra la divulgación no autorizada. Por ejemplo, datos que se van a difundir en un momento determinado (como, información parcial de informes), información personal e información comercial patentada.

Integridad: La aplicación contiene información que debe protegerse de modificaciones no autorizadas, imprevistas o accidentales. Por ejemplo, información de censos, indicadores económicos o sistemas de transacciones financieras.

Disponibilidad: La aplicación contiene información o proporciona servicios que deben estar disponibles puntualmente para satisfacer requisitos o evitar pérdidas importantes. Por ejemplo, sistemas esenciales de seguridad, protección de la vida y predicción meteorológicas.

Autenticación, autorización y auditoría: Prevenir la posibilidad de que un usuario desconocido pueda ganar acceso a los sistemas y datos corporativos es la función que cumplen los métodos de autenticación y autorización, complementados con las funciones de seguimiento (auditoria) que permiten detectar y detener este tipo de ataques.

Las aplicaciones han de diseñarse y desarrollarse cumpliendo con ciertos estándares y requerimientos de seguridad antes mencionados. Entre otros mecanismos, que ayudan a hacer más seguras las aplicaciones, están los certificados de código, o las medidas de protección de las transmisiones de datos entre aplicaciones, tanto en los modelos tradicionales cliente-servidor como en los multicapa y servicios Web.

Es conocido que muchos de los problemas de programación pueden estar relacionados con la seguridad. Algunos suceden por pereza, otros por desconocimiento de un modo mejor de hacer las cosas.

Algunos de estos problemas son defectos en las prácticas de buen desarrollo de software. Estos defectos pueden ser provocados por muchas razones, incluyendo las apretadas fechas de entrega a las que se enfrentan muchos programadores. Otros problemas que es posible mostrar ocurren por no aplicar las técnicas y estrategias que ayudan a que la aplicación sea más robusta y libre de errores. Estas estrategias deben considerarse y ponerse en práctica sencillamente porque hacen que los programas y aplicaciones sean más estables y robustos, además de darle más seguridad al código.

Las situaciones que a continuación se muestran explican un posible problema que amenaza a los programadores. A menudo existen muchas soluciones posibles a un problema, en este capítulo serán presentadas algunas de las soluciones para aplicarse en el capítulo siguiente y desarrollar estrategias que solucionen problemas de seguridad al desarrollar aplicaciones, teniendo en cuenta que aun cuando se trate de problemas distintos, las estrategias tendrán un fin común: robustecer la seguridad.

5.1 EXPLICACIÓN DE LOS PROBLEMAS DE SEGURIDAD PARA LA PROGRAMACIÓN (ANÁLISIS Y DETECCIÓN)

¿Dónde surgen los problemas de seguridad?, este es un buen cuestionamiento que permite establecer el punto de partida para este análisis.

Existen algunos problemas de seguridad que deberá solucionar para poder crear aplicaciones sencillas y prácticas para el usuario.Las situaciones más frecuentes en las que se presentan este tipo de problemas son las siguientes:

- ☑ El usuario que ejecuta la aplicación le niega privilegios a la misma porque se ejecuta desde una ubicación que tiene prohibido el acceso a determinados recursos del sistema, según especificaciones del usuario. Por ejemplo, el usuario puede configurar *Common Language Runtime* para que niegue privilegios de archivo a cualquier aplicación que se encuentre almacenada en una unidad de red. Debe tener esto presente mientras escriba el código, para que así el código responda correctamente a las denegaciones.
- ☑ Se debe evitar que los usuarios que tienen acceso a las aplicaciones *Web* de los servidores *Web* ejecuten en éstos código malintencionado o datos dañados.
- ☑ Según el modo en que configure *Visual Studio*, el servidor correrá mayor o menor riesgo de sufrir los ataques de código malintencionado.

5.1.1 FALTA DE CONCIENCIA EN SEGURIDAD INFORMÁTICA

Algunas veces un programador o un equipo de programadores sencillamente no piensan en las implicaciones de seguridad del código. Esto origina a menudo muchos contratiempos. Los programadores deciden permitir al sistema operativo externo manipular la seguridad o esperar

que el usuario final mantenga la aplicación detrás de un cortafuego. Existe, generalmente, una falta de reflexión dedicada a cómo una aplicación puede influir con otras.

No saber o no preocuparse por la seguridad puede hacer que la aplicación pueda subvertir otros problemas, especialmente si se exagera con la confía en la aplicación diseñada por uno mismo. El ser consciente de la seguridad del desarrollo de aplicaciones es una posible solución. Evidentemente no es posible asegurar que está rigurosamente codificada y que no causará ningún problema relacionado con la seguridad, sin embargo uno puede convertirse en un programador consciente que la seguridad exige tiempo, esfuerzo y una re-evaluación constante. En este aspecto también se debe ser conciente que nunca se sabe si el código programado está realmente seguro hasta que alguien lo rompe.

5.1.2 MAL DISEÑO DE CÓDIGO

El diseñar "mal" el código es también un problema con el que se encuentran los programadores a largo plazo. No está sólo relacionado con la seguridad, sino que afecta a todos los aspectos del mundo de la programación. El código "bueno", es decir, el código robusto, seguro; lo es por su buen diseño. No se obtiene por casualidad, y no hay cantidad suficiente de parches ni cintas virtuales que arregle una decisión mal tomada de diseño de código.

Es recomendable que se **esquematice** todo antes de escribir una sola línea de código. Debe trazarse un **mapa** de todas las interacciones entre componentes y dibujar todas las estructuras necesarias. Esto nos ayudará a tener una idea clara de lo que está sucediendo realmente en el programa. Respecto los esquemas y mapas estos se explican en el siguiente capitulo como parte de las estrategias de seguridad.

Existe una variedad de herramientas y métodos que pueden ayudarnos a diseñar software. Los sistemas como UML pueden ayudar a tomar decisiones coherentes y sanas.

5.1.3 MALA COMPROBACIÓN DEL CÓDIGO

Comprobar mal el código es opuesto al diseño seguro. Después de que el programa se diseña e implementa, es necesario tomar tiempo para comprobarlo tanto como sea posible. Se comprobará en tantas arquitecturas diferentes como sea posible, ya que una vez que se entregue, no es probable que se implemente en la forma para la que se ha diseñado. El mundo real es un sitio difícil para el software, del que se puede abusar, y que se puede utilizar de forma incorrecta de infinitas formas que no se ha previsto durante el diseño. Debe permitirse que alguien que no sea el autor o el implementador compruebe el sistema, ya que a menudo algunos patrones de pensamiento que se utilizan en el diseño y la implementación pueden impedir apreciar "cosas" del mismo modo que lo haría un atacante.

Es importante comprobar nuestro código de la misma forma que lo haría un atacante. Siendo creativos es posible exponer algunos ejemplos de ataques que los otros pueden utilizar contra el propio código:

• Romper los depuradores y desensambladores e intentar que trabajen fuera de sus parámetros normales.

• Comprobar sus acciones de línea de comandos e introducir cadenas largas.

• Si se implementa un analizador sintáctico para archivos configurados, o archivos de cualquier tipo, nos aseguraremos de que los archivos mal formateados no causen problemas.

• Si el programa escucha la información de la red, le arrojaremos desperdicios utilizando un programa como Netcat[10].

Es comprensible que pocas organizaciones puedan dedicar tiempo y esfuerzo a obtener comprobadores de software expertos, además de programadores, y a menudo una persona, o grupo de personas, tienen que cumplir los dos papeles. En tales situaciones, si creamos una pieza de software y la estamos comprobando, intentaremos ser objetivos y ver si es posible romperla. De alguna forma, éste es un desafío interesante para los programadores.

Incluso en entornos de desarrollo de software comercial, que ser capaz de romper software en un entorno de prueba no es nada malo, ya que estamos detectando posibles violaciones de seguridad antes de que el público tenga acceso al código.

5.1.4 NO DESCARTAR POSIBILIDADES

Una parte importante de los fallos relacionados con la seguridad en el software procede del hecho de que los programas a menudo acaban ejecutándose en entornos significativamente diferentes.

Algunas veces los programadores trabajan como en un túnel. Sólo ven las tareas específicas del momento, y no ven el escenario general de desarrollo del software. Esto puede suceder en grandes proyectos, en los que distintas personas realizan distintos módulos de código o transcurren largos periodos de tiempo entre la creación de unos módulos y otros.

Un ejemplo de este tipo de problema está relacionado con el uso de versiones específicas de DLL. Si no comprobamos la autenticidad y las versiones de estas DLL, puede que nos encontremos con una sorpresa desagradable. Existen muchos lugares posibles donde el entorno de ejecución está fuera del control de nuestro programa.

Otro ejemplo de este tipo de problemas sucede cuando un atacante utiliza de forma intencionada recursos de la máquina en la que se está ejecutando nuestro código. Puede intentar comerse toda la memoria disponible o hacer que los discos estén tan atareados que el tiempo de apertura de archivos expire antes de que se abran. Esto se puede manifestar también en el drenaje de recursos de la red para hacer que las conexiones de red requeridas superen el tiempo máximo.

Siempre que el código propio se está ejecutando en un entorno que puede estar casi totalmente controlado por un atacante. Los atacantes intentarán todas estas tácticas y más en un intento de romper un código débil.

---

[10] Netcat es un programa útil para comprobar aplicaciones que permiten acceso a la red. Es una utilidad sencilla para introducir información en una conexión de red. Puede trabajar en los modos TCP y UDP También es posible utilizarse como un servidor de red sencillo, permitiendo a los clientes conectarse y pasarle información. Originalmente se creó para UNIX, pero desde entonces se ha llevado a Windows. La versión de Windows está disponible en http://www.l0pht.com/~weld/netcat/.

Una solución posible es hacer un esquema de todos los posibles problemas que puedan surgir. Por supuesto, como es imposible imaginar el número exacto, es lógico examinar los posibles errores del código que pueden devolver nuestras funciones y actuar de acuerdo con ellos. No se debedar nunca por hecho que las llamadas a funciones van a funcionar siempre.

5.1.5 MALA COMPROBACIÓN DE ENTRADAS

"Entrada" es un término común en este sentido. Es posible definirlo como cualquier información que viene de una fuente externa al programa propio. El alcance de lo que se quiere decir con entrada incluye:

- Datos de archivo.
- Datos de red.
- Datos de entorno.
- Datos como resultado de comunicación entre procesos.

Se debe tomar en cuenta de que **un atacante empezará a romper programas intentando localizar todos los lugares donde se pueden introducir datos**. Éste es un hecho clave que debe recordarse, ya que los fallos que dependen de los datos rompen más programas que ningún otro tipo de fallos.

Por ejemplo, se utiliza un servidor *Web* de red que gestiona peticiones HTTP. Los administradores de este servidor nunca previeron que alguien pudiera intentar pedir un archivo con un nombre de 1000 caracteres. Esto puede hacer que el servidor se colapsase; además de detener las respuestas a las peticiones de todos los usuarios. Los diseñadores de este servicio no previeron los datos que su programa podía recibir, y no crearon ningún código relacionado con ellos. Ninguna petición de archivo a través de HTTP debería ser tan larga; sí existe una forma mejor de tratar esta situación. Depurar la entrada, o al menos comprobar su validez, pudiera haber ayudado a resolver este problema de forma significativa.

En el lenguaje VB.NET podemos realizar alguna aplicación que considere que todos los datos externos están contaminados y necesitan limpiarse antes de que se puedan usar de forma efectiva dentro del programa. Ésta no es mala idea, dado que la mayoría de los datos deben comprobarse y limpiarse o descartarse antes de que se utilicen realmente en nuestros programas.

La comprobación de la entrada es más efectiva cuando los programas saben lo que deben esperar y descarta la información que no se ajusta a ese formato. Es posible realizar algunas pruebas para eliminar datos "malos", pero a menudo estas comprobaciones no son completas y olvidan algunos detalles. Los filtros para toda entrada válida pueden ayudar de forma significativa a la seguridad general de los programas.

Si se quiere tener "seguridad" en las prácticas de programación, puede que sea de utilidad comprobar la entrada de todas las funciones. Esto limita la corrupción de los datos dentro de las aplicaciones, en caso de que una estructura de datos interna pueda ser comprometida. Se comprobara la entrada antes y después de que se llame a cada función para asegurar que está dentro de los "límites" aceptables.

5.1.6 MALA COMPROBACIÓN DE LOS LÍMITES

El no realizar una correcta comprobación de los límites es la causa de muchos de los tipos de ataques más populares. Las inundaciones de buffer basadas en *pilas* y en *acumulaciones* suceden porque cuando hay buffers que contienen datos en memoria, algunas veces un programa intenta meter un buffer grande en uno más pequeño. Los datos adicionales tienen que ir a alguna parte, y si un atacante puede manipular esos datos extra, puede obtener en gran medida el control de la máquina, haciendo que ejecute cualquier código que él elija. Las inundaciones de buffer son probablemente los ataques más comunes contra los sistemas actualmente. Suceden cuando un atacante introduce entradas en un programa para conseguir una inundación de buffer interna. Algunos lenguajes no son propensos a ataques de comprobación de límites, ya que implementan comprobaciones de límites apropiadas como parte de las especificaciones del lenguaje.

5.1.7 CONDICIONES DE ADELANTO

Una condición de adelanto es una situación en que se abre un recurso, pero antes de que se lea de él o se escriba en él, el atacante obtiene el control del mismo para introducirle datos o para hacer que el programa dé una salida de datos incorrecta.

Un ejemplo de esto sucede cuando un programa abre un archivo, y antes de que pueda leer o escribir en él, un atacante logra obtener el control del archivo y hacer que el programa escriba a un archivo diferente o lea datos falsos. Esto exige normalmente al atacante que ralentice primero la máquina desperdiciando ciclos de CPU o recursos de memoria. Algunas veces, los archivos pueden sobrescribirse o se pueden introducir datos falsos en una aplicación, haciendo que haga algo distinto de lo que se pretendía.
Ataques recientes de este tipo se han centrado en programas que implementan archivos temporales. El atacante borra el archivo temporal y lo sustituye por un archivo falso, que pertenece al atacante. Cuando el programa intenta escribir, escribe al archivo controlado por el atacante y la salida puede redirigirse a otro lugar. Esto ha causado serios problemas en algunos sistemas UNIX, pero el problema también puede surgir en los sistemas Windows.

Existen formas de evitar que los atacantes provoquen situaciones de adelanto. Primero, asegurase de verificar que los recursos son los que creemos que deben ser, antes de leer y escribir. Implementar un buen sistema de bloqueo puede evitar asimismo que ocurran algunas de estas cosas. Podemos colocar los archivos temporales en algún lugar al que no pueda acceder un atacante, como en un directorio privado.

5.1.8 ACCESO AL CÓDIGO

Microsoft Visual Studio .NET ofrece mayor control sobre la seguridad de aplicaciones en ejecución que el que proporcionaban versiones anteriores de *Visual Studio*. Aunque *.NET Framework* ofrece más control, requiere mayor responsabilidad por parte del programador durante el desarrollo de la aplicación. Existen en torno al código algunos problemas de seguridad que deberá solucionar para poder crear aplicaciones robustas, sencillas y prácticas para el usuario.

Las situaciones más frecuentes en las que se presentan este tipo de problemas son cuando:

☑ El usuario que ejecuta la aplicación le niega privilegios a la misma porque se ejecuta desde una ubicación que tiene prohibido el acceso a determinados recursos del sistema, según especificaciones del usuario. Por ejemplo, el usuario puede configurar al *CLR* para que niegue privilegios de archivo a cualquier aplicación que se encuentre almacenada en una unidad de red. Debe tenerse esto presente mientras se escribe el código, para que el código responda correctamente a las denegaciones.

☑ Los usuarios que tienen acceso a las aplicaciones Web de los servidores Web ejecuten en éstos código malintencionado o datos dañados.

☑ El servidor, según el modo en que se configure Visual Studio, corra mayor o menor riesgo de sufrir los ataques de código malintencionado.

5.1.8.1 . ACCIONES SEGURAS E INSEGURAS

El propósito del Sistema de Seguridad de Acceso al Código es permitir únicamente la ejecución de aquellas acciones consideradas como seguras, o de otra forma, impedir que las acciones consideradas como inseguras tengan lugar.

El sistema de seguridad de acceso al código de .NET asigna a su aplicación o componente permisos tales como acceso a archivos, interfaz de usuario y de red, como base para determinar qué operaciones seguras o inseguras puede realizar su aplicación. El conjunto de permisos asignados a su aplicación está basado en el nivel de confianza asignado a la misma. Las aplicaciones instaladas, incluyendo las aplicaciones instaladas por medio de un programa de instalación desde Internet, y ejecutadas en su equipo se consideran como de confianza elevada, por lo que tendrán todos los permisos disponibles. Por el contrario, los componentes cargados y ejecutados desde Internet tienen la consideración de desconfianza elevada y obtienen pocos permisos.

El sistema de seguridad de acceso al código de *.NET* utiliza medios sofisticados para determinar qué permisos se asignan a sus componentes o aplicaciones basadas en *Microsoft Visual Basic .NET*. La ubicación desde la que se ejecuta la aplicación es un factor de gran importancia a la hora de determinar qué permisos va a tener su aplicación. Por ejemplo, si el componente gráfico se carga mediante una aplicación ejecutada en su PC local, el componente gráfico tendrá asignado el permiso de eliminación de archivos. Sin embargo, si el componente gráfico se ejecuta directamente desde Internet (en realidad, los componentes o aplicaciones que se ejecutan directamente desde Internet se descargan, en primer lugar, a una caché de descarga especial de Internet contenida en su equipo y, posteriormente, se ejecutan) no tendrá asignado los permisos de eliminación de archivos. La ubicación desde la que se ejecuta una aplicación es una evidencia que utiliza el sistema de seguridad de acceso al código para determinar los permisos que se van a asignar a la aplicación. En el Capítulo siguientes se analizara la forma en la que podrá suministrar otros tipos de evidencias, tal como el nombre de su aplicación, al sistema de seguridad para poder asignar a su aplicación una serie de permisos particulares.

5.1.8.2 CÓDIGO DEMASIADO COMPLICADO

El código sencillo es menos propenso a errores, y menos errores significan menos problemas potenciales de seguridad. El código complicado es difícil de depurar y difícil de escribir. Simplificar el diseño y la implementación es una ventaja en todos los aspectos de la programación.

Reducir las redundancias cuando contribuyan a la complejidad; ser consistentes con la sintaxis de las llamadas y la elección de bibliotecas externas. Si existe más de una forma de hacerlo (y siempre existe), debe elegirse la más sencilla. Reducir la dependencia global de las bibliotecas externas, y crear las funciones propias más generales, de modo que se puedan utilizar fácilmente dentro del programa que se diseñen.

La complicación no es un modo de proteger nuestro código de los atacantes. El hecho de que parezca que puede ser muy difícil descifrarlo no hace que esto sea cierto. Los atacantes son a menudo algunas de las mentes más brillantes en el campo de la seguridad, y pueden encontrar y de hecho encuentran, fallos en el código demasiado complicado. Lo único que hace la complicación es añadir escondrijos y grietas donde pueden surgir los fallos. Cuanto más sencillo sea el código, más fácil será reparar sus fallos.

El realizar acertados y buenos comentarios dentro del código, facilita su comprensión y puede ayudar a ponernos rápidamente al día en código que no hemos tocado durante un tiempo. Esto es especialmente cierto en organizaciones con varios programadores o en planes de lanzamiento de programas en código fuente.

5.1.8.3 CÓDIGO DEMASIADO SENCILLO

Una advertencia en contra del código demasiado sencillo puede parecer extraña después de leer que la complejidad es enemiga de la seguridad, sin embargo debe tomase en cuenta que algunos problemas para los que debemos programar son, de hecho, complicados. Simplificando demasiado el problema, podemos crear también agujeros de seguridad. Si no tenemos en cuenta el alcance total de la finalidad del programa, puede que se nos pasen cosas por alto, o que agrupemos demasiadas cosas en una sola parte. Simplificar demasiado puede evitar que segmentemos el código para obtener las separaciones entre funciones más nítidas posible.

Un posible caso en que el código demasiado sencillo puede causar problemas es en la creación de un analizador sintáctico. Por ejemplo, el analizador sintáctico XML debe seguir las especificaciones, y si se simplifica demasiado, puede que no funcione o que ofrezca al atacante uno o dos puntos de apoyo sobre los que trabajar.

Tener demasiada complejidad no es bueno, como tampoco lo es hacer nuestro código demasiado sencillo. Se recomienda diseñar programas para que trabaje de la forma más sencilla posible, pero también que sea tan completo como sea posible.

5.1.9 AMENAZAS Y VULNERABILIDADES: ANÁLISIS Y DETECCIÓN

ANALISIS

Amenaza es la capacidad potencial de sufrir un ataque. **Vulnerabilidad** es el grado en el que la PC, red o aplicación puede sufrir un ataque. El daño se produce cuando se ejecuta la amenaza (es decir, se lleva a cabo el ataque) contra una vulnerabilidad. Es posible analizar y detectar los daños provocados con una determinada amenaza relativa a una vulnerabilidad, siempre que se considere que:

- Una amenaza nunca se materializa como un ataque.

Si la amenaza existente relacionada con una vulnerabilidad nunca se materializa, es porque: la "suerte" o "falta de conocimiento" ha impedido que ocurra un ataque (aun tiene que presentarse un atacante que descubra la vulnerabilidad).

- Es posible que la vulnerabilidad aparezca antes de que se produzca el ataque.

Se ha resuelto la vulnerabilidad relacionada con su PC, red o aplicación antes de que el atacante la descubriera

- El riesgo de ser descubierto in-fraganti supera a la posible recompensa.

Es posible que una vulnerabilidad no se explote dado que el riesgo percibido de ser capturado y castigado se percibe como muy elevado por los completos sistemas y herramientas de seguridad que acompañan a la aplicación.

- Otras vulnerabilidades pueden resultar más atractivas al atacante.

Este caso puede producirse si el atacante no prevé ningún beneficio si ejecuta el ataque.

Tener una vulnerabilidad en la PC, red o aplicación es similar a dejar la puerta del coche abierta en un estacionamiento público con las llaves puestas y volver varias horas más tarde para comprobar que *no* le han robado el coche. El coche no habrá sido robado por alguno de los siguientes motivos:

- Ningún ladrón ha descubierto el coche.
- Se descubre con rapidez que las llaves quedaron en el coche, y regresa a recuperarlas.
- Las cámaras de seguridad instaladas en el estacionamiento y el guardia de seguridad disuaden a los posibles ladrones.
- El ladrón considera que el VW sedan de color verde, mohoso y del año 1975 no merece la pena y prefiere hacerse del BMW del año que está estacionado al lado.

El análisis de amenazas es el proceso de revisión de la aplicación y de la identificación de las amenazas que se ciernen sobre la misma, incluyendo los riesgos relacionados con la PC o con la red donde se ejecute la aplicación. Debe aplicarse un análisis de amenazas para identificar las mismas y resolverlas antes de que un atacante tenga la oportunidad de causarle algún daño.

El análisis de las amenazas implica el estudio en detalle de los componentes que forman la aplicación y la forma en que se relacionan. Es posible comenzar[11] identificado todas las amenazas relacionadas con la aplicación, incluyendo aquellas que puedan parecer insignificantes y remotas. Después es necesario asignar prioridades a la lista de amenazas con el fin de centrar esfuerzos al intentar acabar con los riesgos más importantes.

Identificar y asignar prioridades

La vida está llena de riesgos. Muchas amenazas ponen en peligro los equipos, redes y aplicaciones, como:

- Gusanos o virus que puedan poner en peligro su red.
- Las aplicaciones de revelación de contraseñas utilizadas para acceder de forma no autorizada a su PC.
- Manipular de empleados en la empresa para obtener información confidencial.

Analizar las amenazas que se ciernen sobre la aplicación es un requisito previo para desarrollar un plan eficaz de acción. El análisis de amenazas sirve igualmente para detectar las amenazas que ponen en peligro los equipos. Los pasos básicos son:

1. **Identificar** todas las amenazas (reales o imaginarias) posibles.

2. **Asignar** prioridades a las amenazas en función de la probabilidad de que se produzcan y de su severidad, y asignar prioridades a las vulnerabilidades en función de la probabilidad de ser descubiertas y de la severidad del riesgo asociado.

Para la ***Identificación de las amenazas*** debe elaborarse una lista exhaustiva de todas las amenazas a las que la aplicación se puede presentar. La tormenta de ideas y pensar cómo lo haría un atacante, son buenos métodos para crear dicha lista, de la misma forma en que se elaboramos una lista exhaustiva de escenarios de pruebas de seguridad.

Esta tarea debe contemplar:

- Analizar el diseño global de su aplicación y de los componentes (.EXE y .DLL) que la forman.
- Identificar todos los métodos públicos expuestos por la aplicación.
- Identificar todas las entradas realizadas en la aplicación, incluyendo las entradas efectuadas por los usuarios, la configuración del registro y los archivos de entrada.
- Pensar en la forma en que la aplicación informa de los errores.
- Bloquear todos los canales de comunicación como, por ejemplo, las conexiones de base de datos y HTTP, existentes entre los componentes de la aplicación.

[11] Aunque es posible efectuar el análisis de amenazas en cualquier instante del ciclo de desarrollo, resulta preferible llevarlo a cabo en la fase de diseño. Al actuar así se pueden identificar los problemas de diseño que pueden exponer a su aplicación a amenazas innecesarias. Al identificar las vulnerabilidades en la fase de diseño se tiene la oportunidad de modificar el diseño de la aplicación en lugar de tener que enfrentarse a un rediseño caro y difícil cuando se descubra la vulnerabilidad en otra fase posterior del ciclo de desarrollo.

- Enumerar los tipos de datos que pasan por dichos canales de comunicación.

En la ***Asignación de prioridades*** a las amenazas es necesario cuales son más inminentes e importantes (más reales) que otras. La severidad de una determinada amenaza está basada en dos factores:

- La probabilidad de que un ataque se produzca en función de la amenaza.
- El daño o pérdida potencial que sufriría como resultado del ataque.

Las amenazas que tienen asociado un valor alto en ambas categorías deberán tener la mayor prioridad en la lista. Si la aplicación es un programa de banca electrónica por Internet que permite a sus clientes ver y actualizar la información personal (por ejemplo, nombre, dirección y número de teléfono) será buen punto de partida para un atacante hacerse de la idea con la identidad de un usuario. Una amenaza para esta aplicación sería que un atacante consiguiera reunir suficiente información (descubriéndola en la Web o interceptándola en su camino entre el cliente y el servidor Web) como para asumir la identidad de un usuario y extraer dinero de su cuenta. La probabilidad de este tipo de ataque sería elevada, dada la visibilidad de la aplicación. Además, el potencial de pérdidas es también elevado. Este tipo de amenazas debe tener asignada una prioridad elevada en la lista.

Una excelente forma de rastrear todas las amenazas que ha identificado es introducir un error para cada una de las amenazas. Incluso sin utilizar una herramienta de seguimiento de errores, es posible seguir el rastro de todos los problemas utilizando cualquier otra aplicación como, por ejemplo, una hoja de trabajo de Microsoft Excel, una base de datos de Microsoft Access o un documento de Microsoft Word. Realizar el seguimiento de las amenazas de esta forma permite seguir el rastro de los problemas de seguridad de la misma forma en que se hace con los demás errores. Los errores introducidos a propósito deben incluir la siguiente información, básicamente:

- Una descripción completa de la amenaza.
- Opcionalmente, la categoría (o categorías) a la que puede pertenecer la amenaza (para esta operación, podrá utilizar el análisis STRIDE mostrado en la tabla 5.1.

La prioridad de la amenaza que deberá definir para representar la prioridad general que haya establecido. Esta prioridad deberá estar basada en la probabilidad de la amenaza y en su severidad. La ventaja de incluir información sobre la amenaza o la lista de los problemas en una base de datos de errores es que todos los problemas relacionados con la calidad del producto (errores y problemas de seguridad) se almacenarán en la misma ubicación. No tendrá que mantener varios documentos. Después de completar el análisis de amenazas de la aplicación y determinar las vulnerabilidades más importantes, deberán decidirse cuál es la forma más idónea de enfrentarse a cada una de ellas. Pode ignorarse las vulnerabilidades, esperando que el viento las disperse y confiando en la buena suerte, o pode efectuarse acciones correctivas para mitigar las amenazas que haya identificado.

ATAQUE	DESCRIPCIÓN
Suplantación (Spoofing) de la identidad de usuario	Un sencillo ejemplo es cuando un atacante inicia una sesión como otro usuario sin el conocimiento o el permiso de este último El atacante podrá realizar todas las operaciones que pudiera llevar a cabo el usuario auténtico. El daño potencial se puede mitigar si el sistema solicita información adicional que el atacante no conoce.
Alteración (Tampering) de los datos	Cualquier dato que pudiera modificar el atacante sin la autorización del usuario legítimo. Entre estos datos se incluyen los datos personales almacenados en uña base de datos, las entradas de claves del registro, el contenido de archivos ejecutables y las opciones de la aplicación almacenadas en un archivo del disco.
Repudio	Cuando una de las partes niega haber realizado una acción y la otra parte es incapaz de demostrar lo contrario. En general, el no repudio significa que debe conservar evidencias de cada uno de los pasos que se hayan realizado en la aplicación o el proceso.
Revelación de información	Cualquier ataque en el que el atacante pueda obtener información sin autorización pertenece a esta categoría.
Denegación del servicio	Es un ataque que puede manifestarse de varias formas afectando a los servidores e incluso las redes.
Evaluación de privilegios	Este ataque implica la obtención de más privilegios de los que debiera de tener.

Tabla 5.1 STRIDE: Una técnica muy popular utilizada en Microsoft para realizar el análisis de amenazas es clasificar cada amenaza entre las distintas áreas potenciales de ataque utilizando el modelo de amenazas de seguridad denominado STRIDE que ha sido definido por la Microsoft Security Task Foce. STRIDE es la abreviatura formada por los tipos de amenazas listados en la Tabla 5.1, la primera letra de cada amenaza forman la palabra STRIDE.

DETECCIÓN

No es posible contemplar todos los tipos de ataques durante el diseño y desarrollo de la aplicación y no podrá tenerse en cuenta todos los posibles problemas de seguridad durante la revisión del diseño o durante las pruebas. Por tanto, existe la posibilidad de que se distribuya la aplicación con ciertos defectos de seguridad.

La detección implica el descubrimiento de alguna señal de que un ataque está a punto de tener lugar o identificar que un ataque está siendo realizado.

Detección temprana

Intentar detectar cuándo se va a producir un ataque (tal como un robo) resulta más difícil que detectar el robo cuando se esté produciendo. Las técnicas de detección temprana implican el descubrimiento de los intentos malintencionados. ¿Cómo puede determinar este tipo de intentos? Salvo que pueda leer la mente del atacante, se tiene que emplear otras técnicas para descubrir los planes. Estas técnicas requieren observar el patrón de comportamiento de uno o más atacantes potenciales y, posiblemente, interceptar pruebas del plan del atacante, tales como un correo

electrónico de un atacante potencial que indique que se está preparando el ataque. Sin embargo, en el caso de los ataques informáticos, no deberá buscar un atacante determinado, sino que intentará localizar evidencias (por ejemplo, patrones o anomalías en elementos tales como los registros del sistema, cabeceras HTTP de las aplicaciones en línea y paquetes de red) que sugieran que alguien está planificando atacarle.

Por ejemplo, pueden controlarse los paquetes enviados al puerto TCP del servidor SQL en busca de una firma, lo que le indicaría que alguien está intentando lanzar un ataque de denegación de servicio al estilo *SQL Slammer*. Incluso si se descubren evidencias de que un ataque está a punto de tener lugar, tal vez encuentre dificultades a la hora de probar que el ataque va a tener lugar realmente o para identificar a los atacantes.

Puede llevar a cabo un mecanismo de detección temprana de ataques sobre la propia aplicación o sistema de las siguientes formas:

- Analizar los grupos de noticias y los sitios Web relacionados con la seguridad (por ejemplo, www.cert.org) en busca de informes sobre ataques realizados a aplicaciones similares a la propia, o ataques efectuados sobre componentes o sistemas que la aplicación utilice. Este tipo de trabajo pone en alerta que un determinado ataque tenga más posibilidades de producirse y proporcione la oportunidad de defenderse contra él.
- Proporcionar información a los usuarios (por ejemplo, notificarles que se ha producido un intento fallido de entrar en la cuenta y la hora en que se produjo el último intento con éxito de inicio de sesión) que pueda indicarles que alguien ha intentado iniciar una sesión (o que lo ha conseguido) utilizando su nombre de usuario y su contraseña. También podrá obtener la ayuda de los usuarios para detectar actividades sospechosas (suponiendo que se proporcione a los usuarios un mecanismo de generación de informes que sea sencillo).
- Diseñar las aplicaciones para que registren todas las actividades que sean sospechosas. Por ejemplo, si la aplicación detecta la recepción de un archivo de comandos HTML o de una instrucción SQL como parte del nombre de usuario de inicio de sesión, deberá escribir dicho nombre de usuario en algún lugar para analizarlo. Además, también deberá registrar cualquier intento de inicio de sesión que se haya producido utilizando nombres de usuario no reconocidos (o varios intentos fallidos de iniciar una sesión en una cuenta de usuario en concreto). Debe revisarse con regularidad los registros para ver si puede detectar un patrón que indique que se está produciendo un ataque o que ya ha tenido lugar.

Resulta difícil *detectar un ataque* que esté teniendo lugar salvo que pueda comprobar los efectos del mismo cuando se esté produciendo (por ejemplo, cuando un ataques DoS ralentiza el comportamiento de su red o aplicación antes de bloquearla por completo). El principal objetivo es asegurarse de que se podrá detectar el momento en que se produce un ataque. El peor caso posible es que un ataque se produzca y no sea detectado (ya sea porque no se han definido los mecanismos de detección necesarios o porque el mecanismo utilizado para registrar las actividades sospechosas, registros de auditoria, han sido eliminados o comprometidos por el atacante).

Deben ejecutarse los siguientes pasos para detectar si una aplicación o un sistema ha sido víctima de un ataque:

- Registrar todas las actividades sospechosas. En caso de que la aplicación registre todos los errores inesperados que ocurran (incluyendo determinada información tal como valores de variables o parámetros que no se esperaban). Utilizar controladores de excepciones del tipo *Try. . . Catch* en toda su aplicación en unión de la clase *System.Diagnostics.EventLog* para registrar todos los errores inesperados que se produzcan.
- Configurar el equipo en el que se ejecutan sus aplicaciones para registrar[12] todos los reinicios no programados, que pueden ser una evidencia de que un atacante está manipulando los archivos del sistema.
- Registar todos los intentos fallidos de inicio de sesión en la aplicación o en el sistema.
- Emplear un sistema de detección de intrusos, *IDS*[13] comercial para ayudarle en la detección de los intrusos.
- Si la empresa cuenta con un elevado número de equipos, realizar inventarios de hardware, es decir, analizar la red en busca de todos los sistemas que se encuentren conectados, para comprobar que no se ha conectado sin autorización ningún equipo a su red. Si un atacante puede introducir una computadora (o cualquier otro hardware tal como un módem conectado a otra computadora) detrás del cortafuegos empresarial, el atacante podrá acceder a otros equipos de la empresa a los que no podría acceder desde el otro lado del cortafuegos.

Si se utilizan registros de auditoria para alertarle de las actividades sospechosas pero el atacante consigue modificar estos registros (eliminando cualquier evidencia del ataque), sus mecanismos de detección no servirán para nada. Y lo peor de todo, no tendrá ninguna evidencia de que su mecanismo de detección se encuentra incapacitado o que ¡está siendo atacado!

La mejor opción es implementar una técnica de defensa en profundidad y de varias capas. La idea de este capitulo es proporcionar las bases para lograr diseñar una aplicación con la hipótesis de que los mecanismos de detección de seguridad que está utilizando pueden ser, a su vez, atacados y comprometidos. Plantearse a sí mismo las siguientes cuestiones: ¿Cuál sería un

[12] Los reinicios del sistema se registran de forma automática si está utilizando Windows 2000 Server o Windows Server 2003

[13] Los sistemas de detección de intrusos son aplicaciones de software que emplean una amplia variedad de técnicas automatizadas para identificar una intrusión o un ataque. Algunas de las técnicas utilizadas son:

* Detección de la firma del ataque Esta técnica es similar a la forma de trabajar que tienen los programas antivirus para detectar la presencia de virus. El sistema de detección de intrusos busca patrones tales como ciertos tipos de paquetes de red, ciertos cambios en los archivos del sistema o ciertas modificaciones en los registros de auditoria que indicarían un ataque.

*Detección de anomalías Con esta técnica, el sistema de detección de intrusos conoce el comportamiento esperado de su aplicación o del sistema informático. El IDS busca posibles desviaciones con respecto al comportamiento esperado. Por ejemplo, puede programarse el IDS con el conocimiento de que estadísticamente sólo el 10 por ciento de todas las ventas realizadas en el sitio de comercio electrónico basado en Web tienen un descuento sobre los precios indicados y que el descuento medio es del 15 por ciento. El sistema le alertará de un posible ataque si detecta que el 20 por ciento de las transacciones se efectúan con un descuento medio del 30 por ciento.

mecanismo de respaldo adecuado en caso de que fallara el mecanismo principal de detección? ¿Qué método alternativo puede utilizarse para detectar el mismo tipo de ataque? El uso de mecanismos redundantes para detectar un ataque es muy recomendado. Por ejemplo, haga que varios sistemas de auditoría registren los mismos fallos. Si, por ejemplo, la aplicación sufre un error provocado por la ausencia de los privilegios necesarios para acceder a una determinada tabla de la base de datos, es necesario hacer que los registros de la aplicación y de la base de datos registren este fallo. Además, utilice diversas técnicas para detectar el mismo tipo de ataque. Por ejemplo, capturar una copia (respaldo) de la base de datos cada cierto tiempo y analizar las diferencias, tales como los cambios que se produzcan en la lista de nombres de usuarios (o cualquier otra información crítica), que deben reflejarse en los registros de auditoría pero que no aparecen en ellos. Este hecho podría resultar una evidencia de que un atacante ha eliminado ciertas actividades del registro. Este tipo de comparaciones de la base de datos (que no aparecen en los registros de auditoría) pueden ser la única forma de detectar las tareas que haya llevado a cabo el atacante.

Aunque sería fantástico que los sistemas automatizados pudieran detectar y evitar todos los tipos de ataques, la verdad es que los sistemas automáticos no pueden hacerlo. Además, los sistemas automatizados encargados de monitorizar la red, equipos o aplicaciones (incluyendo el código de detección contenido en la propia aplicación) en busca de signos de ataque, también pueden ser víctimas de un atacante. Se requiere la supervisión humana para determinar si un ataque ha tenido lugar y para garantizar que los sistemas responsables de detectar los ataques están trabajando tal y como se esperaba. Será responsabilidad de los humanos, por ejemplo, la revisión periódica de los registros de auditoría y el enjuiciamiento de si un ataque se ha producido. También corresponde a los seres humanos realizar la defensa ante un ataque, identificando y aislando los sistemas afectados, determinando cuál es la mejor solución, aplicándola y devolviendo los sistemas a un estado seguro y productivo.

5.2 RESOLUCIÓN DE PROBLEMAS DE SEGURIDAD PARA LA PROGRAMACIÓN

Los conceptos relacionados con el software seguro son, a menudo, "sencillos", pero la mayoría de los programadores raramente los consideramos al diseñar e implementar programas. Los siguientes son los principios fundamentales en la creación de software seguro. Si se siguen estos sencillos principios, es posible mitigar la mayoría de los fallos relacionados con la seguridad del software actual.

Los fallos de seguridad raramente son obvios. A menudo, no se muestran durante las condiciones normales de una prueba. Los atacantes se desvían de su camino para encontrarlos alimentando el propio programa con enormes piezas de información en casi todos los formatos concebibles. También manipulan los depuradores y desensambladores para intentar reconstruir lo que está haciendo el código propio.

La seguridad se convierte en un problema cuando el código de uno actúa en un nivel de autorización diferente al de los usuarios. Esto puede significar que tiene acceso a recursos que un usuario normalmente no tendría, o se utiliza para autorizar acceso a algo. Subvirtiendo este código, un atacante posiblemente puede obtener privilegios elevados, lograr información que no

debería, leer y escribir en recursos más allá del ámbito de sus derechos, y hacer incluso cosas más insidiosas, como reiniciar el servidor a voluntad o llevarlo a un estado en el que no se pueda utilizar.

☑ Dar a nuestro software los mínimos privilegios posibles

La idea de los privilegios mínimos es, a menudo, el concepto más difícil de aceptar para los programadores. El software en sí mismo puede ejecutarse a diferentes niveles de privilegio. Es importante darse cuenta de que un programa que se ejecuta con derechos de sistema da a todas las partes del programa derechos como sistema. Esto significa que lee y escribe archivos y el Registro con derechos de sistema. También significa que puede ser capaz de hacer cosas que nunca se pretendió, como producir un shell de comandos con derechos administrativos o sobrescribir una DEL con código arbitrario. Ésta es la razón por la se debe "segmentar" el código tanto como sea posible, y asignarle niveles de mínimo privilegio a cada uno de estos segmentos. Si no se necesitan privilegios adicionales para una parte del código, no se permite que se ejecuten a este nivel. Dividir el código en "contenedores" de seguridad ayuda también a inspeccionar, probar y mantener nuestro código.

☑ Comprobación de los códigos devueltos

Crear funciones que devuelvan códigos de error y realmente comprobados. Esto es de la mayor importancia. Se comprueba todo lo devuelto de una función llamada. Asegurándose que los archivos que se abren y los recursos que están disponibles. Esto ayuda a la aplicación a que sea más robusta, además de ayudar a prever formas en las que puede fallar la aplicación. Esto es especialmente importante cuando se utilizan librerías externas escritas por terceras personas y que gestionan llamadas del sistema. Cuando se utilizan llamadas externas, puede que no siempre se obtenga lo deseado, y es excepcionalmente importante verificar que las llamadas se ejecuten correctamente.

En Win32 es posible comprobar códigos de error utilizando la llamada *GetLastError()*. Esta llamada devuelve el último código de error de una llamada API de Win32 al hilo actual. Cada hilo mantiene su propio código de error y puede devolverlo a través de esta llamada.

☑ Evitar hacer suposiciones

Hacer suposiciones puede llevar al desastre. Recuerde que el código puede estar ejecutándose en un entorno donde un atacante puede controlar todos o la mayoría de los aspectos. Tiene tiempo para establecer situaciones extrañas para intentar hacer que el software que uno mismo diseño, falle. Esto enlaza también con la comprobación de todos los códigos devueltos. No debe suponerse que hay recursos suficientes disponibles. No debe suponer que una variable de entorno se formateará adecuadamente. No debe suponer que un usuario no va a editar un archivo de configuración o una clave de Registro manualmente. Es importante considerar todas estas cosas. No de nada por supuesto en ningún caso donde se toman datos externos de fuentes que están fuera del control directo del programa. Puede ser acertado incluso suponer que la información que el programa controla puede ser manipulada, y comprobar esta posibilidad también.

☑ **Comprobación de nuestro código**

El control de calidad es una parte del proceso de desarrollo de software que a menudo se descuida, pagando el precio de permitir que salga al exterior software defectuoso. Se necesita un nuevo enfoque de esta industria, hacia la producción de código relativamente libre de fallos. El control de calidad y las comprobaciones tradicionales tienden a pasar por alto los tipos de fallos de seguridad. Los fallos relacionados con la seguridad aparecen normalmente fuera de la gama de uso "normal". Puede que sea necesario formar a las personas que realizan pruebas de control de calidad para que comprueben también lo que está "fuera" de los parámetros normales. Es buena opción establecer entornos extraños y hostiles y que trabajen en los programas desde ahí. Alimentar el código con comandos extraños e inusuales, y analizar cómo responde. Es imposible comprobar todos los casos, pero al menos intentar comprobar tantos como se pueda antes de que se comercialice el software.

☑ **Cerrar en caso de fallo**

"Cerrar si falla" es una filosofía que tiene mucho significado en todos los aspectos de las implementaciones de seguridad. Significa básicamente que si alguien hace que el sistema falle, casi siempre es mejor fallar en un estado donde está denegado todo el acceso, que hacerlo en uno donde está aceptado. Esta filosofía puede abrir las puertas a un ataque de negación de servicio, pero evita que el atacante subvierta aún más nuestro sistema de software.

Los cortafuegos, mientras funcionan normalmente, permiten que el tráfico fluya a su través en una situación estrictamente especificada. Si un atacante puede romper el cortafuegos y hacer que se cierre, sin embargo, todo el tráfico deja de fluir a la red situada detrás del cortafuegos. Esto significa que inutilizar un cortafuegos no aumenta la vulnerabilidad de los sistemas que están detrás de él. Sí significa, sin embargo, que todos los sistemas que están detrás del cortafuegos están desconectados de la red hasta que se solucione el problema.

☑ **Ser paranoico**

La idea en este punto no es la de querer sembrar el pánico en el corazón de los programadores. "Ser paranoico" es una sugerencia general hecha para obligar a ser consciente que hay gente intentando romper el software. Puede pensarse que nosotros somos inmunes a esto por la razón que sea, pero es posible que si alguien ha utilizado el software de uno antes que nosotros, ese alguien lo haya utilizado mal. Puede que no estuvieran intentando romperlo intencionadamente, pero algunas veces las peores pesadillas en materia de seguridad ocurren porque alguien que no se daba cuenta de lo que estaba haciendo introdujo un fallo en una parte del software mal diseñada.

5.2.1 AMENAZAS Y VULNERABILIDADES: PREVENCIÓN Y RESPUESTA

Una vez que haya identificado las amenazas de la aplicación, deberá adoptarse medidas preventivas para mitigar o eliminar las amenazas de alta prioridad que se hayan identificado. Si no es posible eliminar completamente una amenaza, tal como los ataques de Denegación de Servicio (DoS), debe diseñarse la aplicación con el objetivo de reducir las consecuencias de un ataque.

Disminución de las amenazas

La Tabla 5.2 lista algunos ejemplos de los tipos de ataques más frecuentes y de las posibles técnicas a utilizar para mitigar cada ataque.

TIPO DE ATAQUE	DESCRIPCIÓN	TÉCNICAS DE MITIGACIÓN
Superación del lU	El atacante intenta conectarse directamente a una base de datos o a un objeto servidor evitando la interfaz de usuario de la aplicación.	• En todos los métodos *Public*, comprobar que el usuario que ha iniciado la sesión ha sido autenticado por la aplicación antes de llevar a cabo una tarea autorizada. • Reducir la superficie de ataque de la aplicación servidora haciendo públicas únicamente aquellas funciones que lo necesiten realmente. • En el caso de una aplicación servidora Web, bloquear el servidor Web. • Bloquear la base de datos *back-end* que utiliza el servidor Web.
Alteración de los datos o de la entrada	El atacante intenta pasar datos para forzar un fallo catastrófico o utilizar la entrada para revelar secretos o modificar datos. Los ataques de inyección SQL y los ataques de programación de sitio cruzado son ejemplos de técnicas que se pueden utilizar para modificar datos.	• Validar toda la entrada, incluyendo la entrada realizada directamente por el usuario, los archivos de entrada y los valores de la clave del registro que utilice la aplicación. • Mitigar todas las amenazas relacionadas con la entrada, utilizando las técnicas analizadas en el Capítulo 7, tales como las amenazas de inyección SQL.
Denegación de servicio	Un atacante intenta provocar un fallo catastrófico en la aplicación o forzarla a que consuma grandes cantidades de recursos tales como la memoria. También se le conoce como ataque de negación de servicio o denial of service (DoS).	• Proporcionar un cuadro de diálogo de inicio de sesión en el que sólo los usuarios autorizados puedan escribir información. • Limitar la longitud (a un tamaño razonable) de todas las entradas realizadas por el usuario. • Diseñar la aplicación para que sea capaz de dar servicio a muchos más usuarios de los probables. En el caso de aplicaciones Web, presentar una página HTML introductoria al usuario de pequeño tamaño, o detecte cuándo se están recibiendo más peticiones de las que se puedan atender y presente en ese caso un mensaje del tipo lo siento, la aplicación no se encuentra disponible en estos momentos seguido de mensajes de ayuda que indiquen al usuario cómo puede obtener más información.

		• Controlar y registrar el número total de asignaciones de objetos, frecuencia de las peticiones, empleo de la memoria del sistema, asignaciones de las conexiones con la base de datos y empleo del espacio en disco en función del tiempo. Analizar los registros, identificando los cuellos de botella de rendimiento que tenga la aplicación y resuélverlos.
Intercepción de los datos	El atacante puede interceptar y modificar los datos que el cliente envía al servidor.	• Utilizar técnicas de cifrado y de hashing para proteger los datos o para detectar que han sido modificados, tal y como se indica en el punto anterior. • Utilizar canales seguros y cifrados, tales como SSL, para pasar los datos confidenciales.
Revelación de contraseñas	El atacante intenta utilizar la fuerza bruta para adivinar la combinación del nombre de usuario y de la contraseña para iniciar una sesión.	• Imponer una directiva de contraseñas por la que las contraseñas deban contener, al menos, ocho caracteres, utilicen mayúsculas y minúsculas, números y símbolos. • Si un usuario no escribe la contraseña correcta después de un número determinado de intentos (por ejemplo, tres intentos), impedir los intentos posteriores de inicio de sesión para ese mismo usuario durante un determinado periodo de tiempo. Por ejemplo, Windows realiza una pausa de varios segundos después de que se produzca un determinado número de intentos fallidos de iniciar una sesión, dificultando la tarea de las herramientas de revelación de contraseñas en su intento de probar contraseñas aleatorias en rápida sucesión. Además, puede bloquearse una cuenta de usuario después de que se produzca un determinado número de intentos fallidos de inicio de sesión, de forma similar a cómo actúan los cajeros automáticos de los bancos cuando se tragan la tarjeta de crédito después de intentar introducir varias veces sin éxito el PIN correcto de la tarjeta. • Registrar todos los intentos fallidos de inicio de sesión (así como los intentos que hayan tenido éxito) y notificar al usuario esta situación. También puede presentar al usuario la fecha y la hora del último inicio de

		sesión que haya conseguido realizar con éxito como forma de alertar al usuario de que alguien ha intentando iniciar una sesión en su cuenta cuando no estaba utilizando la PC.
Hacerse pasar por otro usuario	Un atacante es capaz de llevar a cabo un ataque haciéndose pasar por otro usuario. El ataque se realiza de tal manera que el usuario no puede probar que él no ha efectuado dichas acciones. Este ataque recibe también el nombre de ataque de repudio.	• Proteger contra la revelación de contraseñas. • Registrar todas las actividades, incluyendo el tiempo en que se produjo el ataque. • Solicitar credenciales adicionales, tales como un número PIN secreto, cuando el usuario realice una actividad importante tal como iniciar una compra de acciones o adquirir un determinado producto.

Tabla 5.2 Tipos de Ataques y posible mitigación.

Responder a un ataque

Debe ejecutarse una serie de pasos para responder con efectividad a un ataque:

- **Impedir nuevos daños:** Desconectar los equipos afectados o detener las aplicaciones afectadas.
- **Conservar las evidencias:** Con el propósito de identificar al atacante y determinar el nivel de daño sufrido, realizar una copia de seguridad de los registros del sistema y de los registros de la aplicación, y analizar los equipos afectados.
- **Analizar el daño**: Aquí es necesario plantear ciertas preguntas: ¿Qué equipos o aplicaciones han sido afectados por el ataque? ¿Qué acciones ha realizado el atacante?
- **Identificar la causa raíz:** Para impedir que el ataque se vuelva a producir es necesario dar respuesta a las siguientes preguntas: ¿Cómo se ha realizado el ataque? ¿Qué vulnerabilidad es necesario corregir para impedir futuros ataques?
- **Corregir el problema:** Si se ha identificado con éxito la vulnerabilidad, tomar medidas para garantizar que el mismo ataque no se va a repetir con éxito. Por ejemplo, si existe un parche de seguridad disponible para solucionar el problema, instalar el parche en todos los equipos y asegurarse que en todos los nuevos equipos también se instalará.
- **Comprobar la solución:** No hay nada peor que implementar una solución acentuando la vulnerabilidad de la aplicación en lugar de resolver el problema.
- **Restaurar las aplicaciones y sistemas hasta recuperar el estado correcto** Esta operación puede implicar la pérdida de datos. Sin embargo, es necesario impedir que los equipos o aplicaciones infectadas entren de nuevo en operación ya que podría revivir el ataque. Además, de evitar la rehabilitación de los equipos que contengan todavía datos *corrompidos* o *corruptos*.
- **Volver a distribuir la aplicación:** Si para corregir el problema es instalar un parche en la aplicación, debe volver a distribuirla. En el caso de aplicaciones *Web*, esta operación puede resultar tan sencilla como volver a instalar la aplicación en el servidor Web.

- **Supervisar:** Búsqueda de indicios para detectar el mismo ataque y buscar otras formas de ataque utilizando los mecanismos de análisis.

Como parte de la solución a un problema de seguridad debe conocerse la manera de cómo prepararse para responder. Debe realizar todos los esfuerzos posibles para proteger completamente la aplicación por medio de diseño orientado a la seguridad, ejecución de pruebas y revisión de versiones (beta) antes de comercializar la versión definitiva. Los problemas de seguridad que surjan después de distribuir la aplicación pueden provocar daños a los usuarios, dañar la reputación de uno como programador y tener una solución costosa. Sin embargo, a pesar de que realice los mayores esfuerzos para proporcionar un producto seguro, es necesario estar preparado (de antemano) para lo peor.

El momento de prepararse para responder a un ataque es antes de distribuir la aplicación y antes de que aparezca el primer problema de seguridad. Debe elaborarse un plan de respuesta que prevea los siguientes puntos:

- **Notificación a los usuarios** ¿Cómo deberá informar a los usuarios de que existe un problema? Será la propia naturaleza de la aplicación y la forma en que la haya distribuido la que defina el mecanismo de notificación que deberá utilizar. Si la aplicación es una aplicación Web, puede incluir una respuesta en la página Web de su aplicación. Si la aplicación está basada en *Windows Forms*, puede enviar un correo electrónico (además de introducir la información en el sitio Web de la empresa) para notificar el problema a los usuarios registrados.
- **Implementación** ¿Cómo se distribuirá el parche de seguridad a los clientes? Debe pensarse en un mecanismo de implementación antes de comercializar la aplicación. Por ejemplo, si resulta apropiado y adecuado para la aplicación, tal vez resulte correcto diseñar una función de actualización automática gracias a la cual la aplicación busque periódicamente actualizaciones y las instale automáticamente. Se trata de un ejemplo en el que planificar por anticipado y preparar una respuesta antes de que la aplicación se comercialice le permitirá diseñar funciones que ayuden a mejorar la satisfacción global de los clientes y disminuir los costos de implementación a largo plazo.

5.2.2 DETECCIÓN DE INTRUSIONES

La mayor parte de los programadores de aplicaciones en red se enfrentan en algún momento de su carrera a un suceso de intrusión en la seguridad. Disponer de un plan de detección de intrusiones posibilita que la notificación se realice pronto, se aminoren al máximo las consecuencias y se agilice la recuperación. Microsoft ofrece varias herramientas para detectar intrusiones, como por ejemplo el registro de sucesos.

La detección de intrusiones es el seguimiento y la notificación de una actividad no autorizada en un equipo o red monitorizados. Las intrusiones pueden provocar el cambio de los derechos de seguridad de los usuarios y archivos, la instalación de archivos que son "*caballos de Troya*" o el acceso inapropiado a la información. La detección de las intrusiones se lleva a cabo a través del examen de los registros del sistema y de la configuración e implementación de servidores de

seguridad, software antivirus y sistemas especializados en detección de intrusiones (IDS, *Intrusion Detection Systems*). Cuando se percibe una actividad sospechosa, se debe investigar y resolver.

Tipos de intrusiones

Las herramientas de detección de intrusiones intentan monitorizar e informar de actividades malintencionadas, como son el acceso no autorizado a archivos y sistemas, ataques (distribuidos) de denegación de servicio, gusanos, archivos que son caballos de Troya, virus informáticos, problemas derivados de la saturación de los búferes, la redirección de aplicaciones, la suplantación de identidad y de datos, ataques DNS, ataques mediante correo electrónico, daños a contenido o datos, y la lectura no autorizada de información confidencial.

Funcionalidad de la detección de intrusiones

La detección de intrusiones implica algunas de las funcionalidades siguientes: alertas, registro, elaboración de informes y prevención.

Las **alertas** son mensajes informativos en tiempo real y de alta prioridad que los procesos de detección de intrusiones envían a los administradores de una red con el fin de advertirles de que se ha producido un suceso de gran riesgo para la seguridad.

Todos los sistemas de detección de intrusiones deben grabar los incidentes notables relacionados con la seguridad en un archivo de **registro** junto con la fecha y hora del suceso, el riesgo y otra información detallada.

Una buena herramienta de detección de intrusiones debería proporcionar **informes** de administración que analicen los archivos de registro con el objeto de disponer de estadísticas y conocer tendencias. Las herramientas avanzadas de detección de intrusiones consiguen identificarlas y evitar que el ataque tenga éxito.

Creación de un plan de detección de intrusiones

La creación de un plan de detección de intrusiones conlleva el establecimiento e implementación de directivas de seguridad, la documentación de referencias para el sistema y la configuración de herramientas que permitan monitorizar las actividades no autorizadas. La creación de un plan de detección de intrusiones incluye los siguientes pasos:

☑ **Inventariar los sistemas que hay que proteger**

Debe realizarse un inventario de la red de modo que sepa lo que está protegiendo. Es especialmente importante observar qué sistemas carecen de la aplicación Visor de sucesos, *NTFS* o *Active Directory* puesto que su protección y administración se pueden dificultar. Considere el uso de *System Management Server* de *Microsoft* para automatizar la realización de inventarios y la detección de los dispositivos de red. Evalúe el riesgo de incidentes de seguridad y comunique sus conclusiones a la dirección.

☑ Crear una directiva de seguridad

Los administradores de seguridad deben establecer directivas de seguridad por escrito, que describan las directrices predeterminadas para mantener protegidos los sistemas. Estas directivas deben identificar quién puede tener acceso a qué activos informáticos, su uso aceptable, las directivas relativas a las contraseñas, restricciones de software y la configuración predeterminada de la seguridad de los sistemas. La mayor parte de las directivas de seguridad del sistema Windows se almacenan en el equipo local mediante la directiva de seguridad local.

☑ Proteger los sistemas

Una vez identificados los sistemas, es crucial comprobar que están configurados de forma segura. *Microsoft* dispone de varios recursos de seguridad, herramientas, listas de comprobación y documentos de procedimientos que pueden ayudarle a reforzar la seguridad de los sistemas *Windows*. Las plantillas de seguridad son el método que prefiere *Microsoft* para proteger la configuración de seguridad de un sistema local. La herramienta *Microsoft Baseline Security Analyzer* analiza los puntos débiles de la seguridad de los sistemas Windows más comunes y realiza recomendaciones en consecuencia.

☑ Documentar referencias de sistemas

El signo más común de intrusión es un comportamiento en el sistema o una modificación del mismo que se desvíe de lo esperado. Con el fin de saber lo que se considera normal, debe examinar los sistemas y el tráfico de red para elaborar una referencia en cuanto a la configuración y la actividad.

Uso de plantillas de seguridad para documentar y comparar

Puede utilizar la herramienta Configuración y análisis de seguridad para ver y documentar las opciones cruciales de configuración de la seguridad. Puede utilizar la misma herramienta para comparar la seguridad actual con respecto a una plantilla predefinida.

Examen y documentación de los derechos de usuarios y grupos

Examine y documente las cuentas de usuario, grupos, recursos compartidos y permisos. Puede usar el complemento Usuarios y grupos locales de Administración del equipo o la utilidad *NT User Manager* para examinar las cuentas de grupo y de usuario. Asimismo, puede usar los comandos *NET USER, NET GROUP* y *NET LOCALGROUP* para hacer una lista de los usuarios y los grupos.

Examen de los programas y procesos activos

Utilizar la utilidad de configuración del sistema (MSCONFIG.EXE) para ver y documentar los programas de inicio automático en *Windows 98, Windows Millennium Edition* y *Windows XP*. Use el Administrador de tareas de *Windows NT*, *Windows 2000 y Windows XP* para ver las aplicaciones y procesos que se estén ejecutando en ese momento.

Netstat y los puertos TCP

Documente los puertos TCP activos pertenecientes a los sistemas monitorizados. Use el comando NETSTAT –a para mostrar los puertos TCP activos. En los sistemas *Windows XP* y *2003*, puede usar NETSTAT –aon para mostrar los procesos implicados con los puertos activos. Haga clic aquí para ver una lista de los números de puerto TCP comunes de Microsoft o aquí para tener acceso a una discusión general de las asignaciones de números de puerto TCP.

Supervisión de red

Desde los primeros tiempos de *Windows* para Trabajo en Grupo 3.11, siempre ha estado disponible alguna forma del Monitor de red de Microsoft. Monitor de red es una herramienta de diagnóstico que captura los paquetes de red para examinarlos, realizar su seguimiento y elaborar estadísticas. Monitor de red, u otra herramienta similar, es la principal utilidad para establecer referencias del tráfico de una red.

Supervisión del rendimiento

Windows ha dispuesto de una utilidad de supervisión del rendimiento para monitorizar y realizar un seguimiento en tiempo real de diversos sucesos. La supervisión del rendimiento se puede utilizar para hacer un seguimiento de los objetos, procesos y procesadores con el fin de establecer umbrales de referencia.

Debe conocer y documentar las averiguaciones que realice para establecer métodos de detección de intrusiones y detectar actividades no autorizadas cuando se produzcan en el futuro.

☑ Usar y examinar los registros de sucesos

Los registros de sucesos son la herramienta principal de supervisión para realizar el seguimiento de la actividad de intrusión en un sistema en particular o contra él. Los administradores de red deben examinarlos en busca de sucesos inesperados e investigar cualquier actividad sospechosa. Los registros de sucesos se pueden examinar de forma manual o mediante una herramienta de cotejo de registros de sucesos.

Registros de sucesos

Los sistemas *Windows NT*, *Windows 2000*, *Windows XP* y *Windows 2003* comparten tres registros de sucesos comunes: de aplicación, de seguridad y del sistema. Tanto los administradores como los usuarios pueden tener acceso a la utilidad Visor de sucesos, en el menú Herramientas administrativas*; Windows 2000, Windows XP y Windows Server 2003* tienen un complemento Visor de sucesos predeterminado para *Microsoft Management Console*. Otros registros, como los del Servidor *DNS*, el Servicio de replicación de archivos y el Servicio de directorio, pueden estar disponibles en función de la plataforma de *Windows* y de cuándo se haya instalado. Cualquier registro puede proporcionar una evidencia de una intrusión en la seguridad.

Automatizar la visualización de sucesos en varios equipos

Si tiene que supervisarse más de un sistema, ir a cada equipo personalmente puede ser una tarea difícil desde un punto de vista administrativo. La utilidad Visor de sucesos le permitirá abrir

los registros de sucesos de equipos remotos, si se disponen de derechos administrativos. *Microsoft* proporciona otras dos herramientas para ver y administrar varios equipos a la vez.

EventCombMT

EventCombMT es una utilidad gratuita de Microsoft que se utiliza para buscar en registros de sucesos remotos a través de dominios diferentes y funciona con los sistemas *Windows NT, Windows 2000, Windows XP* y *Windows Server 2003*.

Microsoft Operations Manager

El uso de una consola de *Microsoft Operations Manager (MOM)* basada en el *Web* ofrece un completo conjunto de características destinadas a ayudar a los administradores a supervisar y administrar los sucesos y el rendimiento de servidores basados en Windows. Permite a los administradores de red centralizar las actividades de administración de auditorías y registros de sucesos. Se pueden crear reglas y directivas predeterminadas para administrar servidores y responder a los sucesos de los servidores que intenten infringirlas.

☑ **Activar la auditoría**

Aunque no se activa de forma predeterminada en ninguna plataforma *Windows*, la auditoría se puede usar para supervisar los intentos fallidos y correctos de acceso a los objetos, como son los inicios de sesión, el acceso a archivos y directorios, los cambios de contraseña, el uso de las aplicaciones, los cambios en los derechos de seguridad, en el Registro o en las directivas, y los intentos de cerrar el sistema. Los sucesos de auditoría se graban en el registro de seguridad.

La auditoría es una herramienta eficaz, pero debe tener cuidado y no supervisarlo todo o se capturará tanta información que dejará de tener utilidad. La mayor parte de los administradores encuentran un buen equilibrio entre el costo y el beneficio que supone supervisar los sucesos fallidos, como puede ser el intento fallido de aumentar los derechos de usuario, y supervisar las actividades fundamentales, como el reinicio del sistema.

La detección de intrusiones se puede llevar a cabo mediante la supervisión de los sistemas en busca de actividad inesperada que se desvíe de las referencias establecidas. Toda actividad sospechosa debe investigarse.

5.2.3 INSPECCIÓN DE CÓDIGO

Una inspección es un auténtico recorrido humano a través del código, buscando problemas. Normalmente, las inspecciones se realizan por motivos de seguridad, pero éste no tiene por qué ser siempre el caso.

La inspección puede ser parte del proceso de comprobación, pero se recomienda que se consideren las inspecciones de seguridad como una parte completamente separada del ciclo de desarrollo.

Debe ser otra persona distinta del original la que debe realizar una inspección de seguridad.

En ***el proceso de inspección*** la inspección de seguridad de las aplicaciones debe ser muy completa y debe intentar cubrir todo el material. Un proceso de inspección puede ayudar a un

inspector a examinar y verificar el código sistemáticamente. El proceso de inspeccionar un código es el siguiente:

1. Hacer un mapa del programa.

Para crear un mapa del programa, es necesario entender lo que sucede en cada parte de él. Tener un programa modular hace que esto sea significativamente más fácil, ya que puede permitir hacer un mapa de cada módulo y seguir la pista de las llamadas de uno a otro fácilmente.

Este paso se llama algunas veces *análisis del flujo de datos*, y es probable que sea la mejor forma de comprender lo que está haciendo cada programa exactamente. Es posible comenzar por el módulo de carga (a menudo main o similar), y se construirá un árbol desde ahí. Dibujar diagramas y crear un cuadro como un mapa resulta de gran ayuda.

Cada vez que se realiza una llamada a una función dentro del código, se realiza un seguimiento hasta el lugar desde donde se la llama. Si es a una biblioteca externa, debe asegurase que comprendemos lo que hace esa llamada. Si es a una parte interna del programa, continuaremos haciendo un mapa de esa parte. Si observamos que existen lazos, se realizan referencias a las partes del mapa que ya se hallan trazado.

Esta técnica intenta proporcionar una buena idea de lo que está realmente sucediendo dentro del programa cuando se inicia. Comprobara lo que sucede a todas las opciones de línea de comandos y se buscará todos los puntos por los que entran datos externos en el programa.

Cuando se realiza un mapa del programa, se comprende muy bien cómo funciona todo él. Debemos tomarnos el tiempo y no saltar partes del programa. Si cuando se concluya, comprueba que ha seguido todas las llamadas y encontrado partes a las que no llegamos, se colocará una nota para que el programador las elimine, ya que son partes de código extrañas, a las que nunca se llama. Puede que sean restos de una versión anterior, o añadidas para depuración. De todos modos, añaden complejidad al programa y éste puede trabajar bien sin ellas.

2. Definir todas las estructuras que están en uso.

Saber qué tipos de estructuras de datos están disponibles en un programa puede aumentar en gran medida la comprensión de lo que se está haciendo. Las estructuras de datos pueden ser internas o tomadas de fuentes externas. Probablemente se encuentren la mayoría de ellas mientras se realiza el mapa, de modo que ninguna estructura real pueda sorprender.

Sirve de ayuda imprimir las partes de archivos incluidos que establecen valores estáticos y definen estructuras. Puede que se quiera incluir también cualquier prototipo de función como parte de una referencia.

Después de reunir las estructuras, se buscará cosas como redundancias en ellas. Algunas veces, dos estructuras similares realizan casi exactamente lo mismo. Esto ayuda a reducir la complejidad del programa. Ahora que ya se tiene un mapa de las estructuras y del programa en su conjunto, debe comprenderse básicamente lo que sucede dentro del programa.

3. Comprobar y verificar todos los puntos de entrada.

Se retrocede y se busca todos los lugares por los que se introducen datos en el programa. Éstos pueden proceder de cualquier fuente, archivo, red, memoria compartida, o línea de comandos.

Es necesario asegurarse que hay filtros y se comprobarán todos los puntos donde haya entrada en el programa. De ser necesario deben considerarse varios escenarios, como los parámetros extra largos, datos formateados con malformaciones y bits extraños o aleatorios.

Hay que asegurarse de que no puede entrar en el programa procedente del exterior ningún dato sin que sea comprobado y verificado como correcto. Esto es: No debe entrar en nuestro programa ningún dato sin ser comprobado.

También verifique que los filtros no son demasiado estrictos como para no permitir el uso del programa. Esto es importante para mantener la funcionalidad.

4. Comprobar que no haya inundaciones.

Revise el código en busca de cualquier lugar donde se coloque un gran buffer dentro de uno más pequeño. En los *buffers* dinámicos como los estáticos, ambos pueden causar problemas. Esto puede o no ser aplicable al programa.

Simplemente se elegirá un punto donde se defina el buffer y se dará seguimiento a través del programa hasta que éste termine o se libere el buffer.

5. Comprobar las posibles condiciones de adelanto.

La comprobación final consiste en buscar lugares en el código donde pueda ocurrir una condición de adelanto. Esto sucede normalmente cuando se utilizan archivos, por lo que es necesario comprobar éstos en primer lugar.

Una condición de adelanto puede darse si hay un lapso de tiempo entre que se abre un archivo y el momento en el que se utiliza realmente. Si existen puntos como estos, debe asegurase de arreglarlos.

5.2.3.1 VALIDACIÓN DE CÓDIGO

La entrada de datos ha evolucionado junto con los métodos que permitían dichas entradas. En la actualidad, puede utilizarse una gran variedad de medios para introducir datos en la PC, incluyendo teclado, ratón, lápices, voz, escáneres de código de barras, etc., incluso la presión del pedal del freno, como entrada para el sistema de frenado ABS de los automóviles.

Es posible utilizar un sofisticado depurador[14] contenido en Microsoft Visual Basic .NET que ayuda a minimizar los errores cometidos en la entrada de datos y prevenir que la aplicación falle o que se comporte de forma errática. Aunque ha habido un rápido avance en el desarrollo de

[14] Es necesario explicar que en este capítulo se muestran los diferentes tipos de entradas y validaciones que se deben considerar al momento de desarrollar y programar aplicaciones en VB.NET y en el capítulo 7 se complementará esta sección exponiendo las herramientas que completan la seguridad respecto a la validación de la entrada.

herramientas de depuración utilizadas para detectar el código que no sea capaz de enfrentarse correctamente con las entradas erróneas, la necesidad de validar la entrada (mediante el desarrollo de código que compruebe la entrada proveniente de todas las posibles fuentes y que impida la entrada de datos que puedan provocar daños) ha crecido exponencialmente.

En los entornos informáticos interconectados de la actualidad no sólo se necesita verificar que la entrada es la correcta, sino que también debe verificar que la entrada (recibida de distintas fuentes) no es dañina. En particular, se necesita proteger su aplicación de todos los ataques relacionados con la entrada, tales como:

• Denegación de servicio (DoS), inyección SQL y ataques de programación de sitio cruzado, por mencionar algunos.

• Ataques de descubrimiento de información que implican la revelación de información sensible ya sea relacionada con el propio sistema o con los datos que se intentan proteger. Por ejemplo, el usuario puede pasar una entrada que produzca un mensaje de error que proporcione detalles del sistema, tales como nombres de archivos, la estructura de directorios o la estructura de la base de datos.

El primer paso que se debe dar para garantizar que las entradas no pueden provocar malas consecuencias es identificar todas las fuentes de entrada de datos que puede utilizar la aplicación. Los tipos de entrada van desde la obvia entrada realizada por el usuario a determinadas formas de entrada tales como la información contenida en las cabeceras *HTTP* o las opciones de configuración de las aplicaciones almacenadas en el disco. *Identificar* las distintas fuentes de entrada resulta esencial para poder analizar las posibles amenazas de seguridad puestas al descubierto por cada forma de entrada. Identificar todas las fuentes de entrada también ayuda a probar si la aplicación es capaz de defenderse frente a todos los posibles ataques que haya identificado.

Entradas directas del usuario

La entrada directa realizada por el usuario es el tipo de entrada con el que se encuentran más familiarizados diseñadores de aplicaciones en *Visual Basic*. Desde su concepción, *Visual Basic* ha sido diseñado para permitir que los diseñadores creen formularios con rapidez y facilidad. Estos formularios solicitan, con frecuencia, entradas directas a los usuarios utilizando controles, como cuadros de texto, cuadros de lista, casillas de verificación y botones de opción.

Las entradas en formato libre permitidas por ciertos controles (por ejemplo, cuadros de texto, cuadros de Formato de texto enriquecido, o RTF, y controles de cuadrícula) son el tipo de entrada que puede suponer el mayor peligro potencial, si dicha entrada no se verifica debidamente. Por tanto, deberá centrar los esfuerzos en la validación de todas las entradas realizadas en este tipo de controles.

Visual Basic .NET proporciona una serie de herramientas (controles, clases, propiedades, métodos y sucesos) que ayudan a validar los datos. Las herramientas disponibles varían dependiendo si está creando un *Windows Forms* o una aplicación *Web ASP.NET*. Estas herramientas ofrecen una primera línea defensiva contra las entradas no válidas de los usuarios.

Sin embargo, el nivel de protección ofrecido por estas herramientas depende de la forma en que se utilice la entrada. Por ejemplo, si la entrada no se va a exportar fuera de la aplicación, verificar la entrada proveniente de la fuente, como primera línea defensiva, puede resultar suficiente (éste es el caso en el que la primera línea de defensa sirve también como última línea de defensa). Si la entrada se almacena y se utiliza posteriormente o si se va a salir fuera de la aplicación, estas herramientas puede que proporcionen poca o ninguna protección. Debe diseñarse siempre las aplicaciones para que dispongan de un mecanismo que proporcione una última línea de defensa.

Entradas no realizadas por el usuario

Las entradas no realizadas por el usuario incluyen cualquier entrada a la aplicación que ésta no solicitada directamente (Por ejemplo, leer un archivo de disco) o que recibe pasivamente (como recibir datos desde un puerto de comunicaciones). Los datos almacenados en archivos o en una base de datos tienen que haber sido introducidos en un momento dado por un usuario y validados en ese mismo momento.

Los datos representan un riesgo para la aplicación. Por este motivo, debe identificarse todas las fuentes de entrada en la aplicación distintas a las de los usuarios y analizar el riesgo asociado con cada una de las fuentes. Entre los ejemplos de entradas no realizadas por los usuarios pueden citase:

- Datos leídos desde el disco.
- Datos obtenidos de una base de datos.
- Datos pasados a través de un puerto o un *socket* de comunicaciones.
- Solicitudes recibidas por su aplicación Web.

Como ejemplo sencillo, suponga que crea una aplicación para un hospital que lee información de los pacientes desde un archivo de texto. Podrá utilizas el código mostrado a continuación pasa leer la edad del paciente como si fuera un valor numérico:

```
Dim hFile As Integer
Dim EdadPaciente As Integer
hFile FreeFile()
FileOpen(hFile, "RegistroPaciente.txt", OpenMode.Input)
EdadPaciente = Integer.Parse(Linelnput(hFile))
FileClose(hFile)
```

Código fuente 5.A.

La información contenida en *RegistroPaciente.txt* representa la entrada en su aplicación. Como tal, deberá validar los datos que lea al igual que validaría cualquier otro dato obtenido de otra fuente de entrada. Por ejemplo, la siguiente instrucción puede producir diversos errores:

```
EdadPaciente = Integer.Parse(Linelnput(hFile))
```

Esta línea de código provocaría una excepción no controlada por cualquiera de los siguientes motivos:

• La línea está vacía.

• La línea contiene caracteres no numéricos, por ejemplo, letras de la A a la Z.

• El número leído excede el valor mínimo o el máximo correspondiente a un entero. Por ejemplo, se producirá una excepción por desbordamiento si la línea contiene el valor 123456789123456789, que es superior al valor máximo permitido para un entero 2.147.483.647.

Además de los problemas potenciales representados por esta línea de código, que extrae los datos desde un archivo, el código realiza una serie de hipótesis, entre las que se incluyen:

• Que existe el archivo *RegistroPaciente.txt*.

• Que la edad del paciente se ha representado en forma numérica.

• Que si la edad del paciente es un número, se trata de un número realista.

• Que si la edad del paciente es un número realista, es la edad actual del paciente.

Si el archivo RegistroPaciente.txt, o su contenido, fuera comprometido por un atacante, el código mostrado previamente fallaría de diversas formas. Por ejemplo, si se modificara el nombre del archivo o si el archivo fuera eliminado o desplazado, la llamada *FileOpen* fallaría provocándose una excepción no controlada que mostraría el nombre del archivo que necesita la aplicación (lo que, posiblemente, constituya información de utilidad para cualquier atacante que quisiera desarrollar otras formas de ataque. Si se modificara el contenido del archivo de alguna forma (por ejemplo, añadiendo nuevas líneas o eliminando otras) o se cambiara la edad del paciente por cualquier otra cadena, fallaría la llamada a *Integer.Parse(LineInput(hFile))* provocándose una excepción no controlada.

Finalmente, un cambio particularmente dañino que podría efectuar un atacante en el archivo RegistroPaciente.txt sería modificar la edad del paciente por un valor distinto. Esta modificación no produciría ningún error pero podría provocar efectos de difícil evaluación en la forma en que se fuera a tratar al paciente. Por ejemplo, si se modificara la edad del paciente de 22 a 85 años, la compañía de seguros del paciente podría denegar automáticamente la cobertura médica, ya que el riesgo asociado con un paciente de 22 años es muy bajo, pero muy alto si se trata de una persona de 85 años.

Para evitar la mayoría de estos problemas, podría agregarse un mecanismo de control de errores a su aplicación. Por ejemplo, como mínimo, debería incluir una instrucción *Try. . . Catch* que englobara a todo el código con el fin de capturar y recuperarse de cualquier excepción, lo que ayudaría a impedir que la aplicación fallara de forma inesperada. Sin embargo, necesita poner mucho cuidado a la hora de informar a los usuarios de los errores producidos.

Para impedir modificaciones no autorizadas en sus datos (por ejemplo, el cambio no autorizado de edad de 22 a 85 años) pode agregarse comprobaciones más profundas para validar la entrada o agregar a su aplicación un mecanismo de información de errores que registre todos los cambios de datos no habituales. Por ejemplo, si el valor leído para la edad no se corresponde con la edad calculada para el paciente en función de la fecha de nacimiento del mismo, o el valor de la edad es significativamente distinto a la edad que se encuentra registrada en la base de datos de su aplicación, la aplicación podría registrar un error advirtiendo que ese dato es sospechoso.

5.2.3.2ENTRADAS, SUBRUTINAS

Cualquier subrutina que pueda ser llamada directa o indirectamente por una aplicación (o componente) externa puede servir como punto de acceso para un atacante. Por ejemplo, en el caso de una aplicación cliente/servidor en la que una persona introduzca los datos en una aplicación cliente y, posteriormente, se pasen los datos a una aplicación servidor, la validación de los datos no puede encontrarse únicamente en la aplicación cliente por los siguientes motivos:

- Los datos nunca se validarán suficientemente en la aplicación cliente.
- Los datos pueden ser comprometidos en su camino hacia el servidor.
- Se puede utilizar otra aplicación para pasar datos no válidos a la aplicación servidora.

Cualquier subrutina que vaya a ejecutar alguna acción de acuerdo con un parámetro que se le haya pasado, debe validarse siempre el parámetro de entrada antes de llevar a cabo dicha acción. Si no tiene éxito a la hora de incluir nuevas verificaciones que comprueben las entradas a la subrutina, un atacante o un usuario mal informado pueden pasar valores que impedirían a la aplicación desarrollar sus funciones con normalidad.

5.2.4 PROTECCIÓN DEL CÓDIGO

En el caso en que una aplicación se ejecute directamente desde un entorno que no sea de confianza (tal como Internet) el sistema de seguridad de acceso al código impide que el código dañino se ejecute al verificar primero si el código tiene permiso para realizar una determinada operación tal como eliminar archivos (este hecho se conoce oficialmente como demanda de un permiso determinado). Por ejemplo, cuando el componente gráfico (contenido en una página Web) ejecute una instrucción *Kill* de *Visual Basic .NET* para eliminar un archivo, la instrucción *Kill* solicita (demanda) el permiso para eliminar archivos, y si no se le concede este permiso, se iniciará una excepción de seguridad y fallará la operación.

Como las aplicaciones basadas en Visual Basic .NET suelen estar formadas por funciones y métodos del propio lenguaje, tal como *File Open*, *Kill*, *Shell* y *Show*, todas estas funciones se verifican internamente para ver si su código cuenta con los permisos necesarios para realizar la acción requerida. Si el sistema de seguridad de acceso al código de .NET no ha asignado los permisos necesarios, se iniciará una excepción y la acción no tendrá lugar. No es necesario hacer nada para activar la seguridad de acceso al código en sus aplicaciones basadas en Visual Basic .NET. Esta función se encuentra activada de forma predeterminada.

Funciones de Seguridad

Si se desarrollan aplicaciones o componentes para ejecutarlos en entornos tales como una red local (intranet) o Internet, puede que la aplicación no se ejecute tal y como se esperaba en estos entornos. Por ejemplo, ciertas instrucciones de Visual Basic .NET pueden provocar excepciones de seguridad que detengan de forma inmediata la ejecución de su aplicación.

Si se desea crear componentes del sistema del tipo de las bibliotecas del sistema .NET (tal como los componentes *Microsoft. VisualBasic, System.Web* y *System.Windows.Forms*, que se encuentran registrados globalmente en el sistema), tendrían que aplicarse técnicas de seguridad de

acceso al código más avanzadas para conseguir que el componente global del sistema sea accesible y seguro para todos los llamadores que no sean de confianza. La aplicación del atributo *AllowPartialiyTrustedCallers*, que suministra evidencias personalizadas que se conocen cuando se utilizan adecuadamente *Demand*, *LinkDemand* y *Assert*. Si se decide aplicar el atributo *AllowPartiallyTrustedCallers* estará indicando al sistema de seguridad de acceso al código que uno mismo será el responsable de proteger todo el código contenido en la aplicación. Antes de aceptar esta responsabilidad adicional, necesita reconsiderar si necesita utilizar un componente del sistema. En la mayoría de los casos, resulta más sencillo crear un componente que no sea del sistema (el tipo de componentes de biblioteca de clases que Visual Basic .NET crea de forma predeterminada) y distribuirlo en cada aplicación, algo que el Asistente para implementación de Visual Basic .NET puede hacer automáticamente.

Seguridad de acceso al código

La seguridad de acceso al código es el sistema de *.NET* que controla el acceso a los recursos a través de la ejecución del código. Esta característica de seguridad es independiente de la seguridad que proporciona el sistema operativo, a la que complementa. Cuando el usuario ejecute la aplicación, *Common Language Runtime* de *.NET* la asignará a una zona. Las cinco zonas son:

- ☑ Mi PC: aplicaciones en el equipo del usuario.
- ☑ Intranet local: aplicaciones en la intranet del usuario.
- ☑ Internet: aplicaciones procedentes de Internet.
- ☑ Sitios de confianza: aplicaciones procedentes de sitios definidos como "De confianza" en Internet Explorer.
- ☑ Sitios que no son de confianza: aplicaciones procedentes de sitios definidos como "Restringidos" en Internet Explorer.

El administrador del sistema asigna permisos de acceso específicos a cada una de estas zonas. Se pueden establecer cuatro niveles diferentes de seguridad para una zona: nivel de plena confianza, media, baja o sin confianza. Estos niveles determinan los recursos a los que podrá obtener acceso una aplicación. La zona, junto con otras pruebas de seguridad, como el editor, el nombre seguro, el sitio Web y la URL del código, determinan los permisos que se conceden al código en tiempo de ejecución. No se puede controlar la configuración de seguridad del equipo del usuario, así que la aplicación debe trabajar con los valores que encuentra mientras se ejecuta. En consecuencia, puede que se le niegue el acceso a unos recursos determinados. Por ejemplo, aunque la aplicación necesite escribir datos en un archivo, puede que el sistema del usuario origine una excepción para denegarle el acceso de escritura en tiempo de ejecución.

Desarrollar una aplicación capaz de enfrentarse a este tipo de situaciones no significa necesariamente que la aplicación tenga que descubrir otro modo de escribir los datos, sino que deberá anticipar que se le puede prohibir la escritura de datos y tener capacidad de respuesta ante tal posibilidad. Si se desea conseguir un código más sólido, puede utilizar otras formas de control de excepciones (*Try…Catch* en *Visual Basic)*, o bien algunos de los objetos del espacio de nombres *System.Security.Permissions*.

Los niveles de seguridad de las zonas se establecen con las herramientas de administración que se agregan durante la instalación de *.NET Framework*. Para obtener más información sobre la configuración de los niveles de seguridad para las zonas de un equipo.

Plena confianza

Generalmente, los programadores trabajan en un entorno de plena confianza. Guardan el código fuente en el disco duro y prueban las aplicaciones en sus equipos de desarrollo. En este entorno de plena confianza, todo el código que compile el programador se podrá ejecutar en el equipo local. No se producirán excepciones ya que, de forma predeterminada, el equipo local se ha definido para ser un entorno de plena confianza.

Confianza parcial

Una zona es de confianza parcial cuando no se le ha asignado plena confianza. La aplicación, cuando se implementa, puede pasar a una zona nueva en la que quizás no se le otorgue plena confianza. Los escenarios más comunes en los que se ejecuta el código con confianza parcial son los dos siguientes:

☑ Código en ejecución que se descarga de Internet.

☑ Código en ejecución que reside en un recurso compartido de red (intranet).

Estos son algunos de los recursos que se pueden denegar en una zona de confianza parcial:

☑ E/S de archivos, incluida la lectura, escritura, creación, eliminación o impresión de archivos.

☑ Componentes del sistema, como valores de registro y variables de entorno.

☑ Componentes del servidor, incluidos los servicios de directorio, el registro, los registros de eventos, los contadores de rendimiento y las colas de mensajes.

Nivel de la zona

Todas las clases de *.NET Framework* y cada uno de sus métodos cuentan con un atributo de seguridad que define el nivel de confianza necesario para ejecutar dichos métodos. En ocasiones, puede que el atributo no sea accesible en tiempo de ejecución debido a estas características de seguridad. El nivel de la zona no es una simple asignación de un determinado nivel de confianza a los atributos, sino que se trata de un conjunto de permisos específicos que se otorgan a determinadas clases y métodos. La aplicación no podrá consultar simplemente el nivel de confianza y predecir los recursos que no se encuentran disponibles. Se puede determinar si la aplicación se ejecuta o no en un nivel de plena confianza.

Desarrollo para entornos de confianza parcial

No existe una única solución para el desarrollo para entornos de confianza parcial. La solución variará en función de la aplicación que se escriba. Asimismo, no basta con probar la aplicación para un nivel de confianza concreto y seguir adelante con el proceso, ya que se debe tener en cuenta que el nivel de confianza puede cambiar durante la ejecución.

El primer paso en el desarrollo para zonas de confianza parcial consiste en escribir código que reconozca que pueden surgir excepciones de seguridad. Observe el siguiente código:

```
Public Sub MakeABitmap()
    Dim b As Bitmap = New Bitmap(100, 100)
    ' Agregar código para dibujar una bonita imagen en el mapa de bits
    b.Save("c:\PrettyPicture.bmp")
End Sub
```

Código fuente 5.B.

Este método se ejecutará sin originar ninguna excepción si el proyecto y el ensamblado correspondiente se encuentran almacenados en el disco duro de su equipo, y uno es miembro del grupo Administradores del mismo. Sin embargo, si implementa esta aplicación en su intranet, se originará una excepción *System.Security.SecurityException* cada vez que la aplicación intente guardar el objeto de mapa de bits. Si no introdujo en el código un bloque del tipo *Try…Catch*, la excepción finalizará la ejecución de la aplicación. Seguramente, esta experiencia de usuario no resultará satisfactoria. Por el contrario, si agrega código para el control de excepciones, la aplicación podrá avisar al usuario de que la aplicación no puede finalizar todas las tareas que necesita. Así como limpiar los objetos existentes, para que no surjan problemas con el código que se ejecute tras el bloque *catch*. El código agregado comunicará al usuario que el archivo no se guardó debido a una denegación de seguridad, al tiempo que separará los errores de seguridad de otros errores de E/S de archivo, como los producidos por nombres de archivo incorrectos. Este método no provoca ninguna brecha en la seguridad, ya que el usuario podrá modificar la configuración para otorgar confianza a la aplicación o evitar que ésta se ejecute.

```
Public Sub MakeABitmap()
    Dim b As Bitmap
    Try
        b = New Bitmap(100, 100)
        b.Save("c:\PrettyPicture.bmp")
    Catch ex As System.Security.SecurityException
        ' Comunicar al usuario el error al guardar.
        MessageBox.Show("Se denegó el permiso para guardar el archivo " & _
           "y no se guardó el mapa de bits.")
    Catch ex As System.Exception
        ' Reaccionar a otras excepciones aquí.
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

Código fuente 5.C.

Las clases, atributos y enumeraciones del espacio de nombres *System.Security.Permissions* permiten controlar aún más las tareas de seguridad de la aplicación. Si se escriben bibliotecas que pueden recibir llamadas de otras aplicaciones, seguramente desee que se comprueben los permisos del código que llama. Puede, por ejemplo, agregarse al principio del archivo de código el siguiente atributo a nivel de ensamblado. El tiempo de ejecución comprobará el permiso cuando se cargue el ensamblado. Si el tiempo de ejecución deniega el permiso solicitado, no se cargará el ensamblado y se originará una excepción de seguridad. Si se agrega este atributo a una aplicación independiente, puede que ésta no se ejecute. Si este atributo aparece en una biblioteca

de clases, puede que ésta no se cargue en tiempo de ejecución. Deberá agregar al código un bloque *try/catch* que llame a la biblioteca de clases.

```
<Assembly: System.Security.Permissions.FileIOPermissionAttribute( _
SecurityAction.RequestMinimum, Write:="c:\PrettyPicture.bmp")>
```

Asimismo, puede solicitarse de forma específica los permisos del tiempo de ejecución con el método *Demand*, como se muestra a continuación.

El tiempo de ejecución podrá aceptar o denegar dicha solicitud. Si se deniega la solicitud, se originará una excepción de seguridad. Deberá modificar el código del modo indicado para solicitar explícitamente el permiso necesario para escribir el archivo de mapa de bits:

```
Public Sub MakeABitmap()
    Dim filename As String = "c:\PrettyPicture.bmp"
    Dim permission As FileIOPermission = _
        New FileIOPermission(FileIOPermissionAccess.Write, _
        filename)
    Dim b As Bitmap = Nothing
    Try
        b = New Bitmap(100, 100)
        permission.Demand()
        b.Save(filename)
    Catch ex As System.Security.SecurityException
        ' Comunicar al usuario el error al guardar.
        MessageBox.Show("Se denegó el permiso para guardar el archivo " & _
           "y no se guardó el mapa de bits.")
    Catch ex As System.Exception
        ' Reaccionar a otras excepciones aquí.
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

Código fuente 5.D.

Comprobación

El segundo paso en el desarrollo para zonas de confianza parcial es la comprobación de la aplicación en varios entornos, especialmente intranet e Internet. De este modo, se provoca el inicio de excepciones de seguridad.

De forma predeterminada, las aplicaciones Web se ejecutan anónimamente, es decir, sin disponer de información sobre la identidad del usuario. Este método resulta útil en sitios que contienen información pública. Para controlar quién obtiene acceso a una aplicación o a otros recursos, se debe agregar la característica de autenticación. La autenticación es el proceso de identificación del usuario de la aplicación y de comprobación de que dispone de acceso autorizado a la misma. Existen varios métodos de autenticación compatibles con ASP.NET, aunque los más utilizados son los siguientes:

Anónima: No se solicita al usuario información sobre su identidad. Este método resulta apropiado para sitios Web con contenido público. Se pueden utilizar *cookies* cuando sea necesaria

la representación del sitio. Para obtener más información sobre el uso de *cookies* en aplicaciones de *ASP.NET*.

Formularios: La aplicación muestra al usuario un formulario de inicio de sesión en el que se solicita información para el inicio de sesión como, por ejemplo, el nombre y una contraseña. El formulario se devuelve al servidor, donde se compara la información con los datos almacenados.

Básica: Este tipo de autenticación se configura con los servicios de *Internet Information Server (IIS)* y es compatible con la mayoría de exploradores. Cuando se habilita este método, el explorador pide al usuario que introduzca su nombre y su contraseña y, a continuación, devuelve la información a la aplicación de *ASP.NET* utilizando el algoritmo de codificación *Base64*, que se puede descifrar fácilmente. Este tipo de autenticación exige que los usuarios dispongan de cuentas de *Windows*. Si se utiliza junto con *SSL* (*Secure Sockets Layer*), el método básico resulta más seguro.

Implícita: Este tipo de autenticación se configura utilizando los servicios de IIS y se encuentra disponible en servidores en los que se ejecuta *Microsoft Windows 2000* o *Windows XP*. La autenticación implícita ofrece un nivel de cifrado de contraseña mayor que el método básico. Este método exige que los usuarios tengan cuentas de Windows almacenadas en *Microsoft Active Directory*.

Integrada de Windows: Este tipo de autenticación es similar a la básica y la implícita, a excepción de que en este método la contraseña y el nombre de usuario no se devuelven a la aplicación *Web*. Esta autenticación resulta especialmente aconsejable en entornos de intranet. Exige que los usuarios dispongan de cuentas de *Windows* y sólo es compatible con el explorador *Internet Explorer*.

Certificados: Un certificado es una clave digital que se instala en un equipo. Cada vez que un usuario trata de obtener acceso al servidor, se solicita la clave. El servidor, a continuación, autentica el certificado en un dominio o en *Active Directory*. Este método resulta apropiado para aplicaciones que exigen un nivel de seguridad tan elevado que se compensa el coste que supone la administración de certificados.

Passport: Microsoft ofrece este servicio de autenticación centralizado. La autenticación de *Passport* resulta apropiada si se utiliza el sitio *Web* con otros sitios de *Passport*, ya que permite el acceso a todos ellos con un único inicio de sesión, o si no se desea mantener una base de datos de usuarios.

La autenticación permite autorizar al usuario de la aplicación, pero esto no significa necesariamente que ya pueda obtener acceso a los recursos, como los archivos y las bases de datos. Los recursos se pueden configurar de forma que se encuentren disponibles para usuarios particulares y no para la aplicación Web. En tal caso, se puede utilizar la representación para facilitar a los usuarios el acceso a los recursos. El proceso del servidor se ejecutará con la identidad del usuario autenticado. Cuando la aplicación utiliza la representación y consulta una base de datos, la aplicación de la base de datos procesa la consulta como si procediera del usuario, no del servidor. Para habilitar la representación, se debe configurar el atributo

"*impersonate*" del elemento de identidad del archivo *Web.config* de la aplicación, como se muestra en el ejemplo. Los archivos *Web.config* se crean con cada proyecto de aplicación Web.

```
<identity impersonate="true">
```

Un paso por delante de la representación se encuentra la delegación, que utiliza la identidad del usuario para obtener acceso a recursos remotos (de otros equipos). No todos los métodos de autenticación admiten la delegación, como se indica en la siguiente tabla.

ADMITE LA DELEGACIÓN	NO ADMITE LA DELEGACIÓN
Básica	Anónima
Integrada de Windows	Implícita
Certificados	Passport
	Formularios

Tabla 5.3. Delegación

5.2.4.1. RECOMENDACIONES PARA LA PROTECCIÓN DEL CÓDIGO

Tener cuidado con el código cifrado

El error más común es utilizar código de cifrado creado por uno mismo, que suele ser el más frágil y fácil de vulnerar. Nunca crear un propio código de cifrado, ya que le ocasionará problemas. No se piense que por el hecho de haber creado algoritmos de cifrado propios los intrusos no podrán vulnerarlo. Los intrusos tiene acceso a depuradores y disponen de tiempo y conocimientos necesarios para determinar exactamente el funcionamiento de estos sistemas (a veces pueden vulnerarlos en cuestión de horas). Por tanto, es recomendable utilizar *CryptoAPI* para aplicaciones Win32; el espacio de nombres *System.Security.Cryptography* posee abundantes algoritmos de cifrado sin fallos de seguridad y ampliamente probados.

Aplicar el principio del menor número de privilegios

El sistema operativo y *Common Language Runtime (CLR)* disponen de una directiva de seguridad por varios motivos. Muchos piensan que la razón de ser de una directiva de seguridad es impedir que los usuarios realicen acciones ilícitas de forma intencionada, como el acceso a archivos para los que no están autorizados, la reconfiguración de la red para que se adapte a sus necesidades y otros actos no permitidos. Aunque es cierto que los ataques internos son bastante frecuentes y es necesario protegerse frente a ellos, existe otra razón para mantener estrictamente esta directiva de seguridad. La directiva de seguridad sirve para proteger el código a fin de que las acciones realizadas por los usuarios, tanto voluntarias como involuntarias (casi tan frecuentes), no causen estragos en la red. Por ejemplo, un archivo de datos adjuntos descargado por correo electrónico y ejecutado en el equipo de un usuario, tiene limitado su acceso exclusivamente a los recursos a los que pueda tener acceso dicho usuario. Una buena directiva de seguridad limitará el daño que pueda realizar, por ejemplo, un archivo adjunto que contenga un troyano. Al diseñar, generar e implementar aplicaciones de servidor, no hay que asumir que todas las solicitudes procederán de usuarios bienintencionados. Si un intruso logra enviar una solicitud mal formada

que (esperemos que no) afecte al comportamiento del código, deseará tener todas las protecciones posibles en la aplicación para reducir el daño. El motivo por el que una empresa dispone de una directiva de seguridad no es solamente que no confíe en el programador o en su código. La directiva también se utiliza para protegerse del código bienintencionado que ha sido vulnerado por intrusos.

El principio del menor número de privilegios establece que todo privilegio debe concederse a la menor cantidad de código posible durante el mínimo tiempo necesario. Es decir, que en cualquier momento, el código esté protegido lo máximo posible. Cuando suceda algún imprevisto, se alegrará de haber aplicado las medidas de seguridad necesarias.

Si se va a utilizar *.NET Framework* para factorizar el código en ensamblados, tenga en cuenta el nivel de privilegios que necesita cada fragmento del código. Posiblemente resulte más fácil aislar aquel código que necesite un mayor nivel de privilegios en ensamblados independientes, a los que podrá conceder más permisos, permitiendo así que la mayor parte de los ensamblados se ejecuten con menos privilegios y, por tanto, aumente la seguridad del código. Un modo sencillo de restringir los privilegios en un ensamblado concreto es a través de solicitudes de permisos en dichos ensamblados. Si lo hace, recuerde que no sólo estará limitando los permisos del propio ensamblado, sino también los de aquellos a los que llame, debido a que se debe recorrer la pila de seguridad de acceso al código (CAS).

Muchos programadores generan sus aplicaciones de manera que los nuevos componentes se puedan utilizar una vez que el producto se haya probado y enviado. Resulta muy difícil asegurar este tipo de aplicaciones, ya que no hay forma de poder comprobar en toda la ruta del código si hay errores y agujeros de seguridad. Sin embargo, si la aplicación es administrada, CLR proporciona una característica que podrá utilizar para bloquear estos puntos de extensibilidad. Mediante la declaración de un objeto de permiso o de un conjunto de permisos y la llamada a las funciones *PermitOnly* o *Deny*, puede agregarse un marcador a la pila que limite los permisos concedidos a cualquier fragmento de código al que llame. Si se sigue este procedimiento antes de llamar a algún complemento, podrá restringirse las acciones de dicho complemento. Por ejemplo, un complemento creado para realizar cálculos de amortizaciones no necesita tener acceso al sistema de archivos. Éste es otro ejemplo del principio del menor número de privilegios, con el que se puede proteger de forma anticipada. Asegúrese de documentar estas restricciones y tenga en cuenta que los complementos con un nivel alto de privilegios pueden sortear estas restricciones con la instrucción *Assert*.

Prestar atención a los modos de fallo

Un fragmento de código puede fallar de tantas formas que sólo pensarlo resulta impresionante. La mayoría de los programadores, preferimos centrarnos en la ruta normal de ejecución. Ahí es donde se realiza el auténtico trabajo. El control de errores se suele hacer lo más rápido posible, sin preocuparse demasiado, para pasar a la siguiente línea de código de verdad.

Esta práctica no es segura. Es necesario prestar mucha más atención a los modos de fallo del código. No se suelen cuidar los detalles de estos fragmentos de código y, con frecuencia, ni siquiera se prueban.

El código que no se comprueba facilita la vulneración de la seguridad. Para paliar los efectos de este problema, tenga en cuenta las tres sugerencias siguientes. En primer lugar, prestar a esos pequeños controladores de error la misma atención que al código normal. Tener en cuenta el estado del sistema cuando se esté ejecutando el código para el control de los errores. En segundo lugar, una vez que escriba una función, ejecutar el depurador sobre ella varias veces y asegurarse de comprobar todos los controladores de error. Incluso esta técnica puede no detectar algunos errores sutiles de temporización. Quizá sea necesario pasar argumentos incorrectos a la función o ajustar el estado del sistema de alguna forma para que se ejecuten los controladores de error. Si se examina completamente el código, a la vez que lo revisa, se tendrá tiempo suficiente para comprobar el estado del sistema cuando se esté ejecutando. Se han descubierto varios errores en la lógica de programación mediante la revisión exhaustiva del código con un depurador. La eficacia de este método está probada. Finalmente, asegurar que los conjuntos de pruebas hacen que las funciones fallen. Intente realizar conjuntos de pruebas que examinen cada línea de código de la función. Estas pruebas pueden ayudar a detectar regresiones, especialmente si automatiza las pruebas y las ejecuta después de cada compilación.

5.2.5 ERRORES EN APLICACIONES DE VISUAL BASIC .NET

Difícil es, por no decir imposible, encontrar al programador que no tenga errores en su código. Por mucho cuidado que pongamos al codificar las aplicaciones, los errores de ejecución serán ese incómodo, pero inevitable compañero de viaje que seguirá a los programas allá donde estos vayan. En primer lugar, antes de abordar el tratamiento de errores en las aplicaciones, y los elementos que nos proporciona el entorno para manipularlos, podemos clasificar los tipos de errores en una serie de categorías genéricas.

Errores de escritura

Son los de localización más inmediata, ya que se producen por un error sintáctico al escribir el código, y gracias al *IDE* de *Visual Studio .NET*, podemos detectarlos rápidamente.

Cuando escribimos una sentencia incorrectamente, dejamos algún paréntesis sin cerrar, etc., el *IDE* subraya la parte de código errónea, y nos muestra un mensaje informativo del error al situar el cursor del ratón sobre el mismo. En el ejemplo de la Figura se ha declarado una estructura *While* que no se cerró con la correspondiente instrucción *End While*; por lo tanto, el IDE nos lo indica.

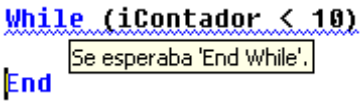


Figura 5.1 Error de escritura de código.

Errores de ejecución

Este tipo de errores son los que provocan un fallo en la ejecución del programa y su interrupción. No obstante, si se utilizan los gestores de error que proporciona la herramienta de desarrollo correspondiente, es posible en algunos casos, evitar la cancelación de la ejecución,

recuperando su control. El ejemplo del Código fuente 1 provoca un error, ya que se intenta asignar un valor que no corresponde al tipo de dato de una variable.

```
Dim dtFecha As Date
dtFecha = "prueba"
```

Código fuente 5.E.

Errores lógicos

Estos errores son los de más difícil captura, ya que el código se encuentra correctamente escrito, produciéndose el problema por un fallo de planteamiento en el código, motivo por el cual, por ejemplo, el control del programa no entra en un bucle porque una variable no ha tomado determinado valor; el flujo del programa sale antes de lo previsto de un procedimiento, al evaluar una expresión que se esperaba tuviera un resultado diferente, etc.

Errores y excepciones

Dentro del esquema de gestión de errores del entorno *.NET Framework*, encontramos las figuras del error y la excepción. Estos elementos son utilizados indistintamente en muchas ocasiones para hacer referencia genérica a los errores producidos; sin embargo, aunque complementarios, cada uno tiene su propia funcionalidad dentro del proceso de tratamiento de un error.

• **Error**. Un error es un evento que se produce durante el funcionamiento de un programa, provocando una interrupción en su flujo de ejecución. Al producirse esta situación, el error genera un objeto excepción.

• **Excepción**. Una excepción es un objeto generado por un error, que contiene información sobre las características del error que se ha producido.

Manipuladores de excepciones

Un manipulador de excepción es un bloque de código que proporciona una respuesta al error que se ha producido, y que se incluye en una estructura proporcionada por el lenguaje a tal efecto, es decir, para la captura de excepciones.

Tipos de tratamiento de error en VB.NET

VB.NET proporciona dos tipos de tratamiento de error: estructurado y no estructurado.

El primero se basa en los esquemas de captura de errores de lenguajes como *C#* y *C++*; gestionando los errores a través de excepciones, y una estructura de control que se encarga de atrapar aquellas excepciones que se produzcan.

El segundo es un sistema heredado de versiones anteriores de VB, y está basado en la detección y captura de errores a través de etiquetas de código, mediante saltos no estructurados en el flujo de la ejecución.

Este tema será tratado y explicado con mayor detenimiento en el capítulo 7 donde se mencionan métodos, técnicas y herramientas para disminuirlos o eliminarlos.

5.2.5.1 EXCEPCIONES EN VISUAL BASIC .NET

No importa lo bien organizado, preparado y equipado que se encuentre una persona, las cosas invariable, inevitable y normalmente acabarán yendo mal. Algunos escribirán contraseñas erróneas, intentarán abrir archivos con los formatos inadecuados y eliminarán inadvertidamente recursos esenciales de sus discos duros. Si después de que estos usuarios realicen este tipo de cosas su programa no funciona como cabría esperar, los usuarios culparán al software. Como diseñadores, puede creerse que el software se ejecuta en un entorno amigable donde todo se ha configurado para funcionar a la perfección. Sin embargo, los usuarios ejecutarán finalmente el software en un entorno que nada tiene que ver con la perfección, donde cualquier cosa puede ir mal e irá mal, y donde los defectos de sus aplicaciones saldrán con seguridad a la luz. Desde una perspectiva de seguridad, los defectos del software no son sólo una molestia; los intrusos pueden explotar las vulnerabilidades de las aplicaciones para atacar el sistema o el software sobre el que se esté ejecutando.

Diseñar software que controle adecuadamente las excepciones[15] permitirá el cumplimiento de dos objetivos: protegerá a las aplicaciones contra ataques y mejorará la satisfacción del usuario.

La necesidad de verificar las entradas realizadas por el usuario y, siempre que sea posible, la de impedir que se produzcan excepciones. Salvo que se controlen adecuadamente, las excepciones provocan el fallo de la aplicación.

Dónde se producen las excepciones

Las excepciones pueden producirse en cualquier lugar del código pero, con frecuencia, surgen cuando se produce una situación que el diseñador no pudo anticipar y que la lógica del programa no puede controlar con eficacia. Supóngase que un usuario introduce su nombre de usuario y su contraseña en el formulario de inicio de sesión del sistema de administración de empleados. Los errores que se producirán con mayor frecuencia es que el usuario introduzca un nombre de usuario y una contraseña que no sean válidos. Otro problema es que el nombre de usuario contenga caracteres que puedan provocar un ataque por inyección SQL. Ambos problemas se pueden controlar implantando un mecanismo de validación de entradas, pero existen todavía otros problemas potenciales que pueden provocar una excepción. Si la base de datos se encuentra en el servidor, una caída de la red puede provocar un mal funcionamiento de nuestro sistema. Si la base de datos se encuentra en la máquina local, el usuario puede eliminar la base de datos o abrirla con otra aplicación. Evidentemente, no es posible proteger a la aplicación contra todo. Existen algunas situaciones frecuentes que se debe tener siempre presente y que las aplicaciones deben detectar y controlar con elegancia:

• Entradas erróneas: como ya se vio anteriormente, si el sistema no detecta las entradas no válidas (por ejemplo, una cadena demasiado larga o un número que no pertenezca al rango esperado) la entrada puede provocar una excepción. Este punto también se aplica a la

[15] Los errores del tipo división por cero y archivo no encontrado son ejemplos de excepciones.

información pasada desde un sistema externo (si no controla los datos entrantes, pueden estar corrompidos). La mejor práctica es asumir que ninguna entrada es válida hasta que se demuestre lo contrario.

• Conflictos entre varios usuarios: Cuando dos usuarios del sistema intentan abrir y bloquear el mismo archivo o escribir el mismo registro de una base de datos, una o ambas acciones fallarán. Una buena práctica es buscar los lugares en los que los recursos (por ejemplo, archivos, bases de datos, puertos o dispositivos de hardware) se abren o acceden por primera vez y escribir el código necesario para detectar si los recursos se encuentran todavía en uso.

• Caídas de la red: Si un servidor no está operando (un cable de red se ha soltado o un servicio de Internet no está disponible) se generará una excepción en la aplicación. Lo que deberemos garantizar es que una caída de la red no va a corromper los datos, no nos hará perder detalles de una transacción y que el sistema se recuperará airosamente cuando la línea vuelva a funcionar.

• Archivos comprometidos o perdidos: Si se elimina o corrompe un archivo o, simplemente, éste se encuentra en un formato distinto al esperado, la aplicación puede producir resultados inesperados. Muchas cosas son las que pueden producir la corrupción de los datos, incluyendo la instalación de una vieja versión de la aplicación que lea o escriba archivos utilizando otro formato, o un usuario que, con intención o sin ella, modifique el contenido de un archivo de recursos.

• Fallos catastróficos: Similar a las caídas de la red, si un equipo cliente tiene un fallo catastrófico en mitad de un proceso (por ejemplo, debido a un corte del suministro eléctrico) ¿se podrá recuperar de forma elegante el cliente y cualquier componente del servidor después de un fallo catastrófico?

• Fatiga: ¿Qué pasa si la máquina en la que se está ejecutando la aplicación se agota y se queda sin espacio libre en el disco o sin memoria, o el procesador se encuentra demasiado ocupado como para procesar otro programa? Se trata de problemas difíciles de detectar porque pueden ocurrir sin previo aviso. La mejor opción es probar la aplicación en una máquina que trabaje prácticamente al límite y comprobar si la aplicación presenta un comportamiento aceptable.

El motivo de que todos estos problemas afecten a la seguridad es porque si un intruso encuentra que la aplicación es vulnerable en cualquiera de estas áreas puede provocar que se produzca una de estas condiciones. Por ejemplo, puede apagar el equipo o desconectarlo de la red en mitad de una operación de proceso de la tarjeta de crédito. Debe comprobarse que el sistema puede controlar ésta y cualquier otra situación con la debida precaución. Todas estas situaciones tienen un punto en común, ya que se encuentran relacionadas con el mecanismo que utiliza la aplicación para comunicarse con un recurso o estímulo externo. Aquí es donde debe garantizar la detección y el control de las excepciones. En términos de seguridad, la interfaz existente entre la aplicación y el resto del mundo se conoce como ***superficie de ataque***. Lo mejor es minimizar la superficie de ataque reduciendo el número de puntos en los que se reciben las entradas de los usuarios o de otros sistemas externos. Cualquier entrada introducida en el sistema tendrá que ser validada y deberá introducir una comprobación de excepciones en el código circundante.

Control de excepciones

La apuesta más segura es suponer que no puede detectar y controlar todas las circunstancias inesperadas y que las excepciones se producirán de forma inevitable. Cuando se produzca una excepción, deberá seguir una serie preestablecida de pasos para controlar la excepción:

• ESCRIBIR controles de excepciones del tipo *Try... Catch* u *On Error GoTo* para todas las interacciones con la base de datos. Es necesario poner especial atención al lugar de la base de datos que se accede por primera vez (ya que si hay caídas de la red o una especificación de archivo errónea para acceder a la base de datos, aquí será donde se detecte primero).

• INFORMAR del error al usuario explicándo brevemente qué ha salido mal y cuál es la siguiente tarea a realizar. Por ejemplos:

- ☑ "No fue posible iniciar la sesión. Inténtelo de nuevo".
- ☑ "No se puede crear un informe de administración. Contacte con el soporte técnico para obtener asistencia".
- ☑ "No se pudo abrir la base de datos. Envíe un mensaje de correo electrónico a support@empresa.com para obtener ayuda".

• PROPORCIONAR al usuario información relacionada con la persona o entidad con la que hay que ponerse en contacto en caso de que algo vaya mal. Idealmente, esta información debería encontrarse en el texto del error que el sistema muestra al usuario.

• REGISTRAR todos los detalles posibles de la excepción para que el administrador, el servicio técnico o el diseñador puedan saber qué ha sucedido en realidad.

• NO proporcione demasiada información en el mensaje de error mostrado al usuario. Por ejemplo, los siguientes mensajes de error adolecen de este defecto:

- ☑ "El nombre de usuario es válido pero la contraseña es incorrecta. Por favor, vuelva a intentarlo".
- ☑ "No se pudo abrir el archivo \\ServidordeRed\BasesdeDatos\MiBaseDatos.mdb".
- ☑ "Instrucción SQL SELECT * FROM Employee WHERE username = 'RKing*' no válida"

El motivo de que estos mensajes de error no sean del todo correctos es que divulgan demasiada información sobre los trabajos internos realizados por el sistema. El primer ejemplo informa al intruso de que ha introducido un nombre de usuario válido y le invita a probar con otras contraseñas hasta que obtenga la correcta. El segundo ejemplo comunica al intruso la ubicación precisa de la base de datos que está utilizando el sistema (si el intruso consigue acceder a la red ya sabrá dónde localizar la base de datos de su aplicación). El tercer mensaje defectuoso revela el contenido de la instrucción SQL, lo que, en ocasiones, permitirá al intruso ajustar su entrada para hacer que la instrucción SQL realice sus nefastos propósitos. Nunca deberá revelar al usuario final los trabajos internos realizados por el sistema.

Para un intruso que desee introducirse en el sistema esta información le resultará de vital importancia. Una buena práctica es no enviar nunca directamente al usuario la información que devuelve Visual Basic .NET cuando se produzca un error en la aplicación. En general, esta información suele carecer de importancia para todo el mundo salvo para el diseñador de la aplicación. En su lugar, la aplicación debería proporcionar mensajes utilizando términos que el usuario pueda comprender.

Al igual que el tema anterior las excepciones se trataran con detenimiento en el capítulo 7, donde se presentaran técnicas para su tratamiento.

5.2.5.1.1 ESTRUCTURAS PARA INTERCEPTAR EXCEPCIONES

Visual Basic .NET permite el empleo de dos tipos de control de excepciones[16]: el control de excepciones *Try…Catch* y el control de excepciones al viejo estilo *On Error GoTo*. El método que elija dependerá de los gustos personales, aunque el control de excepciones *Try...Catch* presenta algunas ventajas sobre *On Error GoTo* por varios motivos. En primer lugar, el control de excepciones *Try. .Catch* ofrece una mayor versatilidad porque podrá anidar un control de excepciones *Try... Catch* dentro de otro bloque *Try… Catch*. En segundo lugar, pueden agregarse varias cláusulas *Catch* para controlar diferentes tipos de excepciones y una cláusula *Finally* para ejecutar código al final del proceso con independencia de que se haya producido o no la excepción. El código mostrado a continuación muestra la sintaxis utilizada en un control de excepciones y el código que se ejecutará por estar contenido en una cláusula *Finally*:

```
Try
    'Algún código
catch exFileNotFound As System.IO.FileNotFoundeException
    'Controlador de excepciones para archivos no encontrados
Catch ex as Exception
    'Controlador de excepciones para todos los demás tipos de excepciones
Finally
    'código que siempre se ejecuta al final del procedimiento
End Try
```

Código fuente 5.F.

Por último el controlador de excepciones *Try. . . Catch* produce un código compilado más limpio, aunque los efectos de este hecho son despreciables. Es posible mezclar ambos tipos de controladores de excepciones utilizando *On Error GoTo* en una función y *Try. . Catch* en otra. Sin embargo, el compilador de Visual Basic no dejará utilizar ambos controladores de excepciones (*On Error GoTo y Try…Catch*) dentro de la misma función.

[16] Para tener una mayor perspectiva del tema de excepciones es recomendable revisar el mismo tema contenido en el capítulo 7 de este trabajo.

5.3. CONCLUSIÓN

Un sistema informático se compone de hardware, software, personal dedicado (administradores y desarrolladores) y lo más importante, datos (el motivo de todo el sistema). Cuando hablamos de operaciones principales el sistema debe permitir almacenar, procesar y transmitir esa información. En el almacenamiento y en la transmisión están sobretodo los puntos clave para que esa información pertenezca solamente a su dueño. Como se trabaja en el capítulo, la seguridad de los recursos conlleva un proceso que abarca el ciclo de desarrollo completo y en el que entran en juego varias tecnologías diferentes. Es posible conseguir aplicaciones con un nivel de seguridad elevado si las tareas de diseño, prueba e implementación se desarrollan con precaución.

Al final, en este capítulo se quiere dar una imagen de la importancia de establecer la seguridad desde el comienzo de una aplicación, y esto es desde que se diseña o se programa. Si bien es muy cierto la cadena es tan fuerte como el más débil de los eslabones.

Pues ciertamente vivimos en una era donde lo más importante es la información, y la tecnología se usa para sacar un mejor provecho de la información que interesa. La seguridad en cómputo no se trata solo de mantener alejado a los posibles intrusos o remover virus y gusanos diariamente; las actividades relacionadas con la seguridad informática tratan de proteger primeramente la calidad de la información que poseemos, y después proteger los medios que se usan para manipularla y difundirla a los interesados.

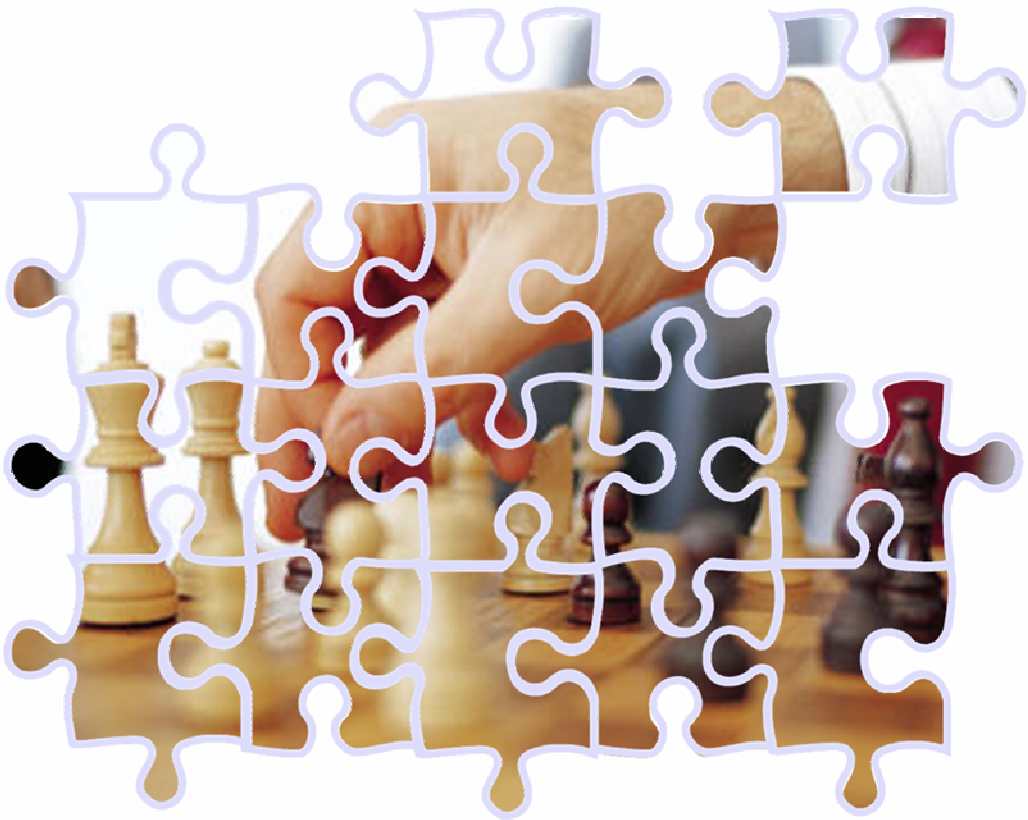
El manejo de los procedimientos de seguridad y de las personas que interactúan con los sistemas son considerados el eslabón más débil dentro de toda la seguridad informática. Existen una gran cantidad de amenazas y ataques directos o indirectos contra la infraestructura de las tecnologías de información, pero existen otros tipos de amenazas más efectivas que no requieren gran participación de la tecnología, y estos ataques son más efectivos que montar un complejo ataque por el hacker mas experimentado.

El problema de la seguridad en un sistema de cómputo se convierte entonces, como se ve a lo largo del capítulo, en un problema de conocimiento y educación. Cambiar la forma en que se ven y se hacen las cosas, se debe de pensar en evitar estar apagando incendios y realizar programación extra para prevenirlos.

Además quiero mostrar en este capítulo que los problemas de seguridad no son sólo los relacionados con la tecnología, sino que además es ahí donde se manifiestan principalmente. Por último, hay que admitir que hace falta mucha capacitación respecto al tema en todos los niveles, porque todos son los beneficiados o afectados con el buen funcionamiento de los sistemas o aplicaciones.

Capítulo 6

Estrategias para robustecer la seguridad en aplicaciones VB.NET



6. ESTRATEGIAS PARA ROBUSTECER LA SEGURIDAD EN APLICACIONES *VB.NET*

En este capítulo describo la necesidad de establecer una estrategia de seguridad y la razón por la que tiene que ser planeada y diseñada objetivamente para ayudar a los desarrolladores que son concientes de la seguridad. El desarrollar una estrategia tiene como objetivo proteger la disponibilidad, integridad y confidencialidad de los datos de los sistemas informáticos de las organizaciones e instituciones. La seguridad informática no debe solo ser de interés para administradores de recursos de información, directores de seguridad informática y los administradores, puesto que llega a tener un valor especial para todos aquellos comprometidos a establecer y cumplir políticas de seguridad, como lo son los programadores y por los cuales se realiza este trabajo de tesis.

Una estrategia ofrece un acercamiento sistemático a esta importante tarea y, como precaución final, también implica el establecimiento de planes de contingencia en caso de desastre.

A pesar que este proyecto quiere enfocar sus esfuerzos en estrategias de programación dentro del código, no es posible dejar de lado las estrategias externas al código que complementan la seguridad, como lo son las estrategias físicas (servidores, *harware*, redes…) o la humana (grupos de respuesta, planes de contingencia…). Dado que si se habla de una Verdadera Estructura de Seguridad aun cuando se presente una programación segura por parte del desarrollador deberán considerase los demás campos de seguridad informática que implican al Administrador, para que realmente sea de utilidad la Estrategia de Seguridad que se implemente.

Contemplando el aspecto administrativo y de desarrollo es posible observar que los datos de los sistemas informáticos están en constante peligro por varias causas: errores de los usuarios o ataques intencionados o fortuitos, como ya se estudiaron ampliamente en el capitulo anterior.

Es común que se produzcan accidentes y ciertas personas con intención de atacar el sistema pueden obtener acceso al mismo e interrumpir los servicios, inutilizar los sistemas o alterar, suprimir o robar información.

Por estos incidentes debe considerarse que los sistemas informáticos pueden necesitar protección lógica (programación, Software) y física (hardware, personal) en algunos de los siguientes rubros de la información:

La **Integridad** de la Información es la característica que hace que su contenido permanezca inalterado a menos que sea modificado por personal autorizado, y esta modificación sea registrada para posteriores controles o auditorias. Una falla de integridad puede estar dada por anomalías en el hardware, software, virus informáticos y/o modificación por personas que se infiltran en el sistema. El sistema contiene información que debe protegerse de modificaciones no autorizadas, imprevistas o accidentales. Por ejemplo, información de censos, indicadores económicos o sistemas de transacciones financieras

La **Disponibilidad** u Operatividad de la Información es su capacidad de estar siempre disponible para ser procesada por las personas autorizadas. Esto requiere que la misma se mantenga correctamente almacenada con el hardware y el software funcionando perfectamente y

que se respeten los formatos para su recuperación en forma satisfactoria. El sistema contiene información o proporciona servicios que deben estar disponibles puntualmente para satisfacer requisitos o evitar pérdidas importantes. Por ejemplo, sistemas esenciales de seguridad, protección de la vida y predicción de huracanes

La **Privacidad** o Confidencialidad de la Información es la necesidad de que la misma sólo sea conocida por personas autorizadas. En casos de falta de confidencialidad, la Información puede provocar severos daños a su dueño (por ejemplo conocer antecedentes médicos de una persona) o volverse obsoleta (por ejemplo: los planes de desarrollo de un producto que se "filtran" a una empresa competidora, facilitarán a esta última desarrollar un producto de características semejantes). Por ejemplo, datos que se van a difundir en un momento determinado (como, información parcial de informes), información personal e información comercial patentada

El **Control** sobre la información permite asegurar que sólo los usuarios autorizados pueden decidir cuando y como permitir el acceso a la misma.

La **Autenticidad** permite definir que la información requerida es válida y utilizable en tiempo, forma y distribución. Esta propiedad también permite asegurar el origen de la información, validando el emisor de la misma, para evitar suplantación de identidades.

Adicionalmente pueden considerar algunos aspectos adicionales relacionados con los anteriores, pero que incorporan algunos aspectos particulares:

- ☑ **Protección a la Réplica**: mediante la cual se asegura que una transacción sólo puede realizarse una vez, a menos que se especifique lo contrario. No se deberá poder grabar una transacción para luego reproducirla, con el propósito de copiar la transacción para que parezca que se recibieron múltiples peticiones del mismo remitente original.

- ☑ **No Repudio**: mediante la cual se evita que cualquier entidad que envió o recibió información alegue, ante terceros, que no la envió o recibió.

- ☑ **Consistencia**: se debe poder asegurar que el sistema se comporte como se supone que debe hacerlo ante los usuarios que corresponda.

- ☑ **Aislamiento**: este aspecto. íntimamente relacionado con la Confidencialidad, permite regular el acceso al sistema., impidiendo que personas no autorizadas hagan uso del mismo.

- ☑ **Auditoria**: es la capacidad de determinar qué acciones o procesos se están llevando a cabo en el sistema, así como quién y cuando las realiza.

Cabe definir Amenaza, en el entorno informático, corno cualquier elemento que comprometa al sistema.

En la figura 6.1 se organizan las amenazas dependiendo del las causas por las que se generan. Las amenazas pueden ser analizadas en tres momentos: antes del ataque, durante y después del mismo.

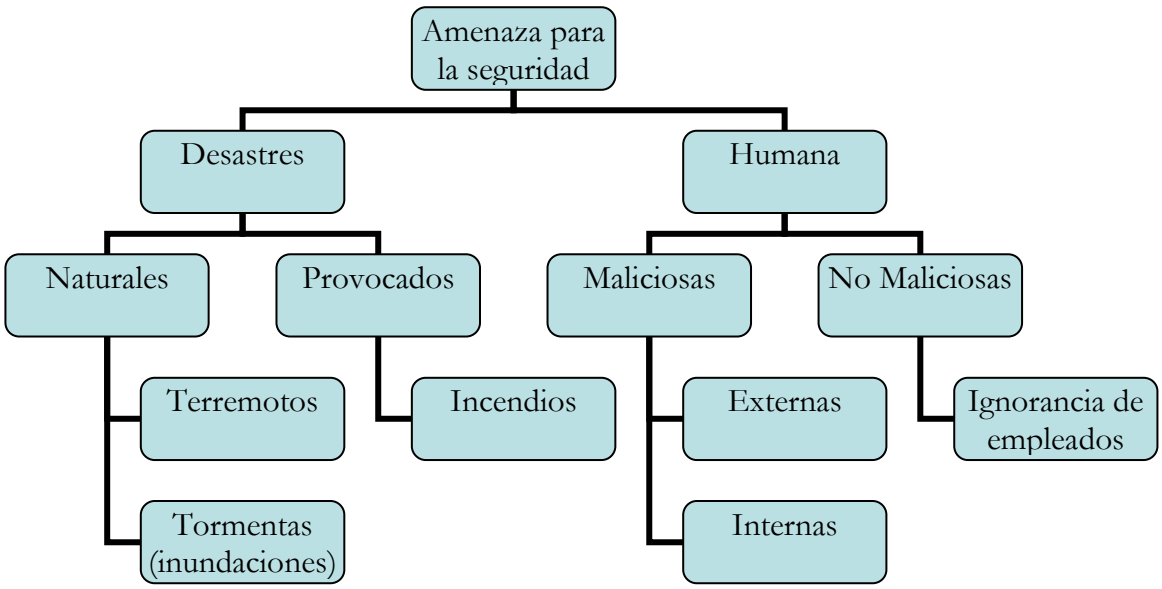


Figura 6.1. Organigrama de Amenazas para la seguridad

Estos mecanismos conforman políticas que garantizan la seguridad de los sistemas informáticos.

a. La **Prevención** (antes): mecanismos que aumentan la seguridad (o fiabilidad) de un sistema durante su funcionamiento normal. Por ejemplo el cifrado de información clara y posterior transmisión.

b. La **Detección** (durante): mecanismos orientados a revelar violaciones a la seguridad. Generalmente son programas de auditoría.

c. La **Respuesta** o **Recuperación** (después): mecanismos que se aplican, cuando la violación del sistema ya se ha detectado, para retomar éste a su funcionamiento normal. Por ejemplo recuperación desde las copias de seguridad (backup) realizadas.

Las preguntas que se hacen un ante un problema de seguridad, están relacionadas con medidas defensivas que no solucionan un problema dado, sólo lo transforma o retrasa. La amenaza o riesgo sigue allí y las preguntas que este técnico debería hacerse son: ¿Cuánto tardará la amenaza en superar la "solución" planteada?, ¿Cómo se hace para detectarla e identificarla a tiempo?, ¿Cómo se hace para neutralizarla?...

Para responderlas debe definirse Riesgo como "la proximidad o posibilidad de daño sobre algún bien". Sean actos naturales, errores, omisiones humanas o actos intencionales, cada riesgo debería ser atacado de las siguientes maneras:

1. Minimizando la posibilidad de su ocurrencia.
2. Reduciendo al mínimo el perjuicio producido, si no ha podido evitarse que ocurriera.
3. Diseño de métodos para la más rápida recuperación de los daños experimentados.
4. Corrección de las medidas de seguridad en función de la experiencia recogida.

El **Daño** es el resultado de la amenaza; aunque esto es sólo la mitad del axioma. El daño también es el resultado de la no acción, o acción defectuosa, del protector. El daño puede

producirse porque el protector no supo identificar adecuadamente la amenaza y silo hizo, se impusieron criterios comerciales por encima de los de seguridad. Derivándose responsabilidades para la amenaza pero también para la figura del protector.

El protector será el encargado de detectar cada una de las Vulnerabilidades del sistema que pueden ser explotadas y empleadas, por la amenaza, para comprometerlo. También será el encargado de aplicar las Contramedidas adecuadas.

La Seguridad indicara el índice en que un Sistema informático está libre de todo peligro, daño o riesgo. Esta característica es muy difícil de conseguir en un 100% por lo que sólo se habla de Fiabilidad y se la define como "la probabilidad de que un sistema se comporte tal y como se espera de él", y se habla de Sistema Fiable en vez de sistema seguro.

Para garantizar que un sistema sea fiable se deberá garantizar las características de Integridad, Operatividad, Privacidad, Control y Autenticidad. Se deberá conocer "qué es lo que se quiere proteger", "de quién se quiere proteger", "cómo se puede lograr esto legislativa y técnicamente"; para concluir con la formulación de estrategias adecuadas de seguridad enfocadas a la disminución de los riesgos.

Comprender y conocer de seguridad ayuda a llevar a cabo análisis sobre los Riesgos, las Vulnerabilidades, Amenazas y Contramedidas; evaluar las ventajas o desventajas de la situación: a decidir medidas técnicas y tácticas metodológicas, físicas e informáticas, con base de las necesidades de seguridad.

Contemplar los aspectos anteriores permite, dentro de una estrategia de seguridad, determinar el tiempo, dinero y esfuerzo que hay que invertir para desarrollar las políticas, herramientas y controles de seguridad apropiados. Cada institución debe analizar sus necesidades específicas y determinar sus requisitos y limitaciones en cuanto a recursos económicos y de programación. Cada sistema informático, entorno y política organizativa es distinta, lo que hace que cada servicio y cada estrategia de seguridad sean únicos. Sin embargo, ***los fundamentos de una buena seguridad siguen siendo los mismos*** y este capitulo se centra en dichos principios para la adecuada programación, teniendo de la mano a otros aspectos físicos y humanos no menos importantes.

Aunque una estrategia de seguridad puede ahorrar mucho tiempo a la institución y proporcionar importantes recomendaciones de lo que se debe hacer, ***la seguridad no es una actividad puntual***. Es una parte integrante del ciclo vital de los sistemas. Las actividades que se describen en este proyecto suelen requerir actualizaciones periódicas o las revisiones correspondientes. Estos cambios se realizan cuando las configuraciones y otras condiciones y circunstancias cambian considerablemente o cuando son modificadas las leyes y normas organizativas. Éste es un proceso iterativo, nunca termina, por lo cual debe revisarse y probarse con periodicidad.

6.1. ¿QUÉ ES UNA ESTRATEGIA DE SEGURIDAD INFORMÁTICA?

Respondiendo a la pregunta, es necesario en principio definir Estrategia como un proceso compuesto por el conjunto de reglas que aseguran una decisión óptima en cada momento, ahora bien, Seguridad aun cuando es un término muy relativo, la palabra "Seguro" es la condición de estar libre y exento del peligro. Si conjunto ambas ideas, puedo definir una Estrategia de Seguridad como aquel proceso que libera y exenta de peligros, partiendo de un conjunto de reglas que optimizan las decisiones en cada momento de peligro. Para contar con una definición completa es necesario definir que el "momento de peligro" es la oportunidad o el instante especifico cuando se presenta un problema, desde mi punto de vista, son tres: Prevención (antes), Detección (durante) y Respuesta o Recuperación (después). Además de una especial y que no debe pasarse por alto en las tres anteriores: el Análisis (antes, después y durante).

Este último es de vital importancia para mejoramiento y corrección de las estrategias que se planten en un momento dado.

Los datos de los sistemas informáticos están en constantes peligros o amenaza por varias causas: errores de los usuarios o ataques malintencionados o fortuitos. Pueden producirse accidentes y ciertas personas con intención de atacar el sistema, pueden tener acceso al mismo e interrumpir los servicios, inutilizar los sistemas o alterar, suprimir o robar información.

El poder disponer de una "Estrategia de Seguridad Informática" se hace necesario e imprescindible en instituciones que deseen mantener Buenas Prácticas de Seguridad Corporativa.

La seguridad informática debe abarca los conceptos de seguridad física y seguridad lógica.

La seguridad física se refiere a la protección del hardware y de los soportes de datos, así como a la de los edificios e instalaciones que los albergan. Contempla las situaciones de incendios, sabotajes, robos, catástrofes naturales, etc.

La seguridad lógica se refiere a la seguridad de uso del software, a la protección de los datos, procesos y programas, así como la del ordenado y autorizado acceso de los usuarios a la información.

En este trabajo de tesis, aun cuando en ambas trabajo, doy una mayor prioridad a la seguridad lógica., teniendo en cuenta que son un "todo" al momento de hablar de Seguridad.

El título de la tesis es "Estrategias de programación para robustecer la seguridad informática al desarrollar aplicaciones en Visual Basic .NET" por lo cual me limitaré a lo largo de este tema (y como lo he venido haciendo en capítulos anteriores) al enfoque de la seguridad lógica, a la seguridad que involucra al programador o al desarrollador de sistemas de cómputo. La mayoría de las estrategias de seguridad que se desarrollen estarán enfocadas al programador, y solo actuará como recurso de apoyo el Administrador. Por tales motivos cuando hable de una estrategia de seguridad contemplaré con mayor énfasis a las Estrategias de programación.

6.1.1. REQUERIMIENTOS DE UN SISTEMA SEGURO

1. **Reconocimiento**: cada usuario debe identificarse al usar el sistema y cada operación del mismo es registrada con esta identificación. En este proceso se consigue que no se produzca un acceso y/o manipulación indebida de los datos o que en su defecto, esta quede registrada.

2. **Integridad**: un sistema integro es aquel en el que todas las partes que lo constituyen funcionan en forma correcta y en su totalidad.

3. **Aislamiento**: Los datos utilizados por un usuario deben ser independientes de los de otro física y lógicamente (usando técnicas de ocultación y/o compartimiento). También se debe lograr independencia entre los datos accesibles y los considerados críticos.

4. **Auditabilidad**: procedimiento utilizado en la elaboración de exámenes, demostraciones, verificaciones o comprobaciones del sistema. Estas comprobaciones deben ser periódicas y tales que brinden datos precisos y aporten confianza a la dirección. Deben apuntar a contestar preguntas como: ¿El uso del sistema es adecuado?, ¿El sistema se ajusta a las normas internas y externas vigentes?, ¿Los datos arrojados por el sistema se ajustan a las expectativas creadas?, ¿Todas las transacciones realizadas por el sistema pueden ser registradas adecuadamente?, ¿Contienen información referentes al entorno: tiempo, lugar, autoridad, recurso, empleado, etc.?,

5. **Controlabilidad**: todos los sistemas y subsistemas deben estar bajo control permanente.

6. **Recuperabilidad**: en caso de emergencia, debe existir la posibilidad de recuperar los recursos perdidos o dañados.

7. **Administración y Custodia**: la vigilancia permite conocer, en todo momento, cualquier suceso, para luego realizar un seguimiento de los hechos y permitir una realimentación del sistema de seguridad, de forma tal de mantenerlo actualizado contra nuevas amenazas.

6.1.2. OBJETIVOS DE LA ESTRATEGIA DE PROGRAMACIÓN

El contar con estrategias de programación para la seguridad informática permite:

- ☑ Asegurar que determinados recursos del sistema, desde el equipo individual que contiene datos y programas hasta toda la red, estén disponibles sólo para los usuarios autorizados.
- ☑ Simplificar la tarea de implementación de seguridad en aplicaciones o sistemas en desarrollo.
- ☑ Robustecer y mejorar sistemas o aplicaciones en desarrollo y desarrolladas.
- ☑ Identificar las principales problemáticas de seguridad que deben solucionar los programadores cuando desarrollan aplicaciones.
- ☑ Diseñar e instrumentar nuevos mecanismos y estructuras idóneas, a partir de un conjunto de reglas.
- ☑ Implementar políticas y controles de seguridad con el objeto de aminorar los posibles ataques y amenazas.

☑ Protección de bases de datos y redes informáticas frente a diversos tipos de amenazas y riesgos.

La programación segura tiene por finalidad no sólo proteger los recursos físicos y lógicos, sino también de impedir que los usuarios dañen el sistema involuntariamente. El contar con estrategias de programación proporciona al desarrollador la herramienta necesaria para diseñar sistemas que den la **confianza** al ser usados.

6.1.3. ¿DE QUIÉN PROTEGERSE?

Se llama Intruso o Atacante a la persona que accede o intenta acceder sin autorización a un sistema ajeno, ya sea en forma intencional o no.

Ante la pregunta de los tipos de intrusos existentes actualmente, "Los tipos de intrusos podríamos caracterizarlos desde el punto de vista del nivel de conocimiento, formando una pirámide como se muestra en la Figura 6.2.

1. Clase A: el 80% en la base son los nuevos intrusos que son pequeños grupitos que se juntan y bajan programas de Internet y prueban.
2. Clase B: es el 12% son más peligrosos, saben compilar programas aunque no saben programar. Prueban programas, conocen como detectar que sistema operativo que está usando la víctima, prueban las vulnerabilidades del mismo e ingresan por ellas.
3. Clase C: es el 5%. Es gente que sabe, que conoce y define sus objetivos. A partir de aquí buscan todos los accesos remotos e intentan ingresar.
4. Clase D: el 3%. Cuando entran a determinados Sistemas buscan la información que necesitan.

Para llegar desde la base hasta el último nivel se tarda desde 4 a 6 años, por el nivel de conocimiento que se requiere asimilar. Es práctica, conocer, programar, mucha tarea y mucho trabajo".

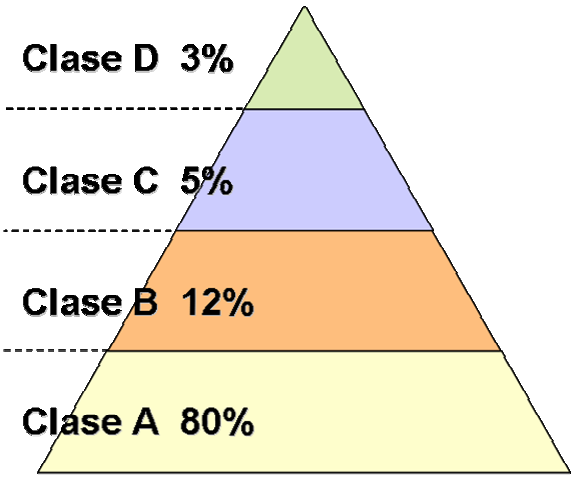


Figura 6.2 Tipo de Intrusos.

6.1.4. ¿QUÉ DEBE PROTEGERSE?

Cuando un sistema es atacado el agresor puede llegar a disponer de diversos elementos de una computadora, desde los datos e información que el sistema maneja pasando por las herramientas o software con los que el sistema se apoya para manipular y "proteger" los datos, hasta agentes como el disco duro, la memoria o el procesador donde se ejecuta la aplicación.

Por lo tanto en cualquier sistema informático existen tres elementos básicos a proteger: el hardware, el software y los datos. Por hardware entendemos el conjunto de todos los sistemas físicos del sistema informático: CPU, cableado, impresoras, CD-ROM, cintas, componentes de comunicación. El software son todos los elementos lógicos que hacen funcional al hardware: sistema operativo, aplicaciones, utilidades.

Entendemos por datos al conjunto de información lógica que maneja el software y el hardware: bases de datos, documentos, archivos. Se habla de un cuarto elemento llamado fungible; que son los aquellos que se gastan o desgastan con el uso continuo: papel, *tonner*, tinta, cintas magnéticas, disquetes.

De los cuatro, los datos que maneja el sistema serán los más importantes ya, que son el resultado del trabajo realizado. Si existiera daño del hardware, software o de los elementos fungibles, estos pueden adquirirse nuevamente desde su medio original; pero los datos obtenidos en el transcurso del tiempo por el sistema son imposibles de recuperar: hemos de pasar obligatoriamente por un sistema de copias de seguridad, y aun así es difícil de devolver los datos a su forma anterior al daño.

Para cualquiera de los elementos descriptos existen multitud de amenazas y ataques que se los puede clasificar en:

1. **Ataques Pasivos**: el atacante no altera la comunicación, sino que únicamente la "escucha" o monitoriza, para obtener información que está siendo transmitida. Sus objetivos son la intercepción de datos y el análisis de tráfico. Generalmente se emplean para:

- Obtención del origen y destinatario de la comunicación, a través de la lectura de las cabeceras de los paquetes monitorizadlos.
- Control del volumen de tráfico intercambiado entre las entidades monitorizadas, obteniendo así información acerca de actividad o inactividad inusuales.
- Control de las horas habituales de intercambio de datos entre las entidades de la comunicación, para extraer información acerca de los períodos de actividad.

Es posible evitar el éxito, si bien no el ataque, mediante el cifrado de la información y otros mecanismos que se tratan más adelante.

2. **Ataques Activos**: estos ataques implican algún tipo de modificación del flujo de datos transmitido o la creación de un falso flujo de datos. Generalmente son realizados por *hackers*, piratas informáticos o intrusos remunerados y se los puede subdividir en cuatro principales categorías, ejemplificadas en la figura 6.3:

- Interrupción: si hace que un objeto del sistema se pierda, quede inutilizable o no disponible.

- Intercepción: si un elemento no autorizado consigue el acceso a un determinado objeto del sistema.
- Modificación: si además de conseguir el acceso consigue modificar el objeto.
- Fabricación: se consigue un objeto similar al original atacado de forma que es difícil distinguirlos entre sí.
- Destrucción: es una modificación que inutiliza el objeto.

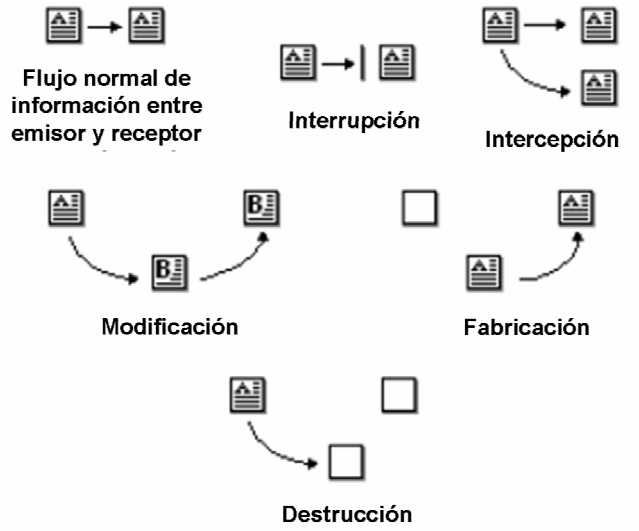


Figura 6.3 Tipos de Ataques Activos (www.cert.org)

Con demasiada frecuencia se cree que los *hackers* son lo únicos que amenazan nuestro sistema, siendo pocos los administradores que consideran todos los demás riesgos analizados.

6.2. ¿QUÉ TIPOS DE ESTRATEGIAS EXISTEN?

En cada sistema, el plan de seguridad debe incluir una estrategia proactiva y otra reactiva.

La **ESTRATEGIA PROACTIVA** o de previsión de ataques es un conjunto de pasos que ayuda a reducir al mínimo la cantidad de puntos vulnerables existentes en las políticas de seguridad y a desarrollar planes de contingencia. La determinación del daño que un ataque va a provocar en un sistema y las debilidades y puntos vulnerables explotados durante este ataque ayuda a desarrollar la estrategia proactiva.

La **ESTRATEGIA REACTIVA** o estrategia posterior al ataque ayuda al personal de seguridad o al sistema mismo a evaluar el daño que ha causado el ataque, a repararlo o a implementar el plan de contingencia desarrollado en la estrategia proactiva, a documentar y aprender de la experiencia, y a conseguir que las funciones del sistema se normalicen lo antes posible.

Para establecer una estrategia adecuada es conveniente pensar una política de protección en los distintos niveles que esta debe abarcar y que no son ni más ni menos que los estudiados: Física, Lógica, Humana y la interacción que existe entre estos factores.

En cada caso considerado, el plan de seguridad debe incluir una estrategia Proactiva y otra Reactiva. Con respecto a la postura que puede adoptarse ante los recursos compartidos:

- Lo que no se permite expresamente está prohibido: significa que la organización proporciona una serie de servicios bien determinados y documentados, y cualquier otra cosa está prohibida.
- Lo que no se prohíbe expresamente está permitido: significa que, a menos que se indique expresamente que cierto servicio no está disponible, todos los demás sí lo estarán.

Estas posturas constituyen la base de todas las demás políticas de seguridad y regulan los procedimientos puestos en marcha para implementarlas. Se dirigen a describir qué acciones se toleran y cuáles no.

6.3. ¿QUÉ ASPECTOS DEBEN CONSIDERAR LAS ESTRATEGIAS DE SEGURIDAD?

6.3.1. POLÍTICAS Y CONTROLES DE SEGURIDAD

El establecimiento eficaz de un conjunto de políticas y controles de seguridad requiere el uso de un método para determinar los puntos vulnerables que existen en los sistemas, en las políticas y controles de seguridad que los protegen. El estado actual de las políticas de seguridad informática se puede determinar mediante la revisión de la siguiente lista.

- ☑ Política de seguridad informática física, como los controles de acceso físico.
- ☑ Políticas de seguridad de la Políticas de seguridad de los datos.
- ☑ Planes y pruebas de contingencias y de recuperación de desastres.
- ☑ Conocimiento y formación en seguridad informática.
- ☑ Políticas de administración y coordinación de la seguridad informática.

Otros documentos que contienen información importante como:

- ☑ Contraseñas del BIOS de los equipos.
- ☑ Contraseñas para la configuración de puertas de enlace o ruteadores.
- ☑ Documentos de control de acceso.
- ☑ Otras contraseñas de administración de dispositivos.

Por lo tanto la revisión de las políticas debe tomar nota de las áreas en las que las políticas son deficitarias y examinar los documentos que existan.

6.3.2. ANÁLISIS DE RIESGOS

El análisis de riesgos supone más que el hecho de calcular la posibilidad de que ocurran cosas negativas.

- ☑ Se debe poder obtener una evaluación económica del impacto de estos sucesos. Este valor se podrá utilizar para contrastar el costo de la protección de la información en análisis, versus el costo de volverla a producir (reproducir).
- ☑ Se debe tener en cuenta la probabilidad que sucedan cada uno de los problemas posibles. De esta forma se pueden priorizar los problemas y su costo potencial desarrollando un plan de acción adecuado.

☑ Se debe conocer qué se quiere proteger, dónde y cómo, asegurando que con los costos en los que se incurren se obtengan beneficios efectivos. Para esto se deberá identificar los recursos (hardware. software, información, personal. accesorios. etc.) con que se cuenta y las amenazas a las que se está expuesto.

La evaluación de riesgos y presentación de respuestas debe prepararse de forma personalizada para cada organización pero se pueden presuponer algunas preguntas que ayudan:

- "¿Qué puede ir mal?"
- "¿Con qué frecuencia puede ocurrir?"
- "¿Cuáles serían sus consecuencias?"
- "¿Qué fiabilidad tienen las respuestas a las tres primeras preguntas?"
- "¿Se está preparado para abrir las puertas del negocio sin sistemas, por un día, una semana, cuanto tiempo?"
- "¿Cuál es el costo de una hora sin procesar, un día, una semana?"
- "¿Cuánto tiempo se puede estar *off-line* sin que los clientes se vayan a la competencia?"
- "¿Se tiene forma de detectar a un empleado deshonesto en el sistema?"
- "¿Se tiene control sobre las operaciones de los distintos sistemas?"
- "¿A que se llama información confidencial y/o sensitiva?"
- "¿La información confidencial y sensitiva permanece así en los sistemas?"
- "¿La seguridad actual cubre los tipos de ataques existentes y está preparada para adecuarse a los avances tecnológicos esperados?"
- "¿A quien se le permite usar que recurso?"
- "¿Quién es el propietario del recurso? y ¿quién es el usuario con mayores privilegios sobre ese recurso?"
- "¿Cuáles serán los privilegios y responsabilidades del Administrador vs. la del usuario?"
- "¿Cómo se actuará si la seguridad es violada?"

Una vez obtenida la lista, de cada uno de los riesgos se efectuará un resumen del tipo:

Tipo de Riesgo	Factor
Robo de hardware	Alto
Robo de información	Alto
Vandalismo	Medio
Fallas en los equipos	Medio
Virus Informáticos	Medio
Equivocaciones	Medio
Accesos no autorizados	Medio
Fraude	Bajo
Fuego	Muy Bajo
Terremotos	Muy Bajo

Tabla 6.1 tipo de Riesgo-Factor

Basados en la tabla 6.1 deben tomarse las medidas pertinentes de seguridad para cada caso en particular, cuidando incurrir en los costos necesarios según el factor de riesgo representado.

6.3.2.1. NIVELES DE RIESGO

Como puede apreciase en la Tabla los riesgos se clasifican por su nivel de importancia y por la severidad de su pérdida:

1. Estimación del riesgo de pérdida del recurso (R_i)
2. Estimación de la importancia del recurso (I_i)

Para la cuantificación del riesgo de perder un recurso, es posible asignar un valor numérico de 0 a 10, tanto a la importancia del recurso (10 es el recurso de mayor importancia) como al riesgo de perderlo (10 es el riesgo más alto).

El riesgo de un recurso será el producto de su importancia por el riesgo de perderlo:

$$W\ R_i = R_i * I_i$$

Luego con la siguiente fórmula es posible calcular el riesgo general de los recursos de la red:

$$W_R = \frac{(WR_1 * I_1 + WR_2 * I_2 + \ldots + WR_n * I_n)}{I_1 + I_2 + \ldots + I_n}$$

Otros factores que debe considerar para el análisis de riesgo de un recurso de red son su disponibilidad, su integridad y su carácter confidencial los cuales pueden incorporarse a la fórmula para ser evaluados

Ejemplo: el Administrador de una red ha estimado los siguientes riesgos y su importancia para los elementos de la red que administra:

RECURSO	RIESGO (R_I)	IMPORTANCIA (I_I)	RIESGO EVALUADO (R_I*I_I)
Router	6	7	42
Gateway	6	5	30
Servidor	10	10	100
PC's	9	2	18

Tabla 6.1. Valuación de Riesgos

Aquí ya puede apreciarse que el recurso que más debe protegerse es el servidor. Para la obtención del riesgo total de la red calculamos:

$$W = (42+30+100+18)/\ 7+5+10+2=7.92$$

Al ver que el riesgo tol de la red es de casi 8 puntos sobre 10 debe pensarse seriamente en buscar las probables causas que pueden provocar problemas a los servicios brindados por los elementas evaluados.

6.3.2.2. MODELADO DE RIESGOS

El modelado de riesgos nos permite comprender los Peligros Potenciales a los cuales la aplicación o sistema está inmerso.

- ☑ Es necesario comprender los peligros para construir aplicaciones seguras
- SSL no resuelve todos los problemas
- ☑ Detectar diferentes defectos mediante revisiones de código y pruebas
- Defectos de Diseño vs. Defectos de Implementación
- ☑ Detectar defectos que puedan de algún otro modo ser encontrados por los atacantes

El Proceso de Modelado de Riesgos consiste en aplicar las siguientes acciones:

- ☑ Descomponer la Aplicación
  - DFD, UML, etc.
- ☑ Construir un Árbol de Peligrosidad
  - Identificar los peligros por tipo
- ☑ Categorizar los Peligros Usando STRIDE
  - *Spoofing identity* (Fingir identidad)
  - *Tampering with data* (Daño a Datos)
  - *Repudiation* (Repudio)
  - *Information disclosure* (Revelación de info.)
  - *Denial of Service* (Negación de Servicio)
  - *Elevation of Privilege* (Aumento de privilegios)
- ☑ Asignar escalas de Peligros / o de Riesgo Estimado
  - Es posible utilizar el DREAD
    - *Damage Potential*
    - *Reproducibility*
    - *Exploitability*
    - *Affected users*
    - *Discoverability*
  - Daño Potencial x Probabilidad de Ocurrencia

6.3.3. ANÁLISIS DE AMENAZAS

Una vez conocidos los riesgos, los recursos que se deben proteger y como su daño o falta pueden influir en la organización es necesario identificar cada una de las amenazas y vulnerabilidades que pueden causar estas bajas en los recursos. Como ya mencioné existe una relación directa entre amenaza y vulnerabilidad a tal punto que sin una no existe la otra tampoco.

Se suele dividir las amenazas existentes según su ámbito de acción:

- Desastre del entorno (Seguridad Física).
- Amenazas del sistema (Seguridad Lógica).
- Amenazas en la red (Comunicaciones).
- Amenazas de personas (*Insiders-Outsiders*).

Se debe disponer de una lista de amenazas (actualizadas y particulares) para ayudar a los administradores de seguridad a identificar los distintos métodos, herramientas y técnicas de ataque que se pueden utilizar. Es importante que tanto los Administradores como programadores actualicen constantemente sus conocimientos en esta área, ya que los nuevos métodos, herramientas y técnicas para sortear las medidas de seguridad evolucionan de forma continua.

La metodología de análisis se basa en los distintos ejemplos (uno para cada tipo de amenaza) y contempla como puede ayudar una política de seguridad en caso de su existencia.

Las listas de amenazas, ayudan a los administradores y programadores de seguridad a identificar los distintos métodos, herramientas y técnicas de ataque que se pueden utilizar en los ataques. Los métodos pueden abarcar desde virus y gusanos a la adivinación de contraseñas y la interceptación del correo electrónico. Es importante que se actualicen constantemente sus conocimientos en esta área, ya que los nuevos métodos, herramientas y técnicas para sortear las medidas de seguridad evolucionan de forma continua.

Es recomendable realizar tablas de Amenaza que agrupen los riesgos específicos, por ejemplo las Amenazas a la red que se muestran en las tablas 6.3 y 6.4.

RIESGO	EJEMPLO
Reunión de Información	Revisión de puertos
	Usar trazado de ruteo para detectar la topología de la red.
	Usar peticiones en broadcast para enumerar los hosts en las subredes
Espionaje	Capturar el tráfico de red para robar passwords
Negación de servicio (DoS)	Desbordamiento de solicitudes SYN
	Desbordamiento de peticiones ICMP
	Paquetes malformados
Origen falso	Paquetes con dirección original falsa

Tabla 6.3. Riesgos de red

O las Amenazas a la aplicación

RIESGO	EJEMPLOS
Inyección de SQL	Incluir un comando DROP TABLE en un campo de entrada
Cross-site scripting	Usar código malicioso del lado del cliente para robar cookies
Daño a campos ocultos	Maliciosamente cambiar el valor de un campo oculto.

Espionaje	Usando un analizador de tráfico para robar passwords y cookies del tráfico en conexiones sin encriptación.
Secuestro de Sesión	Usando una cookie de Identificación de sesión para acceder al estado de sesión de otro usuario.
Identidad falsa	Usando una cookie de autentificación para pasar por otro usuario.
Revelación de información	Permitir que un usuario vea un trazado del stack cuando ocurre una excepción

Tabla 6.4. Riesgos de aplicación

6.3.3.1. EJERCICIO DE ANÁLISIS DE AMENAZAS

Para realizar un análisis de amenazas deberá seguir el siguiente procedimiento en la aplicación:

- ☑ Dedicar un tiempo al análisis de amenazas.
- ☑ Planificar y documentar su análisis de amenazas.
- ☑ Crear una lista de amenazas.
- ☑ Asignar prioridades a las amenazas.

DEDICAR TIEMPO

La fase de análisis de amenazas implica hablar y dibujar algunos diagramas de la arquitectura. El diseñador, el personal de pruebas y todo el mundo relacionado con la seguridad, y que tengan algo que decir sobre el tema que se está tratando, debe reunirse para llevar a cabo un análisis de amenazas. Un pequeño grupo de gente puede generar una lista de amenazas con gran rapidez sin más que mirar los diagramas de arquitectura. El costo no debe ser una limitación cuando se trata de generar una lista de amenazas. Sin embargo, el costo se convierte en un problema importante cuando:

- Realiza un análisis de amenazas de un proyecto ya existente cuyo código ya ha sido desarrollado y para el que nunca antes se ha efectuado un análisis de amenazas.
- Tiene que realizar los cambios necesarios para mejorar la resistencia de su aplicación frente a los ataques.

El costo es importante para los proyectos que ya existen porque, como parte del análisis de amenazas, tiene que revisarse el código existente para asegurar que es seguro frente a los ataques. Además, si en un proyecto ya existente se localiza un importante problema de seguridad, puede que se necesite invertir una considerable cantidad de tiempo para resolver estos problemas.

Si se está comenzando un nuevo proyecto en el que todavía no se haya escrito el código, resulta adecuado efectuar el análisis de las amenazas utilizando la especificación propuesta para el producto como base para este análisis.

Algunos componentes de la aplicación pueden presentar un mayor riesgo que otros, dependiendo del propósito del componente. Por ejemplo, un componente que sea accesible y al que se le pueda llamar desde Internet o desde una red, tendrá más riesgo que un componente

utilizado internamente por la aplicación como, por ejemplo, un componente interno de una biblioteca matemática. Los componentes que tengan un mayor nivel de riesgo deben ser sometidos a un mayor nivel de análisis. Por ejemplo, en estos casos debe realizarse un análisis completo de las amenazas revisando el diseño de la aplicación, su arquitectura y su código (si ya está escrito). Para los componentes de bajo riesgo puede decidir que no necesita efectuar un análisis de amenazas o puede limitar su análisis de amenazas a una inspección superficial de los métodos utilizados.

PLANIFICAR Y DOCUMENTAR SU ANÁLISIS DE AMENAZAS

Debe crearse un documento como parte del análisis de amenazas. El documento incluirá un plan sobre cómo debe llevar a cabo el análisis de las amenazas, expresando los miembros del equipo que estarán implicados, las técnicas de análisis utilizadas, un calendario propuesto y comentarios que indiquen los análisis que no se efectuarán por motivos de costo o por otras prioridades.

A medida que se avanza en el proceso de análisis de amenazas debe incluirse en el documento un diagrama general de la arquitectura de su aplicación, una lista de las amenazas identificadas para cada componente y un resumen de las vulnerabilidades que se encuentren en la aplicación. Esta información será de utilidad en el futuro cuando se vaya a añadir nuevas funciones a la aplicación. Una revisión del análisis de amenazas puede ayudar a mejorar la seguridad de las nuevas funciones basándose en las amenazas previamente descubiertas.

CREAR UNA LISTA DE AMENAZAS

Puede utilizarse diversas técnicas para generar una lista exhaustiva de las amenazas. Entre las más importantes, cabe destacar:

☑ Dibujar un diagrama de la arquitectura de su aplicación y evaluar los componentes que la forman, las relaciones entre los componentes y las entradas aceptadas por dichos componentes.

El diagrama debe incluir los componentes más importantes (.EXE y .DLL) que la forman, dibujando líneas entre los componentes para indicar comunicación o flujos de datos entre ellos. Si se está trabajando en un proyecto de grandes dimensiones, cada equipo de desarrollo del proyecto debe crear un diagrama limitado a los componentes que constituyan su parte de trabajo.

Es recomendado asumir que uno es un atacante y pensar en todas las estratagemas que podría utilizar para poner en peligro a la aplicación. Basándose en este diagrama, ¿qué amenazas puede ver? ¿Qué intentará hacer un atacante? Algunas sugerencias para empezar son: ¿Se puede descifrar la contraseña de un usuario, permitiendo al atacante iniciar la sesión como usuario?, otra forma es suponer situaciones, por ejemplo:

• Un atacante puede intentar iniciar la sesión de un determinado usuario un número de veces superior al permitido por la configuración de la seguridad de la aplicación, para impedir que el auténtico usuario se pueda conectar posteriormente durante el periodo de

tiempo especificado. Por ejemplo, supongase que la aplicación ha sido diseñada para bloquear la cuenta de un usuario después de intentar sin éxito iniciar tres veces una sesión.

• El atacante puede probar introducir instrucciones SQL o *VBScript* en los campos del nombre de usuario y contraseña para provocar un ataque de inyección SQL o de programación de sitio cruzado. Por ejemplo, si el nombre del usuario se muestra como parte de un mensaje de error, se podría facilitar un ataque de programación de sitio cruzado como se comenta en "ataques y métodos de respuesta" en este capítulo.

• Un atacante puede interceptar el nombre de usuario y la contraseña enviada a través de la conexión HTTP entre el cliente y el servidor y, posteriormente, iniciar la sesión como usuario.

• El atacante puede intentar la desactivación de los archivos de comandos para impedir la validación de la entrada del lado del cliente.

• Un atacante puede ver el archivo HTML contenido en el explorador para buscar comentarios que revelaran la estructura de directorios de la aplicación o la base de datos *back-end* que se estuviera utilizando.

• Si se utiliza una *cookie* para iniciar la sesión (como sucede en caso de que se encuentre seleccionada la casilla de verificación "Iniciar sesión de forma automática), el atacante puede intentar interceptar la *cookie* para iniciar una sesión como dicho usuario. Al actuar así, el atacante puede iniciar la sesión como dicho usuario sin conocer ni su nombre ni su contraseña.

• El atacante puede intentar conectarse directamente al servidor Web basado en *Internet Information Services* (JIS), que almacena la aplicación ASP.NET del sistema de administración de empleados, para obtener así acceso directo a archivos tales como *Web.Config* o la base de datos.

El objetivo de este ejercicio es enumerar con rapidez todas las cosas que podría intentar un atacante para poner en peligro la aplicación o el servidor.

☑ Revisar la especificación o el código de la aplicación.

Una técnica muy eficaz para la revisión implica la exploración del código para localizar métodos o palabras clave de *Visual Basic .NET* que indiquen que la aplicación se encuentra expuesta a una amenaza. Un ejemplo de este tipo de amenaza es la palabra clave *Open* que se utiliza para abrir un archivo. Debe examinarse el código que circunda la instrucción *Open* para ver si se está permitiendo el paso de entradas no válidas a dicha instrucción. Si no se valida la entrada efectuada por el usuario, cualquier atacante podría incluir rutas relativas de acceso a directorios, tales como "..\..', como parte de la entrada para abrir cualquier archivo contenido en su PC. Si la aplicación se ejecuta en el servidor, el atacante podría descubrir información sobre su equipo.

Cuando revisa el código, debe buscarse las palabras clave de la Tabla 6.5

PALABRA CLAVE *VISUAL BASIC .NET*	DESCRIPCIÓN
ChDrive, ChDir, MkDir, RmDir	Si se permite que el usuario realice alguna entrada mediante estos comandos, un atacante podría obtener información sobre la estructura de directorios en la que se ejecuta la aplicación o podría agregar o eliminar directorios en el equipo.
Create	Esta palabra clave tiene numerosos usos asociados; los más populares son la creación de archivos, claves del registro y procesos.
Declare	Habrá que inspeccionar con cuidado todas las funciones de *Windows* API declaradas mediante instrucciones Declare que utilice en su aplicación. En particular, deberá buscar las funciones API que haya declarado en los lugares en los que se pase un búfer de cadena para evitar la ejecución de desbordamientos de búfer.
Delete	En general, se trata de un comando muy peligroso que se utiliza en numerosas situaciones como, por ejemplo, para manipular archivos, bases de datos y claves del registro.
Dir	El comando *Dir* sirve para listar los archivos y directorios contenidos en el equipo. Compruebe su empleo para asegurarse de que el atacante no podrá acceder a los detalles de los archivos y directorios. Por ejemplo, si lista los archivos contenidos en un directorio, agrega los archivos a una colección y devuelve la colección como parte de un método *Public* dentro de una biblioteca de clases, el atacante podría llamar a la función *Public* de una forma inesperada para obtener información sobre el servidor en el que se está ejecutando la aplicación.
Environment	Cuando el código contiene la palabra clave *Environment* es una indicación de que el código puede estar utilizando variables de entorno como entradas. Deberá pensar en las posibles amenazas relacionadas con los cambios de los valores asignados a las variables de entorno realizados por un atacante que emplee valores inesperados. Deberá aplicar las técnicas de validación de la entrada a las entradas realizadas utilizando variables de entorno, tal y como se analizó en el Capítulo 5.
Execute	Execute se utiliza en general para llevar a cabo consultas SQL. Compruebe las cadenas SQL para validar las entradas efectuadas por los usuarios y que se pasan como parte de dichas cadenas SQL.
Kill	Esta instrucción sirve para eliminar archivos. Como indica su nombre, resulta inherentemente peligroso que esta instrucción aparezca en el código. Compruebe todas las llamadas a *Kill* para asegurarse de que un usuario no pueda suministrar una entrada.
Open	La instrucción *Open* se utiliza para abrir distintos recursos como, por ejemplo, un archivo o una conexión con una base de datos. Verifique todas las instrucciones *Open* contenidas en la aplicación que utilicen entradas para abrir un archivo o un dispositivo; también deberá analizar la forma en que las instrucciones relacionadas (por ejemplo, *Read* y *Write*), que aparecen después de la instrucción *Open*, utilizan los datos leídos o

	escritos de/a un archivo o dispositivo. En particular, deberá comprobar que un atacante no puede suministrar una entrada que se utilice como parámetro de la instrucción *Open*. También deberá verificar que una instrucción *Open* no puede llamar a varios recursos o al mismo recurso de forma repetida para llevar a cabo un ataque de denegación de servicio. Tendría que existir una instrucción *Close* asociada a cada instrucción *Open* que se ejecutara siempre que se produjera un error.
Params, QueryString, Form	Las colecciones *Params, QueryString y Form* están relacionadas con el objeto *Request* de *ASP.NET*. Estas colecciones representan un conjunto de entradas no verificadas para la aplicación. Compruebe que valida todos los valores de la colección antes de utilizarlos.
Public	Esta palabra clave le obligará a buscar todos los métodos expuestos de manera pública. Cualquier método cuyo objetivo sea exclusivamente interno deberá ser cambiado por *Friend* o *Private* para reducir la superficie de ataque de la aplicación.
Reflection, Type, Assembly	Estos comandos se utilizan en general para explorar una aplicación y buscar los tipos y atributos que expone. Una amenaza potencial es que si expone un miembro de clase públicamente accesible (o un valor proporcionado por una función) de un tipo tal como *Assembly*, *Type* o *Reflection*, cualquier atacante podría utilizar el miembro de clase públicamente accesible (o el valor proporcionado por la función) para conocer más detalles sobre la estructura de su aplicación. Es un ejemplo de un problema en el que la aplicación eleva los privilegios del llamador proporcionándole información a la que no debería tener acceso.
Shell, Start	Cualquiera de estos comandos indica, en general, que se está ejecutando una nueva aplicación o subproceso. Verifique que ninguna persona puede suministrar una entrada cuyo objetivo sea ejecutar aplicaciones de forma arbitraria tales como *Format* o *Delete*. Si se utiliza *Start* con el objeto *Thread*, revisar cuidadosamente el código para localizar problemas relacionados con los subprocesos. En particular, comprobar el código para asegurarse de que un atacante no puede crear y ejecutar un número ilimitado de subprocesos. También debe comprobarse que un atacante no puede manipular la aplicación utilizando varios clientes o peticiones que causen una condición de bloqueo de subprocesos, en el que cada subproceso esté esperando a que otro libere el bloqueo. Cada uno de estos problemas puede conducir a un ataque de denegación de servicio (DoS).

Tabla 6.5. Palabras clave

ASIGNAR PRIORIDADES A LAS AMENAZAS

Una vez que se haya creado una lista exhaustiva de amenazas, tiene que asignarse una prioridad a las mismas para determinar cuáles tendrán que anular. Tiene que asignarse prioridades a las amenazas teniendo en cuenta la prioridad de que se produzca un ataque, combinado con la severidad de los daños asociados a dicha amenaza. Además, tiene que especificarse el grado de vulnerabilidad de la aplicación con respecto a dicha amenaza.

Si se considera la lista de las amenazas mostradas previamente, y se asocia una prioridad y clasifiquemos las amenazas por nivel de prioridad, Por ejemplo, puede crearse un esquema amenaza/prioridad utilizando los valores del 1 al 3, donde las amenazas de prioridad 1 son las más peligrosas y que deberá anular a toda costa. Es posible definir los tres niveles de amenazas considerando su prioridad y el significado:

Prioridad	Significado
1	La probabilidad de que ocurra es elevada, así como su gravedad y grado de vulnerabilidad de la aplicación. Son las amenazas que habrá que resolver.
2	La probabilidad de que ocurra es media, así como su gravedad y grado de vulnerabilidad de la aplicación. También debería anular estas amenazas.
3	La probabilidad de que ocurra es baja, así como su gravedad y grado de vulnerabilidad de la aplicación. Podrá distribuir su aplicación sin anular estas amenazas.

La Tabla 6.6, muestra la forma en que pueden tabularse las amenazas, correspondientes una aplicación que trabaje con usuarios y contraseñas en red, que se pueden encontrar tras tormentas de ideas y los valores de prioridad que ha asignado a cada una de ellas. Además, la tabla contiene también algunos comentarios sobre las amenazas que se ciernen sobre el sistema. Debe proporcionarse comentarios de índole similar cuando se realice el análisis de amenazas de la aplicación. Debe incluirse comentarios (tal y como se muestra en la tabla 6.6) que justifiquen el nivel de prioridad asignado a cada una de las amenazas.

AMENAZA	PRIO-RIDAD	COMENTARIO
Se puede obtener la contraseña de un usuario, permitiendo que el atacante inicie una sesión como dicho usuario.	1	En el estado actual, el sistema de administración de empleados utiliza contraseñas débiles. De hecho, las contraseñas coinciden con los nombres de los usuarios. Se debería modificar la aplicación para que permitiera el empleo de una directiva de contraseñas fuertes.
Un atacante puede interceptar el nombre de usuario y la contraseña enviada a través de la conexión HTTP que une al cliente con el servidor, permitiendo que el atacante se conecte como dicho usuario.	1	La aplicación del sistema de administración de empleados será vulnerable a este tipo de ataque si el nombre de usuario y la contraseña se envían a través de un canal HTTP sin cifrar, como sucede ahora. Se debería modificar la aplicación para que utilizara *Secure Sockets Layer* (SSL) a fin de proteger el nombre de usuario y la contraseña en su camino desde el cliente al servidor.
El atacante podría intentar la introducción de instrucciones SQL o *VBScript* en los campos nombre de usuario y contraseña para provocar un ataque de inyección SQL o de	1	Basándonos en la forma en que la aplicación del sistema de administración de empleados utiliza el nombre de usuario que se ha pasado para buscar la contraseña en la base de datos utilizando una

programación de sitio cruzado. Por ejemplo, si el nombre de usuario se muestra como parte de un mensaje de error, podría provocar un ataque de programación de sitio cruzado.		instrucción SQL, existe la posibilidad de que la aplicación sea susceptible de sufrir un ataque de inyección SQL. Consulte el Capítulo 6 para ver los métodos disponibles para mitigar esta amenaza.
El atacante podría intentar conectarse directamente al servidor Web IIS que hospeda la aplicación de administración de empleados escrita en ASP.NET para acceder a los archivos, tales como el archivo *Web.Config*, o para acceder directamente a la base de datos *back-end*.	1	Dependiendo de cómo se haya configurado el servidor que ejecuta el sistema de administración de empleados, tal vez tenga que bloquear el servidor JIS.
Si se utiliza una *cookie* para iniciar una sesión en el sistema de administración de empleados (como sucede cuando el usuario pulse el botón "Iniciar mi sesión de forma automática') el atacante puede intentar interceptar la *cookie* para iniciar una sesión como el usuario que tenga asignada dicha *cookie*. De esta forma, el atacante podrá iniciar una sesión como dicho usuario sin tener que conocer ni el nombre ni la contraseña.	1	Aunque ASP.NET cifra las *cookies* almacenadas en el equipo cliente, un atacante podría utilizar la *cookie* cifrada (sin necesidad de conocer su contenido) para iniciar una sesión como el usuario que tiene asignada dicha *cookie*. Para ayudar a impedir que un atacante intercepte una *cookie*, deberá pasarla utilizando una conexión SSL.
Un atacante podría ver el contenido del archivo HTML en el explorador con el fin de localizar comentarios que revelaran la estructura de directorios de la aplicación o de la base de datos que se esté utilizando.	2	En la actualizar, la aplicación del sistema de administración de empleados no incluye en los comentarios ninguna información que pueda ser de utilidad a un atacante. Deberá revisar este escenario para el caso de que se agreguen nuevos comentarios a la página de inicio de sesión (o a cualquier otra página Web que se muestre al usuario).
El atacante podría intentar iniciar la sesión utilizando la cuenta de un usuario más veces de las permitidas por las opciones de seguridad del sistema, lo que impediría que el auténtico usuario iniciara la sesión durante el periodo de bloqueo definido para dicha cuenta.	3	La aplicación del sistema de administración de empleados no es vulnerable a este tipo de ataque porque no bloquea las cuentas de los usuarios.
El atacante podría intentar deshabilitar los archivos de comandos del lado del cliente para impedir la validación de las entradas.	3	En el caso del escenario de inicio de sesión, la aplicación del sistema de administración de empleados no valida nada en el equipo cliente.

Tabla 6.6. Prioridad de Amenazas

6.3.3.1.1. RESPONDER A LAS AMENAZAS

Una vez que se haya completado el proceso de generación de una lista de las amenazas y que se haya asignado una prioridad, tiene que darse respuesta a cada amenaza. Una buena forma de analizar estos problemas es registrar un error para cada uno de estos problemas y definir la prioridad del error para que coincida con la prioridad que asignada en el análisis de amenazas. Para cada amenaza, contar con las siguientes opciones:

- ☑ No hacer nada y dejar que su aplicación siga estando expuesta a la amenaza que ha identificado.
- ☑ Agregar nuevas funciones o modificar el diseño de la aplicación para mitigar la amenaza.
- ☑ Recortar las funciones no críticas que exponen a su aplicación a un riesgo de ataque considerable.

Cuando se tome decisiones sobre cómo puede responder de la mejor manera a cada amenaza, debe preguntarse a sí mismo: 'Si soy atacado, ¿estaré a gusto con el resultado?". Si la respuesta es no, debe resolverse el problema o eliminar las funciones que no sean críticas para eliminar la vulnerabilidad. Para cada una de las amenazas de seguridad a las que se da solución introduciendo correcciones en su código, debe agregarse un comentario que sirva como referencia para otros (y para uno mismo cuando vuelva a revisar su código en el futuro) con el fin de resaltar el cambio realizado. Esto ayuda a garantizar que no va a volverse a introducir la vulnerabilidad tiempo después. Por ejemplo, si se realizan otras modificaciones en la misma área de código, el comentario servirá de recordatorio para verificar que no se ha vuelto a exponer a la aplicación a la amenaza original. Es recomendable además hacer que alguien mas revise las correcciones antes de darlas por buenas. El proceso de revisión y explicación de la corrección a otra persona puede abrir mentes y hacer caer en cuenta de otros problemas que la corrección inicial no ha tenido en cuenta.

6.3.4. ATAQUES A LAS APLICACIONES

Sin quitar importancia a los *Hackers* y *Crackers*, el peligro está en los usuarios autorizados, y en el desconocimiento de las vulnerabilidades de nuestro sistema de información.

Al incrementarse la presencia empresarial en-línea, las amenazas potenciales aumentan. Por esto, se han desarrollado sistemas de seguridad, conocidos como contrafuegos o *firewalls*, para proteger la red interna de ataques externos. Los ataques producidos en redes compartidas o en sistemas y aplicaciones basados en Internet se ejecutan a dos niveles distintos: sistema y aplicación. Este subtema se centra en los ataques a nivel aplicación y, específicamente, en las áreas en las que las aplicaciones de Visual Basic .NET resultan vulnerables frente a ataques.

6.3.4.1. ATAQUES: MÉTODOS, HERRAMIENTAS Y TÉCNICAS DE RESPUESTA

☑ ATAQUES DE DENEGACIÓN DE SERVICIO (DOS)

La intención de los ataques de Denegación de servicio (DoS) es forzar el bloqueo parcial o completo de una aplicación. Los ataques DoS no intentan destruir los datos de la aplicación, si no que su objetivo es hacer que la aplicación deje de proporcionar los servicios que tuviera encomendados. Los ataques DoS suelen ejecutarse contra redes y aplicaciones basadas en redes.

Las aplicaciones basadas en Visual Basic .NET, las aplicaciones Web basadas en ASP.NET y las aplicaciones de servicio Web que se ejecutan en Internet son las más vulnerables a los ataques DoS. Los ataques DoS pueden ser de varios tipos. El código de Visual Basic .NET suele padecer con más facilidad ataques basados en los bloqueos de la aplicación y en el acaparamiento de memoria, de CPU y de otros recursos.

- **Técnicas de defensa para los ataques DoS**

Los ataques DoS son los que tienen una defensa más complicada. Con frecuencia resulta difícil identificar en qué lugar es vulnerable su aplicación porque no resulta fácil reproducir las condiciones necesarias para que se produzca un ataque DoS.

Cuando se trata de varios millones de computadoras que accedan simultáneamente a su aplicación Web, no puede evitarse el ataque, si bien podrá intentar mitigar los efectos negativos. Por ejemplo, si la aplicación Web es capaz de detectar que se realizan más peticiones de las que se puedan controlar de forma razonable, la aplicación puede ser capaz de responder con una página HTML distinta notificando a los usuarios que el sitio Web está experimentando un volumen de tráfico superior al normal e indicando a los visitantes que vuelvan a intentar la conexión un poco más tarde. Esta solución resulta más favorable que el hecho de que la aplicación (o el sitio Web mostrado por la aplicación) parezca encontrarse bloqueada porque la página Web no se muestre en una cantidad de tiempo razonable.

Si la aplicación no responde bien ante una demanda inusualmente elevada de usuarios (sin que tenga que tratarse de un ataque), deberá intentar trabajar en el sentido de conseguir que la aplicación sea capaz de controlar los picos de carga. La Tabla de abajo muestra algunas formas de enfrentarse a cada uno de los tipos de ataque DoS.

FORMA DE ATAQUE	EJEMPLOS	TÉCNICA DEFENSIVA
Fallo de la aplicación	Pasar datos no esperados a una aplicación con la intención de hacerla fallar.	Escribir un código sólido que no conduzca a fallos ¿Cómo se puede llevar a cabo este objetivo? Específicamente, necesita validar todas las entradas, tal y como se analiza en el Capítulo anterior; utilice los controladores *Try ... Catch* en la forma apropiada para capturar y controlar todas las excepciones que se produzcan, pruebe el código y atáquelo y realice una auditoría de seguridad del código de su aplicación y del diseño de la misma.
Acaparamiento de memoria	Pasar largas cantidades de datos a una aplicación para saturar la memoria del equipo en el que se ejecuta dicha aplicación.	Limite el tamaño de la entrada y rechace toda entrada repetitiva. Escriba los datos almacenados en la memoria en un archivo o base de datos para ahorrar memoria. Libere juiciosamente ciertos objetos (por ejemplo, las matrices) que consuman gran cantidad de memoria, asignando el valor *Nothing* a la variable objeto cuando los datos dejen de ser necesarios.

Acaparamiento de CPU	Pasar datos a una aplicación para provocar que la aplicación entre en un bucle infinito o que provoque que la aplicación realice un uso intensivo de CPU durante un periodo de tiempo excesivo.	Identifique y resuelva todos los casos presentes en el código en los que un bucle o una llamada recursiva a una función pueda conducir a un bucle infinito cuando se reciba una entrada inesperada.
Acaparamiento de recursos	Intentar agotar un recurso limitado como, por ejemplo, el espacio disponible en disco. Una forma de realizar esta tarea es pasar grandes cantidades de datos a una aplicación y, posteriormente, guardarlos en el disco y forzar una condición de poco espacio en disco o sin espacio en disco.	Controle apropiadamente los archivos, conexiones a bases de datos y referencias a objetos que la aplicación ya no necesite. Revise el código para garantizar que las entradas no verificadas no van a provocar que la aplicación asigne una cantidad de recursos innecesaria o un recurso con un tamaño excesivo.
Acaparamiento del ancho de banda de la red	Reclutar la ayuda de muchas máquinas para efectuar peticiones repetitivas y simultáneas a una aplicación o sistema con la intención de saturar al servidor de la aplicación con las solicitudes de la red.	
Fallo del sistema	Pasar datos no esperados a una aplicación o directamente al sistema (o a la aplicación del sistema) a través de un punto de entrada (por ejemplo, un puerto de la red) con la intención de provocar un fallo en el sistema.	

Tabla 6.7. Ataques y defensas

- **Defensa contra ataques DoS de acaparamiento de memoria y recursos**

Como ejemplo sencillo de un código de Visual Basic .NET que puede sufrir un ataque DoS por acaparamiento de memoria, suponga que se dispone de una aplicación de biblioteca de clases con una clase *Public* denominada *Productos* que contiene un método *Public* denominado *AgregarProducto*, con la siguiente estructura[17]:

[17] El ejemplo de código anterior sólo tiene carácter ilustrativo. NO INTENTE EJECUTAR EL CÓDIGO EN LA PC. Si crea y ejecuta esta aplicación, la máquina puede llegar a tener un comportamiento errático.

```
  Public Class Productos
      'Agregar sentencias Imports System.Collections.Specialized en la parte
      'superior del módulo de código
    Private m_NombresProductos As New Collection
Public Sub AgregarProducto(ByVal NombresProductos As String)
       m_NombresProductos.Add(NombresProductos)
End Sub End Class
```

Código Fuente 6.A.

Supongase que un atacante crea una aplicación cliente que llama a la función *AgregarProducto* de forma repetida, pasando cadenas extraordinariamente largas como:

```
  'Forzar un ataque de acaparamiento de memoria
  Dim CadenaMuyGrande As String
  Dim productos As New Productos
  'Bucle infinito
  Do While True
      'Asignar una cadena de un millón de caracteres
      CadenaMuyGrande = New String(X, 1000000)
      productos.AddProduct(CadenaMuyGrande)
  Loop
```

Código Fuente 6.B.

Como el método *AgregarProducto* carece de todo tipo de validación de datos de entrada, y ni siquiera dispone de una comprobación para ver si el *NombreProducto* es único, el atacante forzará a la aplicación servidora a consumir toda la memoria disponible en el servidor. Sin embargo, incluso antes de que se consuma toda la memoria, el servidor se comportará de forma lenta y será incapaz de responder a ninguna nueva petición. Como otro ejemplo, ¿qué pasaría si la función *AgregarProduct* de la biblioteca de clases agregara la cadena a un archivo, tal como se muestra a continuación?:

```
  Public Sud AgregarProduct(ByVal NombreProducto As String)
  Dic hFile As Integer FreeFile()
  'Agregar sentencias Microsoft.VisualBasic.Compatibility en la parte superior del módulo
de código
  FileOpen(hFile, VB6.GetPath & \ NombreProducto.Txt",OpenMode.Append)
  PrintLine(hFile, NombreProducto)
  FileClose(hFile)
  End Sub
```

Código Fuente 6.C.

Este código atacante producirá un ataque de acaparamiento de recursos o de espacio en disco cuyo resultado final será el mismo que en el caso del ataque de acaparamiento de memoria: el servidor dejará de responder.

La mejor defensa contra los ataques de acaparamiento de memoria y de espacio en disco es limitar el tamaño de la entrada permitida. En el ejemplo anterior, debe comprobarse siempre el tamaño del parámetro de entrada *ProductName*. Además, la subrutina *AddProduct* debe impedir la

introducción de nombres duplicados. Por ejemplo, el parámetro *ProductName* debe estar limitado a un tamaño razonable, tal como 20 caracteres. Para impedir las cadenas duplicadas, antes de agregar una cadena a la colección debe utilizarse el método *Contains* para confirmar que la colección no contiene ya la cadena que está a punto de añadir.

El código mostrado a continuación demuestra que la validación de la entrada (comprobando que la cadena de entrada tiene un tamaño razonable) y el rechazo de los nombres duplicados puede ayudarle a impedir un ataque DoS o, al menos, obligar al atacante a trabajar más duro y mejor para poder lanzar un ataque[18] de este tipo.

```
    'Agregar sentencias Imports System.Collections.Specialized
    'en la parte superior del modulo de código
    Private m_NombresProductos As New StringCollection
    Public Sub AgregarProducto(ByVal NombresProductos As String)
        Const MAXLENGTH_NOMBREPRODUCTO = 20
        If NombresProductos.Length <= MAXLENGTH_NOMBREPRODUCTO AndAlso Not
m_NombresProductos.Contains(NombresProductos) Then
            m_NombresProductos.Add(NombresProductos)
        Else
            MsgBox("Se lanzará una Excepción")
            Throw New ArgumentException("Nombre de producto invalido")
        End If
      End Sub
```

Código Fuente 6.D.

☑ ATAQUES BASADOS EN ARCHIVO O EN DIRECTORIO

Si se utiliza la entrada de usuario como la base para formar el nombre de un archivo o de un directorio con el fin de abrir dicho archivo, cualquier atacante podrá manipular la entrada para abrir el archivo en una ubicación no deseada. Supongase que crea la siguiente función *Public* en una aplicación servidora con el objetivo de guardar las opciones del usuario en un archivo. La intención es guardar el archivo en la misma ubicación que la aplicación, utilizando la función *Application.StartupPath*[19].

```
  Public Sub SaveSettings(ByVal NombreUsuario As String, ByVal Settings As String)
  Dim hFile As Integer FreeFile()
  Dim Filename As String = Appication.StartupPath & "\" & NombreUsuario
      FileOpen(hFile, Filename, OpenMode.Output)
      PrintLine(hFile, Settings)
      FileClose(hFile)
  End Sub
```

Código Fuente 6.E.

[18] Para impedir que un atacante pueda llamar a un método *Public* contenido en una clase servidora, debe utilizarse también una estrategia profundamente defensiva, tal como utilizar técnicas de seguridad basada en roles para verificar que el usuario que realiza la llamada ha sido autenticado y tiene asignados los permisos basados en roles necesarios para llamar a la función.

[19] En el caso de las aplicaciones Web basadas en ASP.NET, utilice Server.Map Path(Request.ApplicationPath) en lugar de Application.StartupPath.

Si un atacante pudiera llamar a esta función pasando una ruta de directorios inesperada, tal como una ruta que contenga dos puntos que representen al directorio padre (..), el atacante puede provocar que la función *SaveSettings* cree un archivo en cualquier parte de la unidad de disco actual, sobrescribiendo cualquier archivo que tenga el mismo nombre. Por ejemplo, si el atacante pasara un nombre del tipo ..\..\..\..\..\..\..\Windows\NotePad.Exe, el método *SaveSettings* sobrescribirá la aplicación Bloc de notas siempre que se den las siguientes circunstancias: que el sistema operativo se haya instalado en la misma unidad que la aplicación, que Windows se haya instalado en un directorio denominado Windows colgando directamente del directorio raíz y que el atacante haya introducido los comandos (..) necesarios para alcanzar el directorio raíz de la unidad de disco.

El atacante se ve ayudado por el hecho de que el sistema operativo trata la superabundancia de especificadores de directorios padre (..) como una referencia al directorio raíz. El atacante puede limitarse a proporcionar una gran cantidad de especificadores (..) para garantizar que se ha alcanzado de sobra el directorio raíz, seguido por el nombre del directorio Windows. En el ejemplo que acabamos de ver, el atacante ha pasado..\..\..\..\..\.. como parte del valor del nombre suponiendo que la aplicación se encuentra anidada, como máximo, seis niveles de directorios por debajo del directorio raíz.

Imagine, además, que en lugar de Notepad.Exe el atacante suministra el nombre de un archivo crítico del sistema operativo o un archivo personal (ubicado en cualquier otro directorio) que quedarán sobrescritos con el contenido de la cadena de opciones. Además, ¿qué pasaría si la aplicación proporciona de forma no intencionada al atacante la capacidad no sólo de guardar el archivo donde el atacante desee sino para determinar el contenido del archivo? El atacante podría sobrescribir un .EXE, tal como NOTEPAD.EXE, con el propio archivo denominado NOTEPAD.EXE conteniendo su propio código ejecutable personalizado (por ejemplo, código que llame a FORMAT.COM con el fin de borrar el contenido de una de sus unidades de disco duro). Los cambios que el atacante haya podido efectuar en el archivo NOTEPAD.EXE permanecerán dormidos hasta que otro usuario lo ejecute Borrará todos los datos contenidos en una de sus unidades de disco duro. Este hecho le hará pensar dos veces antes de permitir el empleo de la entrada del usuario como base para rescribir un archivo y su contenido.

- **Técnicas de defensa para los ataques basados en archivo o en directorio**

Se puede utilizar la siguiente técnica defensiva para frustrar un ataque basado en directorio o archivo.

- Forzar el empleo de nombres canónicos de archivo

Los nombres canónicos de archivo o de directorio son nombres que cumplen una definición estándar. Por ejemplo, la ruta de acceso completa y el nombre de archivo del formulario "C:\ARCHI VOS DE PROGRAMA\WINDOWS NT\ACCESORIOS\WORDPAD.EXE es la representación canónica del nombre de archivo WORDPAD.EXE. Sin embargo, existen otras muchas formas de representar a WORDPAD.EXE, tal y como se muestra en la Tabla 6.8, que se consideran representaciones no canónicas o no estándar del mismo nombre.

NOMBRES DE ARCHIVO NO CANÓNICOS	NOTAS
C:\ARCHIVOS DE PROGRAMA\..\ ARCHIVOS DE PROGRAMA\WINDOWS NT\ACCESORIOS\WORDPAD.EXE	Utilice el especificador del directorio padre (..) para navegar hacia arriba y volver al directorio anterior.
\ARCHI VOS DE PROGRAMA\WINDOWS NT\ACCESORIOS\WORDPAD.EXE	Ruta de acceso relativa al directorio raíz, tal como C:\.
C:\ARCHIVOS DE PROGRAMA\WINDOWS NT\ACCESORIOS\WORDPAD.EXE.	Punto tras el nombre del archivo, lo que significa que no existe otra extensión adicional.
C:\\ARCHIVOS DE PROGRAMA\\\\WINDOWS NT \\\\ACCESORIOS\\\WORDPAD.EXE	Uso excesivo de barras invertidas; el sistema operativo ignora las barras innecesarias.
WORDPAD.EXE	Nombre relativo de archivo, que supone que la unidad y el directorio actuales contienen a WORDPAD.

Tabla 6.8. Nombres Canónicos

El código es vulnerable a ataques de archivo o de directorio sino verifica los errores de canonicalización. Por ejemplo, si el código se limita a comprobar si la primera parte del nombre de archivo que se ha pasado coincide con la ubicación de la aplicación, y se supone que la aplicación se encuentra en el directorio C:\MiAplicacion, el código no vera nada extraño cuando se le introduzca una ruta de directorios similar a C:\MiAplicacion\..\Windows\System32. Sin embargo, esta ruta de acceso puede suponer un grave problema, ya que el código que ha efectuado la llamada tiene ahora acceso completo al directorio C:\WINDOWS\SYSTEM32. El llamador puede ahora examinar, renombrar o eliminar cualquier archivo importante del sistema.

Para solventar este problema, deben convertirse los nombres de archivo y de directorio a un formato estándar o canónico. Al cambiar los nombres del archivo y del directorio a un formato canónico podrá realizarse una comparación punto por punto de la ubicación de los dos archivos o directorios. Si se convirtiera el nombre del directorio C:\MiAplicacion \..\Windows \System32 a formato canónico, el nombre resultante del directorio será C:\Windows\System32, que podrá compararse con el nombre del directorio que la aplicación espera utilizar; claramente C:\Windows\System32 no es el directorio C:\MiAplicacion que debe utilizar la aplicación.

Visual Basic .NET dispone de una función que permite convertir los nombres de archivos a su formato canónico. Se trata del método *Path.GetFullPath*, que proporciona la ruta de acceso absoluta del archivo pasado como argumento.

Si, por ejemplo, se desea garantizar que los archivos de datos que escribe van a tener la misma ubicación que la aplicación, puede llamar al método *Path.GetFullPath* (pasándole el nombre del archivo que desea guardar), pasar el resultado del método *Path.GetFullPath* al método

Path.GetDirectoryName para obtener la ubicación del directorio del archivo y comparar el resultado con la ruta proporcionada por *Application.StartupPath* (para las aplicaciones de Windows) o *Server.MapPath*(*Request.ApplicationPath*) (para las aplicaciones Web). Si las dos coinciden, se tiene garantizado que el archivo se guardará en la misma ubicación que la aplicación. El código mostrado a continuación muestra la forma en la que puede definirse una función denominada *IsValidPath* cuyo objetivo es verificar que al juntar el nombre de un archivo y la ruta de acceso que ha obtenido a partir de una o más cadenas, obtendrá la ruta de acceso de la aplicación:

```
   Private Function IsValidPath(ByVal strNombreArchivo As String)As Boolean
       'TODO: Incluir instrucciones Imports Systern.IO en la parte
       'superior del módulo de código
   Dim strNombreArchivoCanonico As String = Path GetFul 1 Path (strNombreArchivo)
   If Path.GetDirectoryName(strNombreArchivoCanonico).ToLower =_
Application.StartupPath.ToLower Then
           Return True
       End If
       Return False
   End Function
```

Código Fuente 6.F.

Como medida adicional, además de verificar el nombre de la ruta de acceso también puede verificarse que el valor *NombreUsuario* no contiene caracteres que no sean válidos utilizando la clase *Regex* de expresiones regulares para verificar el contenido del nombre de usuario al compararlo contra un conjunto de caracteres válidos. Observe que en este caso hipotético (suponiendo que los nombres de usuario sólo puedan tener caracteres alfabéticos) debe restringirse el conjunto de caracteres válidos para no incluir símbolos especiales en el nombre de archivo tales como el punto, la barra invertida o el signo de pesos. Por ejemplo, podrá utilizar una expresión regular tal como [A-Za-z]+$ para asegurarse de que el nombre de usuario contiene sólo letras y no símbolos, espacios u otros caracteres que pudieran provocar problemas si se utilizaran como parte del nombre de un archivo. Si los nombres de usuario permitidos por la aplicación pueden contener números o caracteres de puntuación, la expresión regular sugerida anteriormente sería demasiado restrictiva. El objetivo ahora es añadir la lógica necesaria (expresiones regulares u otras formas de comprobación de cadenas) para verificar si el nombre de usuario contiene sólo caracteres pertenecientes al conjunto de caracteres permitidos y ninguno más. Si no conoce el tema de las expresiones regulares. Finalmente, debe comprobarse que el nombre[20] de usuario tiene una longitud razonable para evitar un ataque DoS.

[20] Si en el ejemplo mostrado anteriormente el nombre de usuario coincidiera con un nombre de dispositivo, tal como AUX, CLOCK$, COM1-COM9, CON, LPT1- LPT9, NUL o PRN, cualquier intento de abrir un archivo con ese nombre fallaría provocándose una excepción. El motivo de este comportamiento es que el sistema operativo tiene reservados estos nombres especiales de dispositivos y trata los nombres de dispositivos como parte del sistema de archivos. Por fortuna, las funciones de EIS de archivos de Visual Basic .NET le impedirán abrir un dispositivo. Las funciones de EIS de archivos iniciarán una excepción. Aunque las funciones de E/S de archivo detectan cuando se intenta abrir un dispositivo, deberá agregar código para verificar explícitamente si el nombre del usuario coincide exactamente con el nombre de un dispositivo (utilizando una comparación que no distinga en el uso de mayúsculas y minúsculas). Se trata de un ejemplo de empleo de la técnica de defensa en profundidad cuyo objetivo es proteger al máximo la aplicación.

Podrá agregar un sistema de comprobación de errores al método *SalvarSettings* en la forma siguiente:

```
Public Sub SalvarSettings(ByVal NombreUsuario As String, ByVal Settings As String)
    Const MAX_USERNAME_LENGTH As Integer = 15
    Dim hFile As Integer = FreeFile()
Dim Filename As String = Application.StartupPath & "\" & NombreUsuario()
It UserName.Length <= MAX_USERNAME_LENGTH AndAlso IsValidPath(Filename) Then
    FileOpen(hFile, Filenanie.OpenMode.Output)
    PrintLine(hFile, Settings)
    FileClose(hFile)
Else
    Throw New ArgumentException("Nombre del archivo no válido")
End If
End Sub
```

Código Fuente 6.G.

☑ ATAQUES DE INYECCIÓN SQL

SQL son las siglas de *Structured Query Language*, o Lenguaje de consulta estructurado. Se trata de un lenguaje especializado en el proceso de los datos contenidos en una base de datos relacional. SQL es un lenguaje, al igual que Visual Basic .NET, con su propia sintaxis y posibilidades.

Una forma frecuente de utilizar SQL desde la aplicación Visual Basic .NET es incluir comandos SQL en una cadena y, posteriormente, ejecutarla utilizando un objeto de base de datos tal como un objeto de comandos ADO.NET. Su aplicación será vulnerable cuando admita entradas de los usuarios sin verificarlas previamente para alimentar a la cadena SQL que haya desarrollado. Supongase, por ejemplo, la siguiente instrucción SQL:

```
    Dim sql As String
    'Suponga que strLastName se pasa como un parámetro de cadena
    sql = "SELECT * FROM Authors WHERE LastName = '" & strLastName & "'"
```

Código Fuente 6.H.

La intención de la anterior instrucción SQL es obtener todos los autores cuyo apellido sea idéntico al especificado por el parámetro *strLastName*. Sin embargo, la anterior instrucción SQL realiza una hipótesis de gran importancia sobre los contenidos de *strLastName*, en concreto que el parámetro contiene un apellido válido tal como *Theroux* o *Hemrningway*. Pero ¿qué pasaría si el atacante pasara la siguiente cadena como parámetro *strLastName*?

```
Hemmingway' DELETE FROM Authors WHERE LastName = 'Theroux
```

En este ejemplo, el atacante está aprovechándose de la rica naturaleza de SQL terminando la consulta SELECT con "*Hemmingway*" e inyectando un comando SQL adicional con el objetivo de borrar todos los autores cuyo apellido sea *Theroux*. Este es un clásico[21] ejemplo de un ataque de inyección SQL.

[21] El ejemplo resulta eficaz cuando se lanza contra un sistema de bases de datos tal como Microsoft SQL Server 2000 que permite la ejecución de expresiones SQL múltiples como parte de una única cadena SQL. Este ataque no

La propia pantalla de inicio de sesión se presenta con frecuencia ante un atacante como una buena oportunidad para lanzar un ataque de inyección SQL. El motivo es que para realizar una validación del nombre de usuario se sentirá tentado de escribir código tal como el que se muestra a continuación.

```
    'Cadena G_CONNECTION tal y como se define en MainModule.vb
Const G_CONNECTIONSTRING As String = Provider=Microsott.Jet.OLEDB.4.O; & –
"User ID=Admin;Data Source=..\..\..\..\..\EmployeeDatabase.mdb; " & _
"Mode=Share Deny None;"
    Dim strSQL As String = "Select * from Employee where Username ='" & _
    strUsername & "'"
```

Código Fuente 6.I.

Este código resulta problemático porque acepta directamente, sin comprobar, la entrada realizada por el usuario como nombre de usuario y lo introduce en una instrucción SQL utilizando la siguiente línea de código:

```
Dim strSQL As String = "Select * from Employee where Username ='"&_
strUsername & "'"
```

¿Qué pasaría si un atacante introdujera el siguiente nombre de usuario?

```
    Bogus ' OR Username like '%%
```

¿Qué piensa de este nombre de usuario? El atacante suministra cualquier nombre (Bogus vale perfectamente) seguido por un apóstrofe para completar la cláusula del nombre de usuario en SQL, seguido por una instrucción OR condicional y una expresión adicional tal como *Usuario like* '%%' que obtendrá todos los empleados almacenados en la tabla *Empleado*. Observe que el apóstrofe final incluido en la asignación strSQL mostrada anteriormente, complete la expresión generando la siguiente instrucción SQL:

```
Select * from Empleado where Usuario = Bogus OR Usuario like '%%'
```

Este ataque funcionará en la mayoría de las bases de datos y no requiere ningún permiso especial de cuenta de base de datos, tal como un permiso administrativo.

Como se muestra, el nombre *Bogus* seguido de un apóstrofe se utiliza para completar la cláusula WHERE para el campo *Usuario*. El atacante puede incluir cualquier número de comandos a continuación, separados por caracteres punto y coma. En este caso, el atacante ha inyectado el comando *exec* para ejecutar un procedimiento almacenado de SQL Server denominado *xp_cmdshell* (un potente comando que le permitirá ejecutar cualquier aplicación contenida en el sistema). El atacante también ejecuta IISRESET.EXE (ubicado en el directorio de Windows) con el comando /STOP. Este comando forzará el apagado del servidor IIS en el que se está ejecutando el código. Se trata de una forma de ataque de Denegación de servicio (DoS). Los dos guiones finales situados al final del nombre de usuario introducido por el atacante

funcionaría si se ejecutara contra una base de datos de Microsoft Access porque Access no permite la ejecución de expresiones SQL múltiples contenidas dentro de la misma cadena SQL.

indican el inicio de un comentario SQL. Estos guiones son necesarios para comentar el apóstrofe situado al final de la cadena *strSQL*, mostrada en el ejemplo de código anterior. La cadena *strSQL* contendrá el siguiente texto cuando se ejecute:

```
Select * from Employee where Username =Bogus:
exec naster. .xpcmdshell IISRESET /STOP
```

La ejecución de un comando arbitrario utilizando *xp_cmdshell* sólo funcionará si el comando SQL se ejecuta desde la cuenta del administrador de SQL Server. Este factor se podría cumplir con facilidad si utiliza la cuenta sa (y la contraseña asociada) como parte de la cadena de conexión de ADO.NET.

- **Técnicas de defensa para los ataques de inyección SQL**

Podrá seguir ciertos procedimientos para impedir los ataques basados en inyección SQL. Como mínimo, debería probar a utilizar las dos primeras técnicas.

- Validación de los parámetros de entrada

Una técnica que puede utilizarse para impedir los ataques de inyección SQL es comprobar las cadenas de entrada para garantizar que no contienen caracteres innecesarios (en especial, caracteres de puntuación tales como apóstrofes, comillas, comas, puntos y comas y paréntesis) que se podrían utilizar para construir o modificar expresiones SQL. En el cuadro de diálogo de inicio de sesión correspondiente a la aplicación.

No existe ninguna necesidad de que el nombre de usuario (que se introduce en el cuadro de diálogo de inicio de sesión) contenga caracteres de puntuación que puedan permitir al atacante la construcción de instrucciones SQL dañinas. La aplicación requiere que los nombres de usuario estén formados exclusivamente por caracteres alfabéticos. Podrá construir una regla para permitir exclusivamente el empleo de caracteres alfabéticos, utilizando para ello una expresión regular similar a la siguiente:

```
    Dim fUsuarioValido As Boolean = False
    'Agregar instrucciones Imports System.Text.RegularExpressions en la
    'parte superior de módulo de código
    'Note: strüserName contiene el nombre de usuario introducido en el
    'cuadro de diálogo de inicio de sesión
fUsuarioValido(Regex.IsMatch(strUsuario, "^[A-Za-z]+$"))
```

Código Fuente 6.J.

Además, no existe ningún motivo para permitir que el usuario escriba un nombre de longitud infinita. En este caso, será suficiente con admitir nombres de usuario con una longitud máxima de 15 caracteres. Podrá modificar el ejemplo anterior añadiendo una comprobación para garantizar que la longitud es igual o inferior a 15 caracteres:

```
    Dim fUsuarioValido As Boolean = False
    Const MAX_LOG_USUARIO = 15
    'Agregar instrucciones Imports System.Text.RegularExpressions en la
    'parte superior de módulo de código
    'Note: strUserNanie contiene el nombre de usuario introducido en el
    'cuadro de diálogo de inicio de sesión
fUsuarioValido = strusuario.Length <= MAX_LOG_USUARIO _
AndAlso Regex.IsMatch(strusuario. "^[A-Za-z]+$")
```

Código Fuente 6.K.

- Empleo de consultas parametrizadas

Una técnica que funciona correctamente como defensa contra los ataques de inyección SQL es el empleo de consultas parametrizadas. Las consultas parametrizadas permiten definir la estructura de una consulta SQL y dejar variables pendientes de asignación para los parámetros que se introducirán en la consulta. Este hecho elimina la posibilidad de que el atacante pueda modificar la cadena SQL para pasar instrucciones SQL como parte de la entrada. Incluso si el atacante pasara instrucciones SQL, la entrada del atacante sería tratada como un parámetro para la consulta en lugar de poder modificar la propia cadena de consulta subyacente.

La siguiente línea de código es un ejemplo de una cadena de consulta parametrizada:

```
"Select * from Empleado where Usuario = @ParametrosUsuario"
```

Las consultas parametrizadas funcionan tanto con las bases de datos *Microsoft SQL Server* como con las de Microsoft Access.

- Cómo agregar un procedimiento almacenado para validar al usuario

Si la aplicación utiliza una base de datos Microsoft SQL Server (o cualquier otro sistema de base de datos que permita el empleo de procedimientos almacenados) como alternativa al empleo de consultas parameirizadas, podrá considerar la posibilidad de utilizar un procedimiento almacenado para permitir que su base de datos ejecute la consulta. En SQL podrá crear un procedimiento almacenado tal como el siguiente:

```
CREATE PROCEDURE EsUsuarioValido
@usuario VarChar(50)
AS
SELECT *
FROM empleado
WHERE @usuario = liserName
GO
```

Código Fuente 6.L.

Los procedimientos almacenados SQL Server constituyen una eficaz defensa contra los ataques de inyección SQL porque la definición de la instrucción SQL se encuentra congelada dentro del procedimiento almacenado y la entrada que introduzca el usuario no podrá modificarlo. Por ejemplo, los nombres de usuario que suministre el usuario se considerarán como parte del parámetro *Usuario* incluso aunque el Usuario que introduzca el usuario contenga instrucciones SQL.

Cuando se llama al procedimiento almacenado *IsValidUser*, éste devolverá la fila correspondiente al nombre del usuario introducido si es que éste existe y se encuentra en la base de datos. El código mostrado a continuación nos enseña que se puede declarar una cadena SQL para ejecutar un procedimiento almacenado basado en SQL Server denominado IsValidUser, que pasará a strUserName como argumento:

```
Dim strSQL As String = "EsUsuarioValido " & "'" & strUsuario & "'"
```

6.3.5. RESPALDOS

El Backup de archivos permite tener disponible e íntegro la información cuando sucedan los accidentes. Sin un backup, simplemente, es imposible volver la información al estado anterior al desastre.

Es necesario realizar un análisis Costo/Beneficio para determinar qué información será almacenada, los espacios de almacenamiento destinados a tal fin, la forma de realización, las estaciones de trabajo que cubrirá el backup, etc.

Para una correcta realización y seguridad de backups se deberán tener en cuenta estos puntos:

1. Se debe de contarse con un procedimiento de respaldo de los sistemas operativos y de la información de los usuarios, para poder reinstalar fácilmente en caso de sufrir un accidente.
2. Se debe determinarse el medio y las herramientas correctas para realizar las copias, basándose en análisis de espacios, tiempos de lectura/escritura, tipo de backup a realizar, etc.
3. El almacenamiento de los Backups debe realizarse en locales diferentes de donde reside la información primaria. De este modo se evita la pérdida si el desastre alcanza todo el edificio o local.
4. Se debe verificar, periódicamente, la integridad de los respaldos que se están almacenando. No hay que esperar hasta el momento en que se necesitan para darse cuenta de que están incompletos, dañados. mal almacenados, etc.
5. Se debe de contar con un procedimiento para garantizar la integridad física de los respaldos, en previsión de robo o destrucción.
6. Se debe contar con una política para garantizar la privacidad de la información que se respalda en medios de almacenamiento secundarios. Por ejemplo, la información se debe encriptar antes de respaldarse.
7. Se debe de contar con un procedimiento para borrar físicamente la información de los medios de almacenamiento, antes de desecharlos.
8. Mantener equipos de hardware, de características similares a los utilizados para el proceso normal, en condiciones para comenzar a procesar en caso de desastres físicos. Puede optarse por:
   - Modalidad Externa: otra organización tiene los equipos similares que brindan la seguridad de poder procesar la información, al ocurrir una contingencia, mientras se busca una solución definitiva al siniestro producido.

- Modalidad Interna: se tiene más de un local, en donde uno es espejo del otro en cuanto a equipamiento, características técnicas y capacidades físicas. Ambos son susceptibles de ser usados como equipos de emergencia.

Se debe asegurar reproducir toda la información necesaria para la posterior recuperación sin pasos secundarios. Por ejemplo, existe información que es función de otra.

6.3.6. PRUEBAS DEL CÓDIGO

La calidad del producto es un factor que recae directamente bajo control. Un diseñador de aplicaciones en Microsoft Visual Basic .NET, es responsable de escribir código de alta calidad y seguro, además de verificar que el código funciona tal y como se anuncia. Sólo probando el código puede verificarse que funciona correctamente. Incluso asignar un verificador para que pruebe el código no le libera de todas las responsabilidades desde el punto de vista de las pruebas y de la calidad del código.

La calidad del código implica la seguridad del mismo. La seguridad no es una característica que pueda omitir. Es un factor de gran importancia que reflejará la calidad global de la aplicación. Una aplicación que cuente con una estupenda interfaz de usuario y que sea fácil de utilizar, pero que pueda ser manipulada para enviar información sobre las tarjetas de crédito a un atacante, no será una aplicación de calidad.

Las pruebas son críticas para garantizar la alta calidad y la seguridad del código. Al realizar pruebas para garantizar la seguridad, deberá atacar a la aplicación igual que lo haría un atacante para ver cuál es la respuesta. Para ayudar a coordinar su ataque, necesitará elaborar un plan. Que determinen qué pruebas debe incluir en el plan y cómo atacar a la aplicación (ejecución de su plan de pruebas) para verificar que es segura.

Plan de ataque: el plan de pruebas

El plan deberá incluir los siguientes elementos:

Escenarios de empleo que cubran de manera exhaustiva todas las formas en las que se pueda utilizar la aplicación, tanto de manera apropiada como de manera inapropiada (esta última es la más importante).

- Resumen de las pruebas que deberá realizar clasificadas por orden de prioridad.
- Un calendario que indique el momento en que se ejecutarán las pruebas, cuánto tiempo va a durar esta fase y la forma en que el calendario de pruebas va a relacionarse con el calendario de desarrollo.
- Entornos de destino donde se ejecutarán las pruebas, tales como los sistemas operativos en los que el producto va a ejecutarse.

Una parte crítica del plan de pruebas es incluir pruebas para la seguridad. El plan debe tratar la seguridad como una característica especial que requiere unas pruebas específicas. Resaltar la seguridad en un plan de pruebas tendrá los siguientes beneficios:

• Un plan de pruebas orientado a garantizar la seguridad ayuda a mantener a raya a los *hackers*. El mejor momento para bloquear su aplicación es antes de que caiga en manos de estos individuos.

• Hacer hincapié en las pruebas relacionadas con la seguridad permite realizar estimaciones más fiables de cuánto tiempo le va a costar desarrollar y probar la aplicación. Estos pasos ayudan a evitar sorpresas en el camino, como los problemas serios de seguridad descubiertos demasiado tarde en el proceso de desarrollo.

• El plan de pruebas ayuda a detectar con rapidez aquellas características que sean fáciles y rápidas de implementar pero que resulten ser un serio problema desde el punto de vista de la seguridad. El plan de pruebas le da oportunidad de poder replantearse estas características.

• Un plan de pruebas orientado a garantizar la seguridad obliga a pensar las diversas formas en que alguien puede atacar a la aplicación. Da la oportunidad de realizar una tormenta de ideas, de organizar sus pensamientos y de plasmarlos sobre un papel.

Para crear pruebas centradas en la seguridad de la aplicación es necesario identificar, en primer lugar, las formas en que un atacante puede comprometer la aplicación. En otras palabras, es necesario enfrentarse con diversos escenarios basados en lo que un atacante puede hacer.

Cuando se trata de probar la seguridad, debería pensar en todas las cosas horribles que una persona puede hacer con su aplicación.

- Adopte el punto de vista del atacante

Una técnica que le ayudará a generar escenarios útiles es adoptar el punto de vista del atacante cuando se enfrente a la aplicación. Para un atacante, la aplicación no es una elegante interfaz de usuario, es todo lo que se encuentra detrás de la interfaz de usuario lo que le permitirá encontrar un método exclusivo y diabólico para causar estragos. Una de las primeras cosas que hará un atacante cuando instale su aplicación es obtener un inventario de todos los componentes instalados. El atacante analizará el lugar en el que se hayan instalado todos los componentes, las entradas del registro asociadas con dichos componentes, las funciones públicas expuestas por los componentes y todos los archivos de datos que haya instalado la aplicación.

El atacante ejecutará herramientas para explorar los binarios de las aplicación (EXE y DLL) buscando secretos escondidos tales como contraseñas o palabras de paso.

En el caso de tratarse de aplicaciones Web, el atacante ejecutará herramientas que rastrearán paso a paso el sitio Web generado por la aplicación ASP.NET y creará una imagen especular de dicho sitio. Analizará todos los archivos HTML en busca de comentarios que revelen información sobre su aplicación o sobre el servidor donde se ejecuta (información como direcciones IP o nombres de los otros servidores susceptibles de ser atacados). Analizará todos los campos de entrada, incluyendo campos ocultos, como un camino potencial para lanzar ataques de inyección SQL o de programación de sitio cruzado. También analizará todos los archivos de comandos incrustados, tales como *VBScript*, y desactivará la ejecución de los mismos (algo que podrá hacer sin más que configurar una opción en su explorador de Internet, en un intento por superar las validaciones de las entradas realizadas en la aplicación cliente).

Estas técnicas utilizadas por el atacante se conocen normalmente como descomposición o análisis del rastro de la aplicación.

- Creación de un esquema de la aplicación

Debe adoptarse el mismo punto de vista que tendría un atacante y crear un esquema de la aplicación para localizar los puntos vulnerables. Podrá crear este esquema revisando el código fuente y la lista de archivos incluida en la aplicación de instalación. El esquema deberá incluir:

- Todos los componentes, en concreto .EXE y .DLL, que forman su aplicación.
- Todas las funciones *Public* expuestas por las .DLL contenidas en su aplicación.
- Todos los archivos que sirvan como entrada en su aplicación.
- Todos' los archivos creados por su aplicación, incluyendo los archivos temporales.
- Los campos de entrada de los formularios de su aplicación.
- Los datos de los portapapeles pegados por su aplicación.
- Las variables de entorno de las que depende su aplicación.
- Los argumentos de la línea de comandos de los que depende su aplicación.
- Todas las entradas del registro que su aplicación lee o escribe.
- Todos los componentes de terceros llamados por su aplicación.
- Todos los .EXE externos generados por su aplicación durante su ejecución.
- Todas las bases de datos a las que se conecta su aplicación.
- Para el caso de aplicaciones Web, todos los nombres de URL utilizados por su aplicación.
- Todos los *sockets* de red que abre su aplicación.

Los elementos contenidos en esta lista son, en general, elementos informativos para un atacante. Pensar en las formas en que podrá manipular dicha entrada (elementos tales como aquellos enumerados en la lista anterior) para conseguir que la aplicación:

- Tener un fallo catastrófico, que puede derivar en un ataque de denegación de servicio o, peor aún, en un desbordamiento de búfer que permita que el atacante bloquee la máquina o la red cuando la aplicación se encuentre en ejecución.

- Genere un error que revele importantes detalles sobre la aplicación.

- Eleve los privilegios del atacante permitiéndole obtener información o ejecutar acciones que, en principio, no podría llevar a cabo.

- Creación de escenarios basados en incursiones de ataque

Debe crearse escenarios centrados en elementos de ataque (por ejemplo, suministrando entradas erróneas) obtenidos del esquema de la aplicación en un intento de forzar cualquiera de los resultados indeseables que hemos listado anteriormente. A continuación se muestran algunos ejemplos de escenarios que debería incluir en su plan de pruebas, que está basado en el esquema de su aplicación, y lo que haría un *hacker* con ellos:

- Para el caso de aplicaciones Web, intente ejecutar un ataque de inyección SQL introduciendo instrucciones SQL o *VBScript* en todos los campos de entrada.

• Modifique el contenido de los archivos XML que utilice la aplicación para leer las opciones. Por ejemplo, si el archivo XML utiliza una cadena para definir el título de la aplicación, puede probar a modificar el título utilizando una cadena de gran longitud, un número o un conjunto de caracteres extendidos (por ejemplo, caracteres japoneses) con la esperanza de hacer fallar a la aplicación o de provocar una excepción.

• Estudie las funciones *Public* expuestas por un objeto de biblioteca de clases de Visual Basic .NET cuyo fin sea ejecutarse en un servidor de red. Por ejemplo, tal vez compruebe que un objeto servidor contiene funciones *Public* que no deberían ser públicas. Podría crear fácilmente su propia aplicación Visual Basic .NET para obtener información privada del cliente sin más que llamar directamente a las funciones expuestas y pasar valores arbitrarios para localizar clientes y obtener sus números asociados de tarjeta de crédito, por ejemplo.

• En el caso de aplicaciones ASP.NET, utilice un explorador Web no tradicional o un visor de código HTML para modificar los valores contenidos en campos ocultos, tales como los campos que contienen los precios de productos, con la intención de engañar a la aplicación y obtener algún descuento en las compras.

• Intente provocar errores pasando datos que interrumpan las consultas SQL, lo que podría proporcionarle el nombre de Microsoft SQL Server y de la base de datos. Podrá utilizar esta información para intentar conectarse a la base de datos de SQL Server y obtener directamente los datos almacenados en ella.

• Desarrolle una .DLL que tenga el mismo nombre que una .DLL que utilice la aplicación e introdúzcala en su equipo. Intente engañar a la aplicación para que cargue la DLL falsa. Observe que los ensamblados .NET con nombres seguros ayudarán a impedir este tipo de ataque. Por este motivo deberá pensar seriamente en asignar nombres seguros a los ensamblados.

• En el caso de una aplicación Web, intente utilizar una dirección Web (URL) que haga referencia a una página HTML, una página ASP.NET o a un archivo de configuración (tal como un archivo *Web.Config*) al que no tenga garantizado el acceso. Una forma muy frecuente de efectuar este ataque es intentar saltarse una secuencia de pasos (por ejemplo, los pasos necesarios para localizar, adquirir y enviar un producto) que sean obligatorios en su aplicación Web.

Asignación de prioridades a los escenarios

La importancia de los escenarios depende de la aplicación. Por ejemplo, si está creando una aplicación de comercio electrónico basada en Web deberá comprobar que todos los campos importantes resistirán la introducción de entradas erróneas, algo que será más importante que verificar que todos los campos de entrada se muestran exactamente en pantalla en la posición pensada inicialmente. Si la aplicación falla porque se introduce el carácter X en lugar de introducir un número, este problema resultará infinitamente más grave que el hecho de que sus cuadros de texto se encuentran descentrados en 1 píxel. Sin embargo, si está creando un juego basado en *Windows Forms*, que no necesita ninguna entrada de texto en formato libre, las pruebas

se deberían centrar en demostrar la exactitud de la representación de los píxeles, en lugar de verificar las entradas de texto.

Una forma eficaz de asignar una prioridad más elevada a los escenarios más importantes es dejar de escribir código por un momento y reunirse con los miembros de su equipo (incluyendo los miembros de todas las disciplinas, tales como desarrollo, pruebas, documentación y soporte técnico, especialmente estos últimos porque serán al final los que den la cara ante el cliente).

Cuando comience la fase de pruebas es de vital importancia tanto analizar los escenarios que ha decidido no probar como aquellos que ha marcado como críticos. Debe revisarse de forma continua su plan de pruebas a lo largo del ciclo de desarrollo del producto y replantearse la prioridad de cada uno de los escenarios. Si se añade una nueva función que incremente la criticidad de un escenario de baja prioridad o aparece un serio error relacionado con un escenario de prioridad 4 que le obligue a elevar la prioridad del escenario, tendrá la oportunidad (y la obligación) de replantearse sus futuros esfuerzos de pruebas.

Asignación de prioridades a los escenarios relacionados con la seguridad basándonos en las amenazas

Para el propósito de priorizar los escenarios relacionados con la seguridad, se evalúa el nivel de amenaza asociado con cada uno de los escenarios detectados. Por ejemplo, si la aplicación no va a utilizar una base de datos para almacenar o recuperar información los ataques de inyección SQL no serán una amenaza. Podrá utilizar el proceso de modelización de amenazas denominado STRIDE (un acrónimo que engloba a los ataques de tipo general tales como Suplantación', Alteración de datos, Repudio, Revelación de Información, Denegación de servicio y Elevación de privilegios), que se analiza en el Capítulo anterior, para clasificar los escenarios de prueba relacionados con la seguridad.

PRIORIDAD DEL ESCENARIO	PRIORIDAD
1	Los escenarios de prioridad 1 están centrados en amenazas que tengan un fuerte impacto sobre la seguridad de la aplicación. Un ejemplo de prueba de un escenario de prioridad 1 es comprobar que en la aplicación de entrada de datos no puede introducir, en ningún campo de entrada, instrucciones SQL que puedan manipular el contenido de la base de datos subyacente. La aplicación no se puede comercializar con ningún error relacionado con un escenario de prioridad 1. Es imprescindible que se desarrolle (y ejecute) pruebas para todos los escenarios de prioridad 1.
2	Los escenarios de prioridad 2 están relacionados con las amenazas que puedan tener un impacto moderado en la seguridad de la aplicación, o en aquellos casos en los que la aplicación no va a funcionar como se esperaba en su entorno operativo de seguridad. El escenario en el que no es posible ejecutar la aplicación desde un recurso compartido de red debido a una violación de seguridad de acceso al código, es un ejemplo de escenario de prioridad 2. Obligar al usuario a ejecutar la aplicación desde la

	máquina local es un inconveniente para el usuario. Se debe modificar la aplicación para que se pueda ejecutar en todas las zonas de seguridad que estaban previstas. Deberá crear pruebas para todos los escenarios de prioridad 2.
3	Los escenarios de prioridad 3 se centran en aquellos problemas que no obstaculizan seriamente la seguridad de la aplicación pero que perjudican la estética de la aplicación desde el punto de vista de la seguridad. Por ejemplo, un escenario de prioridad 3 sería el caso de que un usuario intentara iniciar una sesión en la aplicación y ésta no se lo permitiera; el usuario podría pensar que ha introducido un nombre de usuario o una contraseña no válidos cuando, de hecho, el motivo real es que el servidor de la aplicación se encuentra deshabilitado por cuestiones de mantenimiento. En este caso, la aplicación debe mostrar un mensaje de error avisando al usuario de que el servidor no se encuentra disponible por mantenimiento. En caso contrario, el usuario se puede sentir molesto al intentar iniciar la sesión varias veces pensando que está escribiendo mal su contraseña.
4	Los escenarios de prioridad 4 no tienen un impacto apreciable sobre la seguridad. En principio no hará falta probar estos escenarios. Por ejemplo, es el caso de un mensaje de error relacionado con la seguridad en el que el texto del mensaje contiene espacios en blanco al final de la frase (espacios que el usuario no podrá detectar). Este es un claro ejemplo que no deberá probar.

Tabla 6.9. Escenarios

Generar pruebas

Una vez que se haya completado el proceso de la tormenta de ideas y se haya asignado una prioridad a los escenarios de prueba, es necesario verificar la forma en que la aplicación responde ante estos escenarios que ha identificado. Cuando se desarrollen pruebas para cada uno de los escenarios contenidos en el plan de pruebas, tiene que concederse con gran importancia a las consideraciones de índole práctica (por ejemplo, en qué forma va a llevar a cabo la prueba o el tiempo que llevará ejecutarla). Como sucede con la generación de los escenarios, también debe realizarse una tormenta de ideas para decidir qué pruebas debe realizarse. La parte más importante de este ejercicio es generar una lista completa de pruebas a las que deberá asignar una prioridad. Por ejemplo, suponga que ha definido un escenario de prioridad 1 en su plan de pruebas que indica: "Asegurarse de que sólo se pueden introducir nombres válidos de usuarios en el campo de entrada correspondiente al nombre de usuario". Tendrá que crearse una lista exhaustiva de las pruebas necesarias para validar este escenario. En este caso, si se define como nombre válido de usuario cualquier nombre que tenga un tamaño máximo de 16 caracteres y que sólo puede contener caracteres alfabéticos, podría utilizar las siguientes pruebas para verificar este escenario:

- Introducir una cadena vacía.
- Introducir una cadena de longitud superior a los 16 caracteres.
- Introducir una cadena que contenga datos numéricos y símbolos.
- Introducir una cadena que contenga códigos de caracteres extendidos
- Introducir cadenas en otros idiomas, por ejemplo en japonés.

- Introducir cadenas alfabéticas válidas que contengan una mezcla de mayúsculas y minúsculas.
- Introducir una cadena que contenga instrucciones SQL.
- Introducir una cadena que contenga el nombre de un archivo existente.
- Introducir el nombre de un usuario que contenga caracteres válidos pero que no sea un usuario contenido en la base de datos.
- Introducir nombres de usuario no válidos una y otra vez utilizando un bucle.
- Ejecutar más de una instancia de la aplicación e introducir simultáneamente los mismos nombres u otros que sean diferentes.
- Ejecutar todas las pruebas anteriores en diversas plataformas, tales como Windows Me y Windows XP.
- Ejecutar las pruebas anteriores en diferentes contextos, por ejemplo, ejecute otros tipos de aplicaciones (tal como aplicaciones de Microsoft Office) para forzar una condición de escasez de memoria y vuelva a probar entonces los escenarios anteriores.

Puede generar un gran número de pruebas para validar un determinado escenario. Deténgase en un punto en el que se sienta a gusto y en el que piense que ha cubierto todos los casos importantes.

Filtrar y dar prioridades a las pruebas de cada escenario

Para mantener las ideas claras y no entrar en un bucle sin fin de pruebas en el que tenga que llevar a cabo un número de pruebas aparentemente infinito, debe asignarse una prioridad a las pruebas identificadas previamente para cada escenario basándose en los siguientes criterios:

- Importancia del escenario que se desea probar preguntándose si la prueba indicará un defecto crítico o un fallo funcional en caso de que no sea superada.
- Aplicabilidad a varios escenarios, si una determinada prueba asociada con un escenario distinto a aquel que estamos evaluando se puede utilizar también en el escenario que esté evaluando, añada una nota en el plan de pruebas para indicar que dicha prueba es aplicable a ambos escenarios
- Costo de ejecución o de implementación de la prueba, si la creación o la ejecución de la prueba requiere una gran cantidad de esfuerzo, considérese la posibilidad de descomponer la prueba en partes más pequeñas o encuentre otras formas de crear otra prueba que le proporcione el mismo nivel de validación.

Durante el proceso de creación del esquema de la aplicación, y de la ejecución de las tormentas de ideas relacionadas con los escenarios y con las pruebas pertenecientes a cada uno de ellos, podrá descubrir defectos en el diseño de la aplicación. Si se decide modificar el diseño para reducir el riesgo y mejorar la seguridad de la aplicación, debe volverse sobre los pasos de creación del esquema de la aplicación y de los escenarios para adaptarlos al nuevo diseño. Las pruebas son un proceso y el plan de pruebas es un documento vivo. Debe tener presente la necesidad de iterar varias veces sus pruebas y su plan de pruebas durante el ciclo de vida del proyecto a medida que se vayan introduciendo nuevos cambios en el producto o a medida que se vayan descubriendo nuevos e importantes escenarios de pruebas.

ERRORES FRECUENTES COMETIDOS EN LAS PRUEBAS

Cuando se trata de hacer pruebas, suelen cometerse un cierto número de errores que impactarán negativamente en la seguridad y en la calidad de la aplicación:

☒ Probar poco y tarde

Como seres humanos tendemos a descomponer las tareas en pasos lógicos. Un paso lógico después de finalizar el desarrollo del código es comenzar las pruebas. Debe evitarse la tentación de esperar hasta haber terminado el desarrollo del código para comenzar a probar. Las pruebas, o, más apropiadamente, el proceso de pruebas, deben comenzarse exactamente en el mismo instante en el que comience el desarrollo del código.

Para evitar las prisas de última hora, deberá pensarse en descomponer el ciclo de desarrollo en varias etapas de corta duración. En cada etapa deberá marcarse unos objetivos a nivel funcional y de seguridad. Al finalizar cada etapa debe haberse logrado y verificado, mediante la ejecución de pruebas, los objetivos propuestos (tanto de seguridad como funcionales).

☒ Fracasar en las pruebas de seguridad

Puede fracasar en la comprobación global de la seguridad o puede fallar al no probar ciertos aspectos de la seguridad que se le hayan pasado por alto. Dejar a un lado completamente los temas asociados con la seguridad resulta una total irresponsabilidad, especialmente si va a instalar y utilizar la aplicación en un entorno compartido. Sin embargo, no probar ciertos temas de seguridad porque no es consciente de los problemas asociados suele ser aceptable (y comprensible), ya que el problema de la seguridad sigue siendo desconocido, y mal comprendido, en muchos casos

Por definición, resulta imposible probar los aspectos de la seguridad que le sean desconocidos. Sólo hasta que aparezca una nueva amenaza de seguridad puede comenzarse a desarrollar pruebas para verificar el comportamiento de su aplicación frente a esta determinada amenaza.

El desafío es mantenerse informado de los nuevos problemas de seguridad a medida que vayan apareciendo. Cuando sea consciente de la aparición de una nueva amenaza de seguridad, debe probar la aplicación frente a dicha amenaza.

Otro error cometido con frecuencia es no volver a probar la aplicación (o componente) cuando la modifique para cumplir un nuevo objetivo. Por ejemplo, imagínese que ha desarrollado un componente que actúa como un objeto empresarial de nivel intermedio. Debe volverse a probar dicho componente si va a comenzar a distribuirse utilizando otro canal de distribución, tal como un servicio Web a través de Internet. Siempre que introduzca una aplicación (o componente) en un nuevo entorno, estará sometida a un nuevo conjunto de amenazas. Asegúrese de volver a probar su aplicación para comprobar su comportamiento frente a estas nuevas amenazas.

☒ Fracasar a la hora de estimar el costo de las pruebas

Saber cuándo ha terminado de desarrollar todo el código suele ser más sencillo que predecir el momento en el que la aplicación es totalmente segura y se encuentra libre de todo tipo de fallo

crítico. El calendario de desarrollo de la aplicación se suele utilizar como base para determinar el calendario de comercialización del producto, y finalizar el código suele tomarse erróneamente como el punto final de todo el trabajo. Debe intentar pensar no en términos de calendario de desarrollo sino en términos de calendario del producto. Compruebe que ha tenido en cuenta el tiempo necesario para estabilizar la aplicación cuando defina el ciclo de desarrollo de la aplicación. Debe invertirse tanto tiempo o más para ejecutar las pruebas que para desarrollar la propia aplicación.

☒ Esperar demasiado de la información obtenida con la versión beta

No piense que las versiones beta van a permitir encontrar todos los errores existentes en la aplicación, en particular los errores de seguridad. Son muchos los motivos que tendrán los clientes para probar el producto, pero casi nunca tendrán en mente realizar las pruebas en las que se está pensando como diseñador. Salvo que un cliente ofrezca generosamente sus servicios como probador o salvo que encuentre demasiadas dificultades a la hora de utilizar el nuevo producto, no recibirá información interesante de su parte (por ejemplo, información que le diga que el sistema de autenticación es ineficaz o que la aplicación es propensa a sucumbir ante ataques de denegación de servicio). Los usuarios de versiones beta suponen que se conocen estos problemas y que se resolverán antes de comercializar la versión final. La única vez que se puede esperar obtener una información de utilidad es cuando alguien haya invertido parte de su dinero, haya descargado el producto o lo haya comprado en alguna tienda y haya comenzado a usarlo.

☒ Suponer que los componentes desarrollados por terceros son seguros

Muchas empresas han aprendido a la fuerza que no deben confiar plenamente en los componentes desarrollados por terceros. No confíe en que los componentes desarrollados por terceros van a mantener segura su aplicación. Utilizar componentes de terceros puede disminuir el tiempo de desarrollo pero no acortará en absoluto el tiempo que tendrá que dedicarle a las pruebas. No puede trasladar la responsabilidad de probar los componentes desarrollados por terceros a esas mismas empresas.

Es una responsabilidad probar que la aplicación interactúa de una manera segura con todos los componentes desarrollados por terceros como, por ejemplo, con los componentes de nivel intermedio, controles de clientes o controles Web.

6.3.7. MECANISMOS DE SEGURIDAD (EN EL ENTORNO DE DESARROLLO DE VISUAL STUDIO .NET)

Desde el punto de vista informático, un mecanismo de seguridad es la base o estructura de una herramienta que proporcionan la solución a un determinado problema.

Por ejemplo: es de interés proteger los servidores y los sistemas o aplicaciones del ataque de código malintencionado y datos dañados. Para esto, existen varios mecanismos en el entorno de desarrollo que se pueden aprovechar en las estrategias. Los siguientes mecanismos de seguridad permiten garantizar la seguridad de las aplicaciones en una red:

☑ Cifrado

• El cifrado puede hacerse utilizando sistemas criptográficos, y se puede aplicar extremo a extremo o individualmente a cada enlace del sistema de comunicaciones.

• El mecanismo de cifrado soporta el servicio de confidencialidad de datos al tiempo que actúa como complemento de otros mecanismos de seguridad.

☑ Firma Digital

• Se puede definir la firma digital como el conjunto de datos que se añaden a una unidad de datos para protegerlos contra la falsificación, permitiendo al receptor probar la fuente y la integridad de los mismos.

• La firma digital supone el cifrado, con una componente secreta del firmante, de la unidad de datos y la elaboración de un valor de control criptográfico.

• Para verificar la firma, el receptor descifra la firma con la clave pública del emisor, comprime con una función *hash* al texto original recibido y compara el resultado de la parte descifrada con la parte comprimida. Si ambas coinciden el emisor tiene la garantía de que el texto no ha sido modificado.

• Como el emisor utiliza la clave secreta para cifrar la parte comprimida del mensaje, puede probarse ante una tercera parte, que la firma sólo ha podido ser generada por el usuario que guarda la componente secreta.

☑ Control de Acceso

• Este mecanismo se utiliza para autenticar las capacidades de una entidad, con el fin de asegurar los derechos de acceso a recursos que posee.

• El control de acceso se puede realizar en el origen o en un punto intermedio, y se encarga de asegurar si el enviante está autorizado a comunicar con el receptor y/o a usar los recursos de comunicación requeridos.

• Si una entidad intenta acceder a un recurso no autorizado, o intenta el acceso de forma impropia a un recurso autorizado, entonces la función de control de acceso rechazará el intento, al tiempo que puede informar del incidente, con el propósito de generar una alarma y/o registrarlo.

☑ Integridad de Datos

• Para proporcionar la integridad de una unidad de datos la entidad emisora añade una cantidad que se calcula en función de los datos. Esta cantidad, probablemente encriptada, puede ser una información suplementaria compuesta por un código de control de bloque, o un valor de control criptográfico. La entidad receptora genera la misma cantidad a partir del texto original y la compara con la recibida para determinar si los datos no se han modificado durante la transmisión.

• Para proporcionar integridad a una secuencia de unidades de datos se requiere, adicionalmente, alguna forma de ordenación explícita, tal como la numeración de secuencia, un sello de tiempo o un encadenamiento criptográfico.

☑ **Intercambio de Autenticación**

• Existen dos grados en el mecanismo de autenticación:

– Autenticación simple. El emisor envía su nombre distintivo y una contraseña al receptor, el cual los comprueba.

– Autenticación fuerte. Utiliza las propiedades de los criptosistemas de clave pública. Cada usuario se identifica por un nombre distintivo y por su clave secreta. Cuando un segundo usuario desea comprobar la autenticidad de su interlocutor deberá comprobar que éste está en posesión de su clave secreta, para lo cual deberá obtener su clave pública.

• Para que un usuario confíe en el procedimiento de autenticación, la clave pública de su interlocutor se tiene que obtener de una fuente de confianza, a la que se denomina Autoridad de Certificación. La Autoridad de Certificación utiliza un algoritmo de clave pública para certificar la clave pública de un usuario produciendo así un certificado.

• Un certificado es un documento firmado por una Autoridad de Certificación, válido durante el período de tiempo indicado, que asocia una clave pública a un usuario.

6.3.8. EQUIPOS DE RESPUESTA A INCIDENTES (GRUPO DE RESPUESTA)

Es aconsejable formar un equipo de respuesta a incidentes. Este equipo debe estar implicado en los trabajos proactivos del profesional de la seguridad. Entre éstos se incluyen:

☑ El desarrollo de instrucciones para controlar incidentes.
☑ La identificación de las herramientas de software para responder a incidentes y eventos.
☑ La investigación y desarrollo de otras herramientas de seguridad informática.
☑ La realización de actividades formativas y de motivación.
☑ La realización de investigaciones acerca de virus.
☑ La ejecución de estudios relativos a ataques al sistema.

Estos trabajos proporcionarán los conocimientos que la organización puede utilizar y la información que hay que distribuir antes y durante los incidentes. Una vez que el administrador de seguridad y el equipo de respuesta a incidentes han realizado estas funciones proactivas, el administrador debe delegar la responsabilidad del control de incidentes al equipo de respuesta a incidentes. Esto no significa que el administrador no deba seguir implicado o formar parte del equipo, sino que no tenga que estar siempre disponible, necesariamente, y que el equipo debe ser capaz de controlar los incidentes por sí mismo.

El equipo será el responsable de responder a incidentes como virus, gusanos o cualquier otro código dañino; invasión; engaños; desastres naturales y ataques del personal interno. El equipo también debe participar en el análisis de cualquier evento inusual que pueda estar implicado en la seguridad de los equipos o de la red.

6.3.8.1. PLAN DE CONTINGENCIA

El Plan de Contingencia es un documento hecho para permitir a una instalación de cómputo reestablecer sus operaciones en el caso de un desastre, tales como: terremotos, incendios, sabotajes, fallas de hardware, errores humanos, etc.

El propósito principal es "mantener a la empresa o institución y sus actividades operando aún en una situación de desastre". Es decir, habilitar a la organización para responder y sobrevivir a problemas críticos o catastróficos, de modo que permita una pronta recuperación de la operación normal.

Es muy importante el poder desarrollar un plan que resulte práctico y eficaz, y poder probarlo para asegurar que éste funcionará cuando se requiera. Para lo cual se debe considerar que la pérdida parcial o total de las facilidades del procesamiento de datos, puede causar, entre otras cosas:

- Pérdidas financieras directas e indirectas
- Pérdidas de producción
- Pérdida de clientes
- Costos de compensación
- Pérdidas de control
- Información errónea o incompleta
- Base pobre para la toma de decisiones

Existen, al menos, cuatro opciones para Desarrollar un Plan de Contingencia:

- Desarrollo en casa. Esta opción es barata, pero puede tomar mucho tiempo para su implementación por falta de experiencia, así como la necesidad de diseñarlos desde cero. Tiene la ventaja que lo podemos ajustar a las necesidades específicas de la organización.
- Comprar un paquete de software. Existen en el mercado este tipo de programas con opciones y menús para guiar al usuario en el desarrollo del plan. Esta opción es más sencilla y más rápida que la anterior, pero generalmente, más cara.
- Contratar consultores. Aunque esta opción es la más cara de todas, un consultor experto nos puede ahorrar mucho tiempo debido a su experiencia con otros clientes del ramo.
- Combinación de las anteriores. El plan puede ser diseñado desde cero o usando un paquete de software, y luego puede ser revisado por un consultor, el cual podría incluso proveer el equipo de respaldo y soporte necesarios. El consultor puede entonces evaluar si el plan es adecuado o no.

  - El plan de contingencia va a depender del tipo de actividad que realiza la organización (naturaleza y efecto que pueda producir).
  - En una empresa el Plan de Contingencia debe considerar dos aspectos: operacional y administrativo.

- Aspecto Operacional: en el nivel operacional, cada usuario debe saber qué hacer cuando aparezca un problema. Además debe conocer la respuesta a la pregunta: ¿a quién hay que llamar?. Es muy importante que el plan de contingencia determine quién debe tomar todas decisiones durante la recuperación del desastre y establezca la disponibilidad y entrenamiento del personal debidamente experimentado.
- Aspecto Administrativo: en el nivel administrativo se debe asegurar la implantación de un plan adecuado y la debida actualización del mismo, conforme van cambiando las situaciones en la empresa. Dicho en otras palabras, conviene estar actualizando el plan, debido a factores tales como, rotación de personal, cambios de tecnología en equipos, comunicación, aplicaciones o sistemas, apertura de nuevos negocios o locales de venta, etc.

Ciclo de Vida de los Desastres

En la decisión de qué es lo que se debería incluir en el Plan de Contingencia, debe conocerse el ciclo de vida de un desastre. El ciclo de vida de un desastre consta de cuatro fases o períodos de tiempo:

1. Operaciones Normales

   Indican el período de tiempo antes de que ocurra un desastre. Esta fase del Plan debe incluir la práctica de las operaciones que pretenden prevenir un desastre desde que comienza y de aquellas que ayudan a mitigar el impacto del mismo y prever lo que podrá ocurrir.

2. Respuestas de Emergencia

   Las respuestas derivadas de una situación de emergencia ocurren durante las pocas horas que siguen inmediatamente a un desastre. Esta fase del plan identifica las actividades que puedan necesitar mayor atención durante este período con la finalidad de asegurar una respuesta a la organización y proporcionar una lista de verificación (*checklist*) de las omisiones importantes que puedan pasar inadvertidas durante la confusión que acompañan a los desastres.

3. Procesamiento Interno

   Es un procesamiento alternativo que representa el tiempo de duración de la contingencia en relación con el soporte de las funciones esenciales de la empresa hasta que la capacidad de procesamiento normal sea restaurada.

4. Restauración

   Indica el período de tiempo destinado a aquellas actividades que se necesita realizar para recuperar una condición o capacidad de procesamiento en su operación normal. La restauración involucra necesariamente pasos de planeación, organización y control de tales actividades. Cuando un servicio falla, la tarea principal del servidor debe ser la recuperación mientras que la responsabilidad principal del usuario es la de dar continuidad a las operaciones.

Elementos de un Plan de Contingencia

Algunos de los elementos que se deben incluir son los siguientes:

1. Lista de Personal Clave de la empresa y teléfonos de contacto
2. Lista de distribución de equipamiento computacional
3. Lista de proveedores (contactos y números de teléfonos)
4. Ubicación del Plan de Contingencia y revisión periódica
5. Procedimientos de recuperación post-desastre y normalización de los servicios
6. Funciones del Comité de Recuperación y responsabilidades individuales
7. Acuerdos contractuales con las locaciones de respaldo y su vigencia
8. Descripción del hardware, software, arquitectura del equipo y red LAN
9. Hardware del centro de cómputo primario, equipo periférico y configuración del software
10. Lista de trabajos y procesos prioritarios y sus agendas
11. Procedimientos manuales alternativos.
12. Ubicación y disponibilidad de archivos de datos, diccionarios de datos, procedimientos de control de trabajos, programas, documentación y manuales.

Planeación de un Plan de Contingencia

El objetivo de un Plan de Contingencia es permitir que una organización "sobreviva" a un desastre y continúe las operaciones del negocio normalmente. Algunos problemas más importantes que deben incluirse son:

- Entrenamiento del personal responsable de algunos procesos o actividades críticas.
- Reducir las operaciones y resultados de procesos
- Reducir inmediatamente el daño y pérdidas.
- Establecer sucesiones de la dirección y fuerza de emergencia
- Facilitar la efectiva coordinación de tareas de recuperación
- Identificar líneas críticas de los negocios y funciones de soporte.
- Una Metodología de Desarrollo de un Plan de Contingencia
  - FASE 1: Descubrir y Analizar Información
  - FASE 2: Identificar Alternativas de Procedimientos
  - FASE 3: Identificar y Documentar
  - FASE 4: Identificar Información Adicional
  - FASE 5: Finalizar y Revisar Plan

6.3.9. EVALUACIÓN DE COSTOS

Desde un punto de vista oficial, el desafío de responder la pregunta del valor de la información ha sido siempre difícil, y más difícil aún hacer estos costos justificables, siguiendo el principio que "si desea justificarlo, debe darle un valor".

Establecer el valor de los datos es algo totalmente relativo, pues la información constituye un recurso que, en muchos casos, no se valora adecuadamente debido a su intangibilidad, cosa que no ocurre con los equipos, la documentación o las aplicaciones.

Además, las medidas de seguridad no influyen en la productividad del sistema por lo que las organizaciones son reticentes a dedicar recursos a esta tarea. Por eso es importante entender que los esfuerzos invertidos en la seguridad son costeables.

La evaluación de costos más ampliamente aceptada consiste en cuantificar los daños que cada posible vulnerabilidad puede causar teniendo en cuenta las posibilidades. Un planteamiento posible para desarrollar esta política es el análisis de lo siguiente:

- ¿Qué recursos se quieren proteger?
- ¿De qué personas necesita proteger los recursos?
- ¿Qué tan reales son las amenazas?
- ¿Qué tan importante es el recurso?
- ¿Qué medidas se pueden implantar para proteger sus bienes de una manera económica y oportuna?

Con esas sencillas preguntas (más la evaluación de riesgo) se debería conocer cuáles recursos vale la pena (y justifican su costo) proteger, y entender que algunos son más importantes que otros.

El objetivo que se persigue es lograr que un ataque a los bienes sea más costoso que su valor, invirtiendo menos de lo que vale. Para esto se define tres costos fundamentales:

☑ CP: Valor de los bienes y recursos protegidos.

☑ CR: Costo de los medios necesarios para romper las medidas de seguridad establecidas.

☑ CS: Costo de las medidas de seguridad.

Para que la política de seguridad sea lógica y consistente se debe cumplir que:

• CR> CP: o sea que un ataque para obtener los bienes debe ser más costoso que el valor de los mismos. Los beneficios obtenidos de romper las medidas de seguridad no deben compensar el costo de desarrollo del ataque.

• CP> CS: o sea que el costo de los bienes protegidos debe ser mayor que el costo de la protección.

CR > CP> CS y lo que se busca es:

• "Minimizar el costo de la protección manteniéndolo por debajo del de los bienes protegidos". Si proteger los bienes es más caro de lo que valen (el lápiz dentro de la caja fuerte), entonces resulta más conveniente obtenerlos de nuevo en vez de protegerlo.

• "Maximizar el costo de los ataques manteniéndolo por encima del de los bienes protegidos". Si atacar el bien es más caro de lo que valen, al atacante le conviene más obtenerlo de otra forma menos costosa.

Se debe tratar de valorar los costos en que se puede incurrir en el peor de los casos contrastando con el costo de las medidas de seguridad adoptadas. Se debe poner especial énfasis en esta etapa para no incurrir en el error de no considerar costos, muchas veces, ocultos y no obvios (costos derivados).

Con el **Valor Intrínseco** es el más fácil de calcular ya que solo consiste en otorgar un valor a la información contestando preguntas como las mencionadas y examinando minuciosamente todos los componentes a proteger. Deben abarcarse todas las posibilidades, intentando descubrir todos los **valores derivados de la pérdida** de algún componente del sistema. Muchas veces se trata del valor añadido que gana un atacante y la repercusión de esa ganancia para el entorno, además del costo del elemento perdido. Deben considerarse elementos como:

- Información aparentemente inocua como datos personales, que pueden permitir a alguien suplantar identidades.
- Datos confidenciales de acuerdos y contratos que un atacante .podría usar para su beneficio.
- Tiempos necesarios para obtener ciertos bienes. Un atacante podría acceder a ellos para ahorrarse el costo (y tiempo) necesario para su desarrollo.

Una vez evaluados los riesgos y los costos en los que se está dispuesto a incurrir y decidido el nivel de seguridad a adoptar, podrá obtenerse un **punto de equilibrio** entes estas magnitudes:

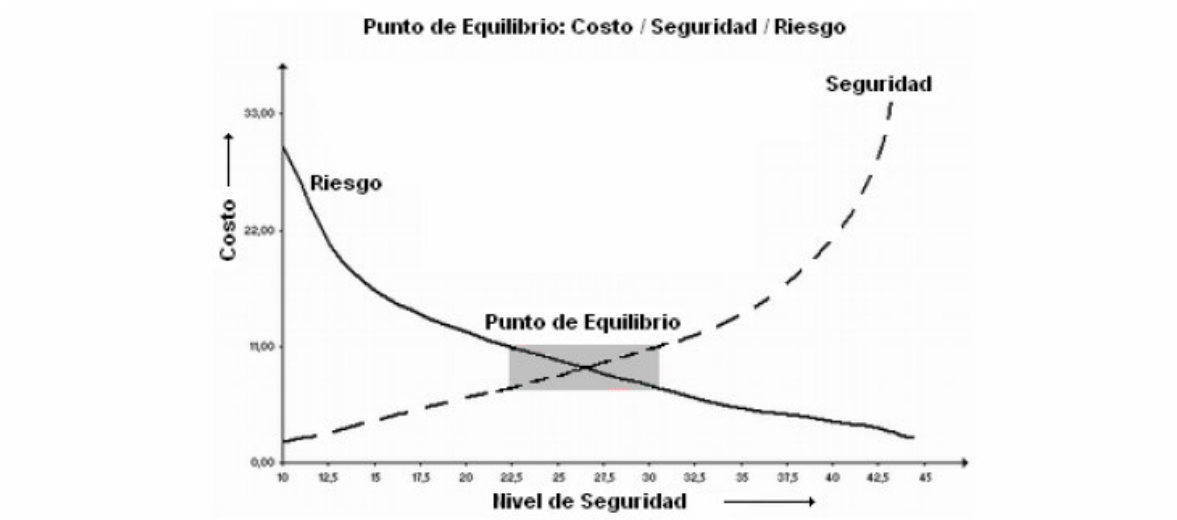


Figura 6.4 Punto de Equilibrio Costo vs. Nivel de Seguridad

Como puede apreciarse los riesgos disminuyen al aumentar la seguridad (y los costos en los que incurre) pero como ya se sabe los costos tenderán al infinito sin lograr el 100% de seguridad y por supuesto nunca se lograr a no correr algún tipo de riesgo Lo importante es lograr conocer cuan seguro se estará conociendo los costos y los riesgos que se corren, un punto de Equilibrio.

6.4. ¿CÓMO SE DISEÑAN LAS ESTRATEGIAS DE SEGURIDAD EN LA PROGRAMACIÓN?

La pregunta de este subtema permite explicar y comprender una metodología para definir una estrategia de seguridad informática que puedan utilizar los programadores para implementar y desarrollar políticas, controles y estructuras de seguridad en las aplicaciones o sistemas con el objeto de aminorar los posibles ataques y amenazas. La metodología puede utilizarse en todos los tipos de ataques a sistemas, independientemente de que sean intencionados, no intencionados o

desastres naturales, y, por consiguiente, se puedan volver a utilizar en distintos casos de ataque. La metodología se basa en los distintos tipos de amenazas, métodos de ataque y puntos vulnerables, que ya han sido explicados a lo largo de este y del anterior capítulo.

Es necesario considerar seis puntos de partida al crear las estrategias de seguridad, que a continuación se muestran en el orden de prioridad de análisis.

1. Determinar puntos vulnerables (debilidades) y amenazas.
2. Identificar métodos, herramientas y técnicas de ataques probables.
3. Establecer estrategias proactivas y reactivas.
4. Pruebas y estudio de resultados.
5. Revisar las políticas y controles actuales de seguridad.
6. Revisiones y ajustes periódicos.

6.4.1. METODOLOGÍA PARA LA DEFINICIÓN DE ESTRATEGIAS DE SEGURIDAD

Las estrategias de seguridad informática se pueden utilizar para definir e implementar directivas y controles de seguridad con el objeto de aminorar los posibles ataques y amenazas. La metodología debe poder ser utilizada en todos los tipos de ataques a sistemas, independientemente de que sean intencionados, no intencionados o desastres naturales.

La presente metodología está diseñada para ayudar a los responsables de la seguridad a desarrollar estrategias que permitan proteger la disponibilidad, integridad y confidencialidad de los datos de los sistemas informáticos de las organizaciones.

Esta metodología pretende mostrar que cada análisis, evaluación, prueba o mecanismo es una pieza que construye el proceso que libera y exenta de peligros partiendo de un conjunto de reglas que optimizan las decisiones antes, durante y después del peligro. Esta metodología es análoga a un rompecabezas (puzzle), donde la idea fundamental es embonar las piezas hasta formar una figura final. Debe cumplirse la regla que al embonar las piezas, estas deben permitir que otras le sean añadidas y además entre las embonadas se cree una figura parcial de la final, de tal forma que al embonar las figuras parciales se forme la figura final. Si estas reglas básicas las transformamos a una metodología para la definición de estrategias de seguridad, puedo decir que:

- La figura final del rompecabezas corresponde a la estrategia de seguridad
- Las reglas para armar el rompecabezas corresponden a revisar las políticas y controles actuales de seguridad para construir la estrategia de seguridad.
- Voltear las piezas del rompecabezas cara arriba corresponde a tener un panorama claro del objetivo para poder predecir posibles ataques, analizar riesgos y contemplarlos en la elaboración de la estrategia de seguridad.
- Cada una de las piezas en el rompecabezas corresponde los aspectos que debe considerarse en la estrategia de seguridad.
- Cada figura parcial construida del rompecabezas corresponde a los métodos, las estrategias (proactivas y reactivas), herramientas y técnicas que se construyen por separado de la estrategia de seguridad.

- El comprobar que se continúa creando la figura al embonar una nueva pieza en el rompecabezas corresponde a realizar un análisis, prueba y revisión del seguimiento de los pasos o ajustes al crear métodos, herramientas, estrategias (proactivas y reactivas) y técnicas en la estrategia de seguridad.

Al realizar esta analogía no pretendo decir que es "cosa de juego" crear una estrategia de seguridad y mucho menos que sea una tarea sencilla, por el contrario, quiero mostrar que puede llegar a ser una tarea tan compleja y laboriosa como armar un rompecabezas de más de 1000 piezas.

VOLTEAR LAS PIEZAS DEL ROMPECABEZAS CARA ARRIBA (Predecir posibles ataques y analizar riesgos)

La primera fase de la metodología es determinar los ataques que se pueden esperar y las formas de defenderse contra ellos. Es imposible estar preparado contra todos los ataques; por lo tanto, hay que prepararse para los que tiene más probabilidad de sufrir la aplicación o el sistema. Siempre es mejor prevenir o aminorar los ataques que reparar el daño que han causado.

Para mitigar los ataques es necesario conocer las distintas amenazas que ponen en peligro los sistemas, las técnicas correspondientes que se pueden utilizar para comprometer los controles de seguridad y los puntos vulnerables que existen en las políticas de seguridad. El conocimiento de estos tres elementos de los ataques ayuda a predecir su aparición e, incluso, su duración o ubicación. La predicción de los ataques trata de pronosticar su probabilidad, lo que depende del conocimiento de sus distintos aspectos. Los diferentes aspectos de un ataque se pueden mostrar en la siguiente ecuación:

Amenazas+ Motivos + Herramientas y técnicas + Puntos vulnerables = Ataque

Que explica que un Ataque siempre estará en función de las amenazas, motivos para realizar el ataque, las herramientas y técnicas para atacar y los puntos por los cuales puede ocurrir el ataque. Para comprender de manera amplia este primer punto es recomendable considerar y analizar el tema relacionado al "Análisis de riesgo" y "Ataques a las aplicaciones" en este mismo capítulo.

DETECTAR LAS PIEZAS CLAVE (Analizar cada tipo de amenaza y tipo de método de ataque)

Considere todas las amenazas posibles que causan ataques en los sistemas. Entre éstas se incluyen los agresores malintencionados, las amenazas no intencionadas y los desastres naturales. Este tema fue tratado con mayor detenimiento en "Análisis de Amenazas" en este mismo capítulo.

Amenazas como empleados ignorantes o descuidados, y los desastres naturales no implican motivos u objetivos; por lo tanto, no se utilizan métodos, herramientas o técnicas predeterminadas para iniciar los ataques. Casi todos estos ataques o infiltraciones en la seguridad se generan internamente; raras veces los va a iniciar alguien ajeno a la organización.

Para todos los tipos de amenazas, el personal de seguridad necesita realizar un Ejercicio de amenazas, de esta manera puede determinarse la forma de responder este tipo de incidentes.

Para iniciar un ataque, se necesita un método, una herramienta o una técnica para explotar los distintos puntos vulnerables de los sistemas, de las políticas de seguridad y de los controles. Los agresores pueden utilizar varios métodos para iniciar el mismo ataque. Por lo tanto, la estrategia defensiva debe personalizarse para cada tipo de método utilizado en cada tipo de amenaza. De nuevo, es importante que los profesionales de la seguridad estén al día en los diferentes métodos, herramientas y técnicas que utilizan los agresores. Puede encontrar una explicación detallada al respecto en "Amenazas a la seguridad". La siguiente es una lista breve de estas técnicas:

- ☑ Ataques de denegación de servicio
- ☑ Ataques de invasión
- ☑ Ingeniería social
- ☑ Virus
- ☑ Gusanos
- ☑ Caballos de Troya
- ☑ Modificación de paquetes
- ☑ Repetición de paquetes
- ☑ Adivinación de contraseñas
- ☑ Interceptación de correo electrónico

SEPARAR Y COLOCAR LAS PIEZAS CLAVE (Definir estrategias, herramientas, técnicas…)

Lo ideal en esta etapa es definir las "subestrategias" de seguridad que son las herramientas, técnicas, etc., que se construyen a partir de las amenazas y ataques que se localicen de los dos análisis anteriores. Cuando digo subestrategia no me refiero a una estrategia alternativa sino a una "división" de la estrategia principal. Por eso cuando hablo de a una estrategia proactiva o una estrategia reactiva estoy pensando en la subestrategia de seguridad.

EMBONANDO LA PRIMER ESTRATEGIA

La **estrategia proactiva** es un conjunto de pasos predefinidos que deben seguirse para evitar ataques antes de que ocurran. Entre estos pasos se incluye observar cómo podría afectar o dañar el sistema, y los puntos vulnerables que explota (pasos 1 y 2). Los conocimientos adquiridos en estas evaluaciones pueden ayudar a implementar las políticas de seguridad que controlarán o aminorarán los ataques. Éstos son los pasos de la estrategia proactiva:

1. Determinar o Predecir el daño que causará el ataque.
2. Establecer y determinar los puntos vulnerables y las debilidades que explotará el ataque.
3. Reducir los puntos vulnerables y las debilidades que se ha determinado en el sistema para ese tipo de ataque específico.
4. Elaborar planes de contingencia.

El seguimiento de estos pasos para analizar los distintos tipos de ataques tiene una ventaja adicional: comenzar a emerger un modelo, ya que en los diferentes factores se superponen para diferentes ataques. Este modelo puede ser útil al determinar las áreas de vulnerabilidad que plantean el mayor riesgo para la institución. También es necesario tomar nota del costo que supone la pérdida de los datos frente al de la implementación de controles de seguridad. La ponderación de los riesgos y los costos forma parte de un análisis de riesgos del sistema.

Considerando la metodología que en un principio se planteó el resultado de este primer embonado es la figura 6.5:

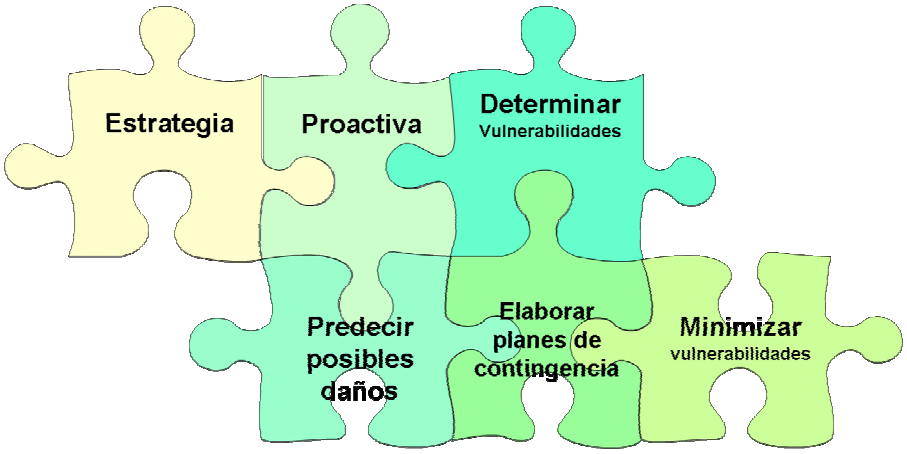


Figura 6.5 Armado parcial del rompecabezas de la Estrategia Proactiva

Las políticas y controles de seguridad no serán, en ningún caso, totalmente eficaces al eliminar los ataques. Éste es el motivo por el que es necesario desarrollar planes de recuperación y de contingencia en caso de que se quebranten los controles de seguridad.

Predecir y determinar el daño posible que puede causar un ataque

Los daños posibles pueden oscilar entre pequeños fallos del equipo y la pérdida, catastrófica, de los datos. El daño causado al sistema dependerá del tipo de ataque. Si es posible, utilice un entorno de prueba o de laboratorio para clarificar los daños que provocan los diferentes tipos de ataques. Ello permitirá al personal de seguridad ver el daño físico que causan los ataques experimentales. No todos los ataques causan el mismo daño. Éstos son algunos ejemplos de las pruebas que hay que ejecutar:

- ☑ Simular un ataque con virus a través de correo electrónico en el sistema del laboratorio y ver el daño que ha provocado y cómo recuperarse de la situación.
- ☑ Utilizar la ingeniería social para adquirir un nombre de usuario y una contraseña de algún empleado ingenuo y observar cómo se comporta.
- ☑ Simular lo que ocurriría ante un incendio en la sala de servidores. Mida el tiempo de producción perdido y el tiempo necesario para la recuperación.
- ☑ Simular un ataque de virus dañino. Anote el tiempo necesario para recuperar un equipo y multiplique ese tiempo por el número de equipos del sistema infectados para averiguar el tiempo de inactividad y la pérdida de productividad.

También es aconsejable implicar al equipo de respuesta a incidentes ya mencionado, ya que es más probable que un equipo, en lugar de una sola persona, consiga localizar todos los tipos distintos de daños que se han producido.

Determinar los puntos vulnerables o las debilidades que pueden explotar los ataques

Si se pueden descubrir los puntos vulnerables que explota un ataque específico, se pueden modificar las políticas y los controles de seguridad actuales o implementar otras nuevas para

reducir estos puntos vulnerables. La determinación del tipo de ataque, amenaza y método facilita el descubrimiento de los puntos vulnerables existentes. Esto se puede reconocer por medio de una prueba real.

A continuación encontrará una lista de los posibles puntos vulnerables. Éstos representan solamente unos pocos de los muchos que existen e incluyen ejemplos en las áreas de seguridad física, de datos y de red.

Seguridad física:

- ☑ ¿Hay bloqueos y procedimientos de entrada para obtener acceso a los servidores?
- ☑ ¿Es suficiente el aire acondicionado y se limpian regularmente los filtros? ¿Están protegidos los conductos de aire acondicionado contra robos?
- ☑ ¿Hay sistemas de alimentación ininterrumpida y generadores, y se comprueban en los procedimientos de mantenimiento?
- ☑ ¿Hay equipo para la extinción de incendios y procedimientos de mantenimiento apropiados para el equipo?
- ☑ ¿Hay protección contra el robo de hardware y software? ¿Se guardan los paquetes y licencias de software y las copias de seguridad en lugares seguros?
- ☑ ¿Hay procedimientos para almacenar los datos, copias de seguridad y software con licencia en las instalaciones y fuera de ellas?

Seguridad de datos:

- ☑ ¿Qué controles de acceso, controles de integridad y procedimientos de copias de seguridad existen para limitar los ataques?
- ☑ ¿Hay políticas de privacidad y procedimientos que deban cumplir los usuarios?
- ☑ ¿Qué controles de acceso a los datos (autorización, autenticación e implementación) hay?
- ☑ ¿Qué responsabilidades tienen los usuarios en la administración de los datos y las aplicaciones?
- ☑ ¿Se han definido técnicas de administración de los dispositivos de almacenamiento con acceso directo? ¿Cuál es su efecto en la integridad de los archivos de los usuarios?
- ☑ ¿Hay procedimientos para controlar los datos importantes?

Seguridad de la red:

- ☑ ¿Qué tipos de controles de acceso (Internet, conexiones de la red de área extensa, etc.) existen?
- ☑ ¿Hay procedimientos de autenticación? ¿Qué protocolos de autenticación se utilizan en las redes de área local, redes de área extensa y servidores de acceso telefónico? ¿Quién tiene la responsabilidad de la administración de la seguridad?
- ☑ ¿Qué tipo de medios de red, por ejemplo, cables, conmutadores y enrutadores, se utilizan? ¿Qué tipo de seguridad tienen?
- ☑ ¿Se ha implementado la seguridad en los servidores de archivos y de impresoras?
- ☑ ¿Hace uso la organización del cifrado y la criptografía en Internet, redes privadas virtuales (VPN), sistemas de correo electrónico y acceso remoto?
- ☑ ¿Se ajusta la organización a las normas de redes?

Reducir o Minimizar los puntos vulnerables y debilidades que puede explotar un posible ataque

La reducción de los puntos vulnerables y las debilidades del sistema de seguridad que se determinaron en la evaluación anterior es el primer paso para desarrollar políticas y controles de seguridad eficaces. Ésta es la compensación de la estrategia proactiva. Mediante la reducción de los puntos vulnerables, el personal de seguridad puede hacer disminuir tanto la probabilidad de un ataque como su eficacia, si se produce alguno. Tenga cuidado de no implementar controles demasiado estrictos, ya que la disponibilidad de la información se convertiría en un problema. Debe haber un cuidado equilibrio entre los controles de seguridad y el acceso a la información. Los usuarios deben tener la mayor libertad posible para tener acceso a la información.

Elaborar planes de contingencia

Un plan de contingencia es un plan alternativo que debe desarrollarse en caso de que algún ataque penetre en el sistema y dañe los datos o cualquier otro activo, detenga las operaciones comerciales habituales y reste productividad. El plan se sigue si el sistema no se puede restaurar a tiempo. Su objetivo final es mantener la disponibilidad, integridad y confidencialidad de los datos.

Debe haber un plan para cada tipo de ataque y tipo de amenaza. Cada plan consta de un conjunto de pasos que se han de emprender en el caso de que un ataque logre pasar las políticas de seguridad. El plan de contingencia debe:

- ☑ Determinar quién debe hacer qué, en qué momento y en qué lugar para que la organización siga funcionando.
- ☑ Ensayarse periódicamente para mantener al personal informado de los pasos de la contingencia actual.
- ☑ Abarcar la restauración de las copias de seguridad.
- ☑ Explicar la actualización del software antivirus.
- ☑ Abarcar el traspaso de la producción a otra ubicación o sitio.

Los siguientes puntos resaltan las distintas tareas que deben evaluarse para desarrollar un plan de contingencia:

- ☑ Evaluar las políticas y controles de seguridad de la organización para utilizar todas las oportunidades destinadas a reducir los puntos vulnerables. La evaluación debe tratar el plan y los procedimientos de emergencia actuales de la organización y su integración en el plan de contingencia.
- ☑ Evaluar los procedimientos actuales de respuesta ante emergencias y su efecto en el funcionamiento continuo de la organización.
- ☑ Desarrollar respuestas planeadas a ataques, integrarlas en el plan de contingencia y anotar hasta qué punto son adecuadas para limitar el daño y reducir el impacto del ataque en las operaciones de procesamiento.
- ☑ Evaluar procedimientos de copia de seguridad, que incluyan la documentación más reciente y pruebas de recuperación de desastres, para evaluar su adecuación e integrarlos en el plan de contingencia.

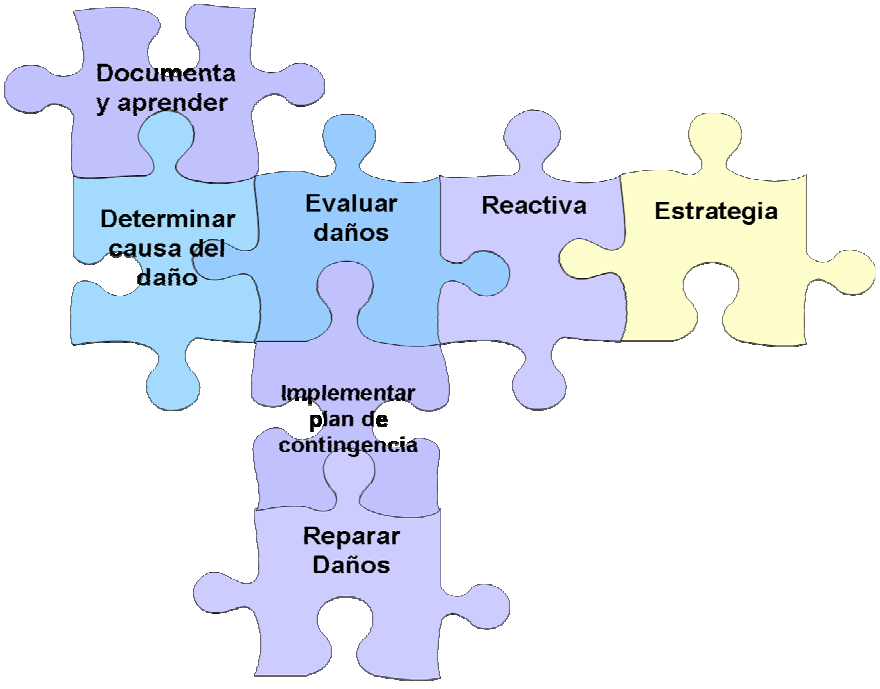
- ☑ Evaluar planes de recuperación de desastres para determinar su adecuación con el fin de proporcionar un entorno operativo temporal o a largo plazo. Los planes de recuperación de desastres deben incluir la prueba de los niveles de seguridad necesarios, con el fin de que el personal de seguridad pueda ver si siguen exigiendo la seguridad en todo el proceso de recuperación o en operaciones temporales y el traspaso de la organización otra vez a su sitio de procesamiento original o a un sitio nuevo.

Redactar un documento detallado que describa los distintos descubrimientos en las tareas anteriores. El documento debe mostrar:

- ☑ Todos los casos para probar el plan de contingencia.
- ☑ El impacto de las dependencias y de la ayuda planeada de fuera de la organización, y las dificultades que la obtención de los recursos esenciales tendrán en el plan.
- ☑ Una lista de prioridades observadas en las operaciones de recuperación y el fundamento para establecerlas.

EMBONADO DE LA SEGUNDA ESTRATEGIA

La **estrategia reactiva** se implementa cuando ha fallado la estrategia proactiva y define los pasos que deben adoptarse después o durante un ataque. Ayuda a identificar el daño causado y los puntos vulnerables que se explotaron en el ataque, a determinar por qué tuvo lugar, a reparar el daño que causó y a implementar un plan de contingencia, si existe. Tanto la estrategia reactiva como la proactiva funcionan conjuntamente para desarrollar políticas y controles de seguridad con el fin de reducir los ataques y el daño que causan. Éstos son los pasos de la estrategia reactiva:



1. Evaluar Daños.
2. Determinar la causa del daño.
3. Reparar daños.
4. Documentar y aprender
5. Implementar plan de contingencia

El equipo de respuesta a incidentes debe incluirse en los pasos adoptados durante o después del ataque para ayudar a evaluarlo, a documentar el evento y a aprender de él.

En la figura 6.6 se muestra el embonado parcial de la estrategia reactiva.

Figura 6.6. Armado parcial del rompecabezas de la Estrategia Reactiva

Evaluar el daño

Determine el daño causado durante el ataque. Esto debe hacerse lo antes posible para que puedan comenzar las operaciones de restauración. Si no se puede evaluar el daño a tiempo, debe implementarse un plan de contingencia para que puedan proseguir las operaciones comerciales y la productividad normales.

Determinar la causa del daño

Para determinar la causa del daño, es necesario saber a qué recursos iba dirigido el ataque y qué puntos vulnerables se explotaron para obtener acceso o perturbar los servicios. Revise los registros del sistema, los registros de auditoría y las pistas de auditoría. Estas revisiones suelen ayudar a descubrir el lugar del sistema en el que se originó el ataque y qué otros recursos resultaron afectados.

Reparar el daño

Es muy importante que el daño se repare lo antes posible para restaurar las operaciones comerciales normales y todos los datos perdidos durante el ataque. Los planes y procedimientos para la recuperación de desastres de la organización (que se tratan en el documento acerca del diseño de la seguridad) deben cubrir la estrategia de restauración. El equipo de respuesta a incidentes también debe poder controlar el proceso de restauración y recuperación, y ayudar en este último.

Documentar y aprender

Es importante documentar el ataque una vez que se ha producido. La documentación debe abarcar todos los aspectos que se conozcan del mismo, entre los que se incluyen el daño que ha causado (en hardware y software, pérdida de datos o pérdida de productividad), los puntos vulnerables y las debilidades que se explotaron durante el ataque, la cantidad de tiempo de producción perdido y los procedimientos tomados para reparar el daño. La documentación ayudará a modificar las estrategias proactivas para evitar ataques futuros o mermar los daños.

Implementar un plan de contingencia

Si ya existe algún plan de contingencia, se puede implementar para ahorrar tiempo y mantener el buen funcionamiento de las operaciones comerciales. Si no hay ningún plan de contingencia, es necesario desarrollar un plan apropiado basado de la documentación del paso anterior.

VERIFICAR Y REALIZAR EL EMBONE ENTRE LAS PIEZAS RESTANTES

Revisar el resultado y hacer simulaciones

Un paso no menos importante en la estrategia de seguridad es revisar los descubrimientos establecidos en los primeros pasos (predicción del ataque). Tras el ataque o tras defenderse de él, es necesario revisar el resultado con respecto al sistema. La revisión debe incluir la pérdida de productividad, la pérdida de datos o de hardware, y el tiempo que se tarda en recuperarlos. Documente también el ataque y, si es posible, haga un seguimiento del lugar en el que se originó,

qué métodos se utilizaron para iniciarlo y qué puntos vulnerables se explotaron. Para obtener los mejores resultados posibles, realice simulaciones en un entorno de prueba.

Revisar la eficacia de las políticas

Si hay políticas para defenderse de un ataque que se ha producido, hay que revisar y comprobar su eficacia. Si no hay políticas, se deben redactar para aminorar o impedir ataques futuros.

Ajustar la política en consecuencia

Si la eficacia de la política no llega al estándar, hay que ajustarla en consecuencia. Las actualizaciones de las políticas debe realizarlas el personal directivo relevante, los responsables de seguridad, los administradores y el equipo de respuesta a incidentes. Todas las políticas deben seguir las reglas e instrucciones generales de la organización. Por ejemplo, el horario laboral puede ser de 8 a.m. a 6 p.m. Podría existir o crearse una política de seguridad que permita a los usuarios conectarse al sistema solamente durante este horario.

Considerando las estas tres piezas restantes podemos generar otra parte del rompecabezas donde se parte de un análisis de los ataques, amenazas o riesgos que pueden suscitarse en nuestra estrategia.

Así tenemos por diagrama el que se aprecia en la figura 6.7:

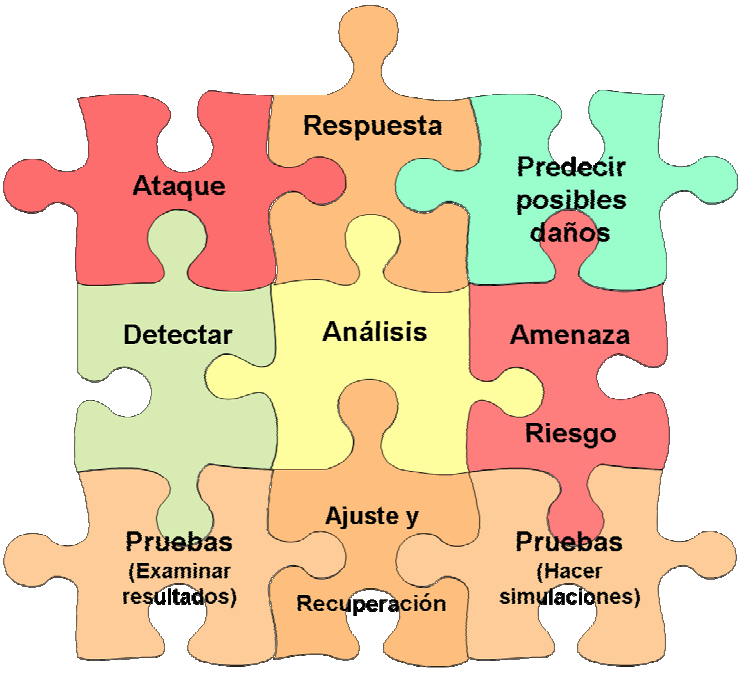


Figura 6.7. Rompecabezas parcial de las piezas restantes.

Con este último ejemplo quiero mostrar que realizar un diagrama de esta forma puede simplificar el comprender los conceptos y procedimientos al momento de establecer la seguridad en las aplicaciones o los sistemas. Una de las formas en que es posible "leer" parte de este diagrama es la siguiente: "En un **análisis** del sistema se **detecta** un **ataque** al cual es necesario

darle una **respuesta** pronta…", o bien otra de interpretarlo es: "en el **Ajuste** del sistema, que resultó de las **simulaciones**, es posible realizar un **análisis** que permita establecer las **amenazas** y de esta manera **predecir** los posibles daños y realizar una **respuesta** al momento de ser atacado". Así cualquiera de los subgrupos de piezas en el rompecabezas puede ser leído o interpretado, de ser necesario se puede especificar a cada una de las amenazas o ataques que se desean estudiar. El crear una estrategia de esta manera, permite mejorar y ajustar las características y aspectos en cualquier momento, bebido a que se va creando y analizando por partes.

EL FIN DEL ROMPECABEZAS

No debe olvidarse que es necesario embonar todas las piezas y grupo de piezas (herramientas, técnicas, subestratégias) que resulten de este análisis, con la finalidad de concentrar todas las ideas y conceptos útiles para mejorar y aplicar la estrategia de seguridad. Como parte de la metodología sólo presento de manera genérica algunas de las piezas que pueden componer el gran rompecabezas de la estrategia de la seguridad. A diferencia de un juego de rompecabezas, la base de la estrategia que se platea aquí puede expandirse para hacerlo más completo y adecuado a las necesidades de los programadores y administradores de sistemas. A continuación se muestra en la figura 6.8 la estrategia de seguridad integrada de cada aspecto que se estudió y explicó:

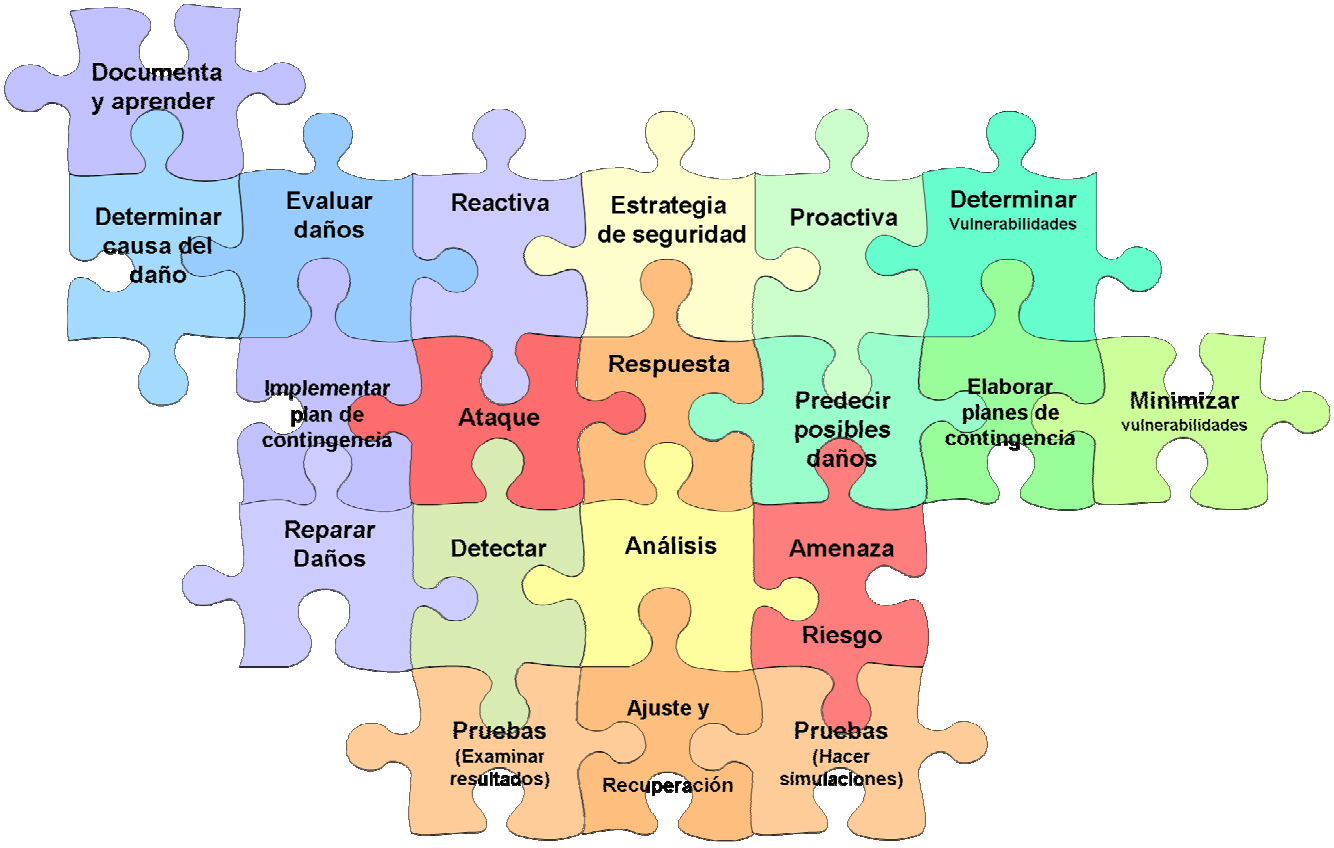


Figura 6.8. Rompecabezas final de la Estrategia de seguridad

6.5. ¿CÓMO SE APLICA UNA ESTRATEGIA DE SEGURIDAD?

6.5.1. EJEMPLOS DE APLICACIÓN DE LA ESTRATEGIA DE SEGURIDAD

Ejemplo 1: De una amenaza no intencionada

Un empleado, no desea perder la información que ha guardado en el disco duro. Desea hacer una copia de seguridad de esta información, así que la copia a su carpeta particular del servidor que resulta ser también el servidor principal de aplicaciones de la organización. No se han definido cuotas de disco para las carpetas particulares de los usuarios que hay en el servidor. El disco duro tiene 6.4 GB de información y el servidor tiene 6.5 GB de espacio libre. El servidor de aplicaciones deja de responder a las actualizaciones y peticiones porque se ha quedado sin espacio en el disco. El resultado es que se deniega a los usuarios los servicios del servidor de aplicaciones y la productividad se interrumpe. A continuación con la figura 6.9, se explica la metodología que se debería haber adoptado antes de que el usuario decidiera hacer una copia de seguridad de su disco duro en su carpeta particular.

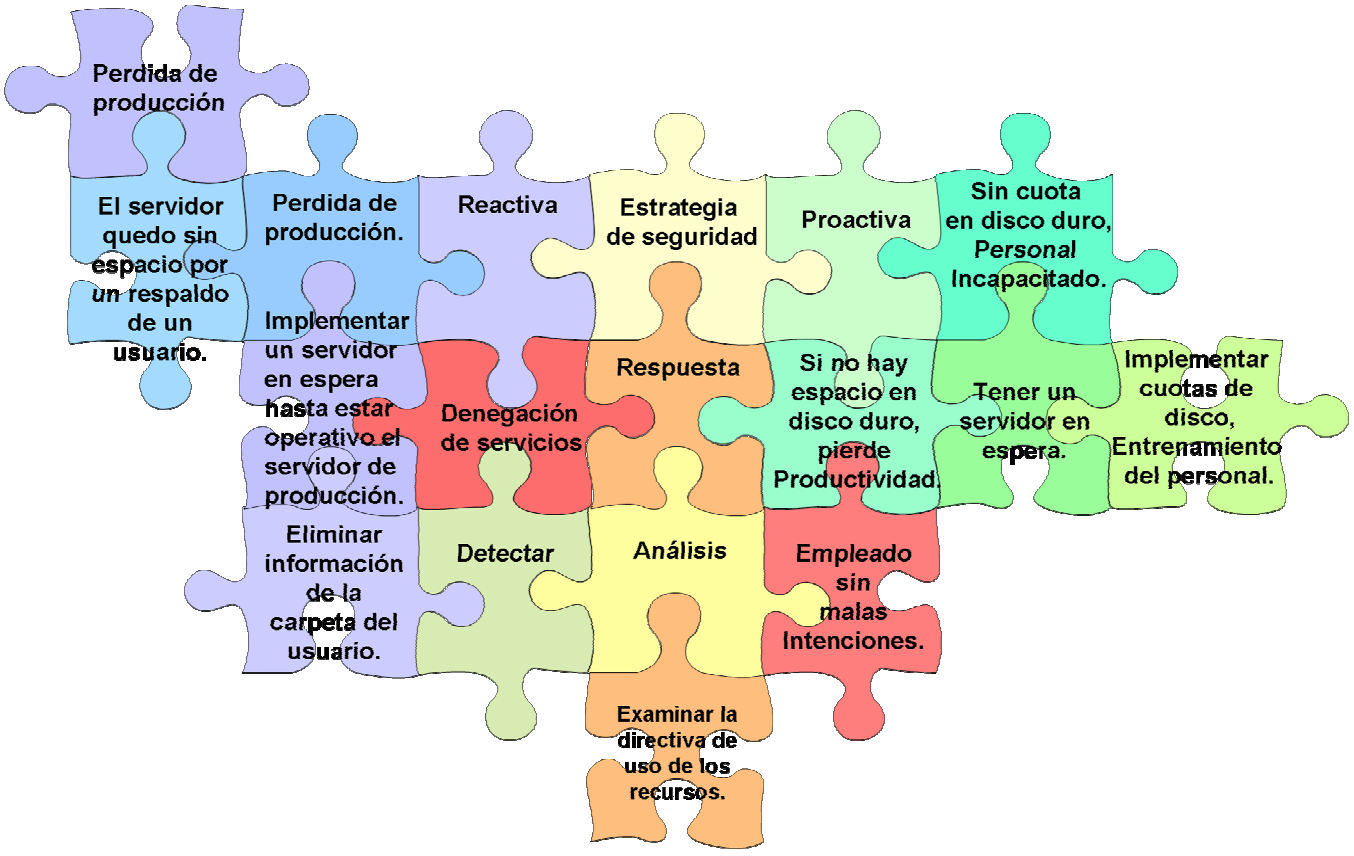


Figura 6.9. Ejemplo de una estrategia

De esta forma se tiene el siguiente análisis:

Predecir Ataques y evaluar riesgos de la denegación de servicios cuando el usuario abusa de los recursos.

Amenaza: empleado sin malas intenciones.

Estrategia proactiva:

- Daños posibles: si no hay espacio en disco duro, se puede perder productividad
- Vulnerbilidades: sin cuota en disco duro, sin entrenamiento del personal.
- Minimizar Vulnerabilidades: Implementar cuota de disco y elaborar un entrenamiento del personal.
- Plan de contingencia: Tener posiblemente un servidor de espera.

Estrategia reactiva:

- Daños: Pérdidas de producción
- Causa: El servidor quedo sin espacio en disco porque el usuario copió todo el disco duro a la carpeta particular.
- Reparación: Eliminar información de la carpeta particular del usuario
- Plan de contingencia: Implementar un servidor en espera hasta que vuelva a estar operativo el servidor de producción.

Examinar resultados: Perdida de resultados

Efectividad de la política: Examinar la directiva de uso de los recursos y las políticas de entrenamiento del personal.

Ejemplo 2: amenaza malintencionada (agresor externo)

Virus nuevo que altera los sistemas de correo electrónico de todo el mundo.

El resultado del análisis de riesgo es:

Predecir ataques y evaluar riesgos si se deniegan servicios al utilizar correo electrónico para comercio electrónico y otras funciones empresariales, y el servidor de correo no funciona.

Amenaza: Atacante malintencionado

Método de ataque: Virus de correo electrónico.

Estrategia proactiva:

- Daños: Si se tiene el correo electrónico, también se detiene la productividad.
- Vulnerabilidades: No hay búsqueda o firmas de virus en la base de datos.
- Minimizar Vulnerabilidad: Implementar la exploración antivirus o actualizar la base de datos de firma de virus poniéndose en contacto con el proveedor.
- Plan de contingencia: Tener un servidor en espera.

Estrategia reactiva:

- Daños: Perdida de producción.
- Causas de daño: El servidor de correo tuvo un error debido a un virus de correo electrónico.

- Reparar daños: Implementar un servidor en espera hasta que vuelva a estar operativo es servidor de producción.
- Planes de contingencia: Implementar un servidor en espera hasta que vuelva estar en operación el servidor de producción.

Eficacia de la directiva: Examinar la directiva de detección y exploración de virus.

Ejemplo 3: amenaza malintencionada (agresor interno)

Un empleado, trabaja para una empresa que diseña naves espaciales. Una organización competidora se pone en contacto con el empleado para ofrecerle una gran suma de dinero por robar información del diseño de la organización más reciente. El empleado no tiene los derechos necesarios para tener acceso a la información. En una conversación telefónica con un usuario que tiene derechos de acceso, simula que es uno de los administradores. El empleado dice al usuario que está realizando un trabajo administrativo habitual y le solicita su nombre de usuario y contraseña para comprobarlos con los registros del servidor. El usuario accede a dar al empleado dicha información.

El resultado del análisis es el siguiente:

Predecir ataques y evaluar riesgos para la información que se puede robar mediante ingeniería social.

Tipo de amenaza: Empleado atacante malintencionado

Método de ataque: Ingeniería Social

Estrategia proactiva:

- Daños: si se roba la información confidencial, hay pérdidas de beneficios
- Vulnerabilidades: Conciencia por la seguridad entre los empleados.
- Minimizar vulnerabilidades: Implementación de entrenamiento sobre conciencia de la seguridad

Estrategia reactiva:

- Daños: Perdidas de beneficios e información confidencial
- Causas de daño: El empleado reveló un nombre de usuario y una contraseña.
- Reparar daños: Implementar entrenamiento sobre conciencia por la seguridad y ponerse en contacto con las autoridades correspondientes.

Resultados: La información confidencial se puede robar por la existencia de poca conciencia por la seguridad, lo que produciría perdías de ingresos.

Eficacia de política: Examinar la directiva de entrenamiento sobre conciencia por la seguridad.

Ejemplo 4: amenaza no intencionada (desastre natural)

La organización XYZ no tiene sistemas de detección y protección contra incendios en la sala de servidores. Un programador de los sistemas de la organización deja un par de manuales encima del aparato de aire acondicionado. Durante la noche el acondicionador de aire se caliente y comienza un incendio que arrasa la sala de servidores.

Resultado de la estrategia de seguridad:

Predecir ataque y evaluar riesgos de un Incendio

Tipo de amenaza: natural, incendio.

Método de ataque: Ninguno

Estrategia proactiva:

- Daños: Perdida de información, *hardware*, productividad.
- Vulnerabilidad: No hay protección o detección contra incendios instalada, o los sistemas de detección de incendios no funcionan correctamente.
- Minimizar Vulnerabilidades: Implementar políticas para la prevención y/o detección de incendios, mantenimiento periódico de los sistemas de protección contra incendios.
- Plan de contingencia: Asegúrese de que se realizan periódicamente copias de seguridad y se almacenan fuera de las innataciones. Si es posible, tener hardware de repuesto.

Estrategia reactiva:

- Daños: Perdidas de información, hardware y productividad.
- Causas de daño: Incendio causado por el bloqueo del aparato de aire acondicionado
- Reparar daños: Volver a construir los servidores y restaurar las copias de seguridad más recientes.

Resultados: Los incendios pueden tener efectos desastrosos sobre los sistemas informáticos.

Eficacia de la directiva: Examinar o implementa directivas de detección de incendios.

6.5.2. IMPLEMENTACIÓN DE UN SISTEMA SEGURO

Una vez que se realice un análisis de la amenazas y riegos, con una adecuada estrategia de seguridad, del sistema o aplicación que se este implementando se recomienda llevar a cabo una ***etapa de depuración***[22] (en un equipo de prueba y no en el de implementación).

En la característica de copia de proyectos de Visual Studio .NET se incluye una opción que permite implementar una aplicación con un archivo de configuración (*Web.config*) diferente al utilizado durante el desarrollo. Es probable que la depuración esté habilitada en el archivo de desarrollo y, si éste se implementa, permitiría a los usuarios examinar la pila de llamadas al

[22] Si debe realizarse la depuración en un servidor de implementación, sólo instale el componente de depuración remota y desinstálelo cuando se haya finalizado el proceso, además es recomendable dejar al servidor sin conexión mientras se realiza la depuración.

originarse una excepción. Se recomienda llevar a cabo la implementación con un archivo de configuración independiente que no permita la depuración.

El último elemento de las estrategias de seguridad debe ser las pruebas y el estudio de sus resultados, se lleva a cabo después de que se han puesto en marcha las estrategias reactiva y proactiva. La realización de ataques simulados en sistemas de pruebas o en laboratorios permite evaluar los lugares en los que hay puntos vulnerables y ajustar las políticas y los controles de seguridad en consecuencia.

Estas pruebas no se deben llevarse a cabo en los sistemas de producción real, ya que el resultado puede ser desastroso. La carencia de laboratorios y equipos de pruebas a causa de restricciones presupuestarias puede imposibilitar la realización de ataques simulados. Para asegurar los fondos necesarios para las pruebas, es importante que los directivos sean conscientes de los riesgos y consecuencias de los ataques, así como de las medidas de seguridad que se pueden adoptar para proteger al sistema, incluidos los procedimientos de las pruebas. Si es posible, se deben probar físicamente y documentar todos los casos de ataque para determinar las mejores políticas y controles de seguridad posibles que se van a implementar.

Determinados ataques, por ejemplo desastres naturales como inundaciones y rayos, no se pueden probar, aunque una simulación servirá de gran ayuda. Por ejemplo, se puede simular un incendio en la sala de servidores en el que todos los servidores hayan resultado dañados y hayan quedado inutilizables. Este caso puede ser útil para probar la respuesta de los administradores y del personal de seguridad, y para determinar el tiempo que se tardará en volver a poner la organización en funcionamiento.

La realización de pruebas y de ajustes en las políticas y controles de seguridad en función de los resultados de las pruebas es un proceso iterativo. Nunca termina, ya que debe evaluarse y revisarse de forma periódica para poder implementar mejoras.

La mayoría de las aplicaciones sólo necesitan que *.NET Framework* se encuentre instalado en el servidor. Si se instala Visual Studio .NET o sus componentes de servidor en el equipo de implementación, aparecerán en éste los grupos Depuradores y Programadores de VS, y debe restringirse los miembros de sus usuarios. Asimismo, es aconsejable deshabilitar el descubrimiento dinámico.

La ***implementación de medidas de seguridad***, es un proceso Técnico-Administrativo. Como este proceso debe abarcar TODA la organización, sin exclusión alguna, ha de estar fuertemente apoyado por el sector gerencial, ya que sin ese apoyo, las medidas que se tomen no tendrán la fuerza necesaria.

Se debe tener en cuenta que la implementación de Políticas de Seguridad, trae varios tipos de problemas que afectan el funcionamiento de la organización. La implementación de un sistema de seguridad conlleva a incrementar la complejidad en la operatoria de la organización, tanto técnica como administrativamente.

Por esto, es necesario sopesar cuidadosamente la ganancia en seguridad respecto de los costos administrativos y técnicos que se generen. Es fundamental no dejar de lado la notificación a

todos los involucrados en las nuevas disposiciones y, darlas a conocer al resto de la institución con el fin de otorgar visibilidad a los actos de la administración. Luego de evaluar elementos y establecer la base del análisis, se originan un programa de seguridad, el plan de acción y las normas y procedimientos a llevar a cabo.

Para que todo lo anterior llegue a buen fin debe realizarse un ***control periódico*** de estas estrategias, que asegure el fiel cumplimiento de todos los procedimientos enumerados.

Con el objeto de confirmar que todo lo creado funciona en un marco real, se realiza una ***simulación de eventos*** y acontecimientos que atenten contra la seguridad del sistema. Esta simulación y los casos reales registrados generan una realimentación y revisión que permiten adecuar los sistemas generadas en primera instancia.

Es importante destacar que la Seguridad debe ser considerada desde la fase de diseño de un sistema, si la seguridad es contemplada luego de la implementación del mismo el personal se enfrentará con problemas técnicos, humanos y administrativos muchos mayores que implicaran mayores costos para lograr, en la mayoría de los casos, un menor grado de seguridad.

Existen otras acciones que un sistema digno de calificarse como seguro debe considerar y algunas de estas se muestran a continuación.

6.5.3. CONSIDERACIONES EN UN SISTEMA SEGURO

Debe tomarse la seguridad de los sistemas muy en serio. Todas las personas que pertenezcan al equipo de desarrollo, al equipo de dirección del proyecto, deben compartir la creencia de que el sistema será atacado tarde o temprano y, por este motivo, necesitará invertir en seguridad.

Los sistemas que estén conectados a una intranet padecen riesgos de seguridad. Personas contrariadas, empleados despedidos que aún tienen acceso al sistema, todos pueden atacar de forma intencionada al sistema. Un operador puede cometer un error que cueste millones de pesos a la empresa, algo que un buen sistema de seguridad puede detener a tiempo.

No debe dejarse la seguridad para el final del proyecto. La expresión "Primero desarrollemos las funciones y dejemos la seguridad para el final", están indicando claramente que se 'Olvidarán de la seguridad". Cuando esto sucede, la seguridad se recorta o se relega a una posición de importancia menor. La mejor solución es pensar en la seguridad como en otra función del diseño, o como una parte esencial de otras funciones; tiene que diseñarse y desarrollarse la seguridad a la vez que los demás componentes.

Resulta mucho más barato realizar los cambios en el servidor, implementar el acceso a SQL Server y añadir la seguridad basada en roles al principio del proyecto que al final.

Cuando se comienza a diseñar la seguridad, debe conocerse el nivel de seguridad que se desea asignar al sistema. Por ejemplo, el patrocinador del proyecto puede querer limitar el esfuerzo dedicado a mejorar la segundad a no más del diez por ciento del total del tiempo de desarrollo, o tal vez confíe lo suficiente en el equipo para permitir un libre acceso a la base de datos. Se trata de la última palabra del patrocinador, pero sea cual sea la decisión que tome, debe conocerse y

registrarse explícitamente el nivel de seguridad que se desee alcanzar (de esta forma facilitará el desarrollo y la toma de decisiones durante todo el ciclo de vida del proyecto).

Todo el equipo de trabajo comprometido con la seguridad. Resulta esencial que todo el equipo del proyecto este comprometido con la seguridad, es decir que sepa cómo se diseñan y desarrollan sistemas seguros. Si la gente no conoce las técnicas de seguridad no las utilizarán, y la revisión del código no siempre solucionará todos los problemas. Resulta más sencillo pensar en términos de seguridad desde el principio que intentar resolver posteriormente los problemas realizando revisiones de la seguridad. Para crear un sistema seguro, es necesario que todo el equipo se encuentre comprometido, que piense en términos de seguridad, que escriba código seguro y que esté siempre alerta ante la aparición de posibles problemas de seguridad.

La forma más sencilla de hacer que todo el equipo se sienta comprometido es dedicar unos días a formar al personal. Durante este tiempo, puede realizarse una introducción a los desbordamientos de búfer, la validación de las entradas, el control de excepciones y otras técnicas importantes en la programación segura.

Debe diseñarse una arquitectura segura, poniendo los componentes clave en equipos independientes. De esta forma es posible evitar puntos de fallo único. Para cumplir con este punto es posible recurrir a los diseños gráficos de los sistemas a partir de las necesidades del sistema y de la institución.

Si se está desarrollando una aplicación Web y se dispone de un control limitado sobre la arquitectura, existen cuatro puntos que pueden llevarse a cabo para detener la mayoría de los ataques provenientes de la Web. En otras palabras, si no hace nada más, deberá llevarse a cabo lo siguiente:

- Instalar un cortafuegos y limitar el tráfico únicamente a los puertos que su aplicación necesite.
- Instalar URLScan en la máquina IIS.
- Aplicar todos los parches de segundad emitidos por Microsoft tan pronto como se encuentren disponibles.
- Instalar un analizador de virus y mantener actualizada la base de datos de firmas de virus.

Realizar modelado de riesgos y amenazas. En este capítulos se analizan el modelado de riesgos que se utilizan para determinar cuáles son las vulnerabilidades de seguridad de un sistema. Este proceso se puede llevar a cabo durante cualquier fase de un proyecto y presenta muchas ventajas al realizar este análisis de riesgos durante la fase de diseño del sistema. El análisis de riesgos siempre sigue la misma trayectoria:

1. Identificar a los posibles intrusos.
2. Analizar las posibles formas en que un intruso puede atacar al sistema y generar una lista de vulnerabilidades.
3. Clasificar las vulnerabilidades en función del riesgo, teniendo en cuenta que la importancia del riesgo depende del daño potencial y de la probabilidad del ataque.
4. Elegir la acción que se debe realizar para cada vulnerabilidad. Para las vulnerabilidades que tengan un alto riesgo se debe resolver el problema o modificar la arquitectura para que

desaparezca la vulnerabilidad. Para las vulnerabilidades que tengan un bajo riesgo la respuesta puede ser eliminar o neutralizar la vulnerabilidad, o ignorar, suavizar o limitar el daño que puede provocar dicha vulnerabilidad. La acción que se lleve a cabo depende del nivel de seguridad que se decida utilizar en el sistema. Si no se puede anular o neutralizar la vulnerabilidad, siempre será mejor conocer su existencia durante la fase de diseño que descubrirla cuando un intruso ataque con éxito el sistema.

Cuando se esté realizando el modelo de riesgos siempre resulta útil analizar las amenazas potenciales que se ciernen sobre todo el sistema y que no afectan sólo a la aplicación que esté creando. Debe tenerse en cuenta otras aplicaciones, la arquitectura del dominio y todos los aspectos del sistema, Si se dispone de un presupuesto limitado y tiene que elegirse entre resolver problemas relacionados con el bloqueo, la autenticación o la autorización, dé la máxima prioridad al bloqueo sobre cualquier otro tema. El bloqueo debe tener la mayor prioridad porque si un intruso consigue introducirse en el sistema superando los procesos de autenticación y autorización, invertir tiempo en la mejora de otras tecnologías resultará infructuoso.

Usar funciones predefinidas por el lenguaje de desarrollo o el sistema operativo, diseñar el sistema para sacar el máximo partido a la seguridad intrínseca de Windows. Windows dispone de buenas funciones de autenticación, autorización y cifrado, que han superado ya procesos importantes de análisis de riesgos, revisiones de seguridad y una gran cantidad de pruebas. Siempre que sea posible, utilice *estas funciones predefinidas* en lugar de crear las propias.

Existe un error de concepto muy habitual. En general se piensa que hacer un sistema más seguro es hacerlo más complicado. Añadir seguridad a una aplicación no significa necesariamente agregar una segunda pantalla de inicio de sesión. Si una función de seguridad no tiene un uso sencillo, los usuarios intentarán evitar su empleo. Por el contrario, la mejor idea es distribuir la seguridad de forma uniforme por la aplicación para que el usuario apenas la perciba. Todo esto hará que la ***aplicación sea más sencilla y más fácil de utilizar***, y conseguirá que la seguridad no afecte a otras funciones.

Protegerse de las puestas traseras. Por otra parte resulta frecuente que los diseñadores y administradores del sistema deseen *introducir puertas traseras*[23] en una aplicación.

Existen tres problemas principales relacionados con el empleo de las puertas traseras: 1) ocultan problemas de uso, si el sistema tiene un empleo poco apropiado para los diseñadores, es casi seguro que también los usuarios lo encontrarán incómodo. 2) nunca se eliminan del todo, si los diseñadores pueden añadir puertas traseras a su antojo, las puertas traseras seguirán existiendo cuando se comercialice el producto, creando un fallo en la seguridad. La mejor opción es no tener que añadir puertas traseras. 3) ocultan defectos en la seguridad, con mucha frecuencia, los diseñadores descubren numerosos defectos durante el desarrollo de la aplicación. Si los

[23] Una puerta trasera es un tipo de acceso directo, un camino para saltarse todas las medidas de seguridad. Entre los ejemplos de puertas traseras cabe citar las opciones secretas de la línea de comandos que iniciarán la aplicación en un modo inseguro, cuentas de inicio de sesión con plenos poderes y cuentas de inicio de sesión con contraseñas en blanco o fáciles de recordar. Las puertas traseras se suelen crear para acelerar la implementación o como una salvaguarda para el caso de que los diseñadores bloqueen de forma inadvertida el sistema durante el desarrollo

diseñadores utilizan puertas traseras para acceder al sistema, no estarán utilizando las funciones de seguridad.

En términos de seguridad, *agregar un firewall* [24] o cortafuegos de red es un hardware o software que filtrará la información que lo atraviesa. Por ejemplo, puede ponerse un cortafuegos entre un servidor Web e Internet. También puede llevarse a cabo este mecanismo de forma lógica, utilizando un software tal como el cortafuegos que incluye Windows XP.

Por ultimo, *diseñar una aplicación pensando en el mantenimiento* significa diseñarla con la idea de que incluso después de que el programa haya entrado en la fase productiva, tendrá que realizarse nuevas tareas de desarrollo. Como por ejemplo: Aplicar paquetes de servicio y parches, llevar a cabo actualizaciones de la aplicación, empleo de registros de supervisión (diseñar algunas funciones de administración remota), diseño de un interruptor de apagado (método sencillo para apagarlo todo) así si un intruso ha conseguido introducirse en su sistema, sea posible apagar las máquinas e impedir que el intruso cause mayores daños.

6.6. CONCLUSIÓN

En este capítulo se han analizado las estrategias, los métodos, los pasos, etc., que deben darse para diseñar una aplicación segura. Se han cubierto todas las etapas del desarrollo real de la aplicación. Se presentaron los conceptos clave que deben recordarse como lo son que: debe comenzar a pensarse en seguridad desde el principio del proyecto, que la seguridad implica la colaboración de todo el equipo de trabajo durante todo el ciclo de vida del desarrollo del software.

La seguridad de los recursos conlleva un proceso que abarca el ciclo de desarrollo completo y en el que entran en juego varias tecnologías diferentes. Pueden conseguirse aplicaciones con un nivel de seguridad elevado si las tareas de diseño, prueba e implementación se desarrollan con precaución. También es posible aprovechar las tecnologías de seguridad que ofrecen Microsoft, el sistema operativo y los exploradores Web para garantizar la seguridad de las aplicaciones.

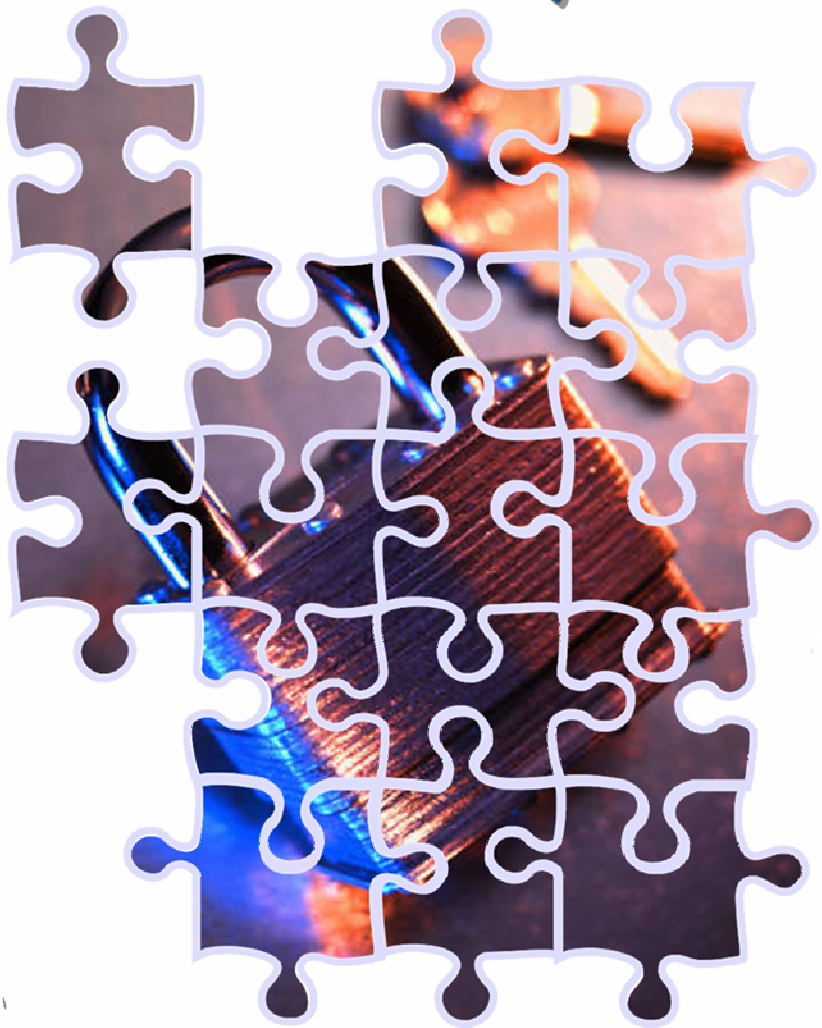
Todo lo anterior permite darse cuenta que:

- ☑ La información de los sistemas informáticos están en constante peligro.
- ☑ La implementación de una solución de seguridad es siempre un equilibrio entre la seguridad y la facilidad de uso.
- ☑ La creación de un entorno informático seguro es responsabilidad de toda la institución.
- ☑ El éxito de una estrategia de seguridad dependerá de la planeación, técnicas y tecnologías utilizadas.

[24] Los cortafuegos se utilizan para restringir el flujo de paquetes TCP y UDP basándose en el puerto que utilice. Un puerto es un número de 16 bits incluido en un paquete TCP y UDP. Cada servicio utiliza un puerto distinto. Por ejemplo, las comunicaciones basadas en Secure Sockets Layer (SSL) utilizan el puerto 443 y SQL Server utiliza el puerto 1433.

Capítulo 7

Herramientas y Técnicas para Asegurar los programas y sistemas de Visual Basic .NET



7. HERRAMIENTAS Y TÉCNICAS PARA ASEGURAR PROGRAMAS Y SISTEMAS DE *VISUAL BASIC .NET*

El presente capítulo por si mismo esta diseñado para que se convierta en una herramienta para los programadores que quieren trabajar y diseñar con "seguridad" sus aplicaciones y sistemas. Se trata del diseño de una guía que permita al programador conocer y aplicar las técnicas y herramientas que existen entorno a la seguridad y que resultan muy útiles para completar los sistemas y aplicaciones.

Contemplar las herramientas y técnicas a la par del diseño y la programación en las aplicaciones forma parte de una estrategia de programación de seguridad, porque al apoyarse de dichas técnicas y herramientas es posible crear sistemas o aplicaciones robustas y seguras.

Al final del capítulo anterior una de las recomendaciones es utilizar las funciones y herramientas predefinidas en lugar de las propias. Con el capítulo 7 quiero ser congruente y mostrar dichas funciones y herramientas que faciliten la tarea de programación en las aplicaciones realizadas en Visual Basic .NET y un poco mas haya, las realizadas sobre el sistema operativo Windows.

La finalidad de este capítulo no es la de forzar a que se utilicen las herramientas y funciones que muestro, quiero que a partir de esta guía sea posible construir un juicio en los programadores y desarrolladores sobre la utilidad que llega a representar el uso e implementación de las técnicas, funciones y herramientas en los sistemas donde la prioridad es la seguridad.

Para que esta guía pueda convertirse en una verdadera herramienta es necesario utilizarla con una técnica que permita su consulta y aplicación. La descripción de esta técnica resulta de la comprensión de lo que se conoce como las **Capas de Seguridad**.

Para comprender este concepto es necesario explicar que las restricciones de seguridad del sistema operativo constituye el conjunto final de permisos que debe tenerse en consideración cuando se desarrolla un sistema o aplicación. Por ejemplo, puede definirse un rol en la aplicación, por ejemplo el de director de la institución, a su vez se le proporciona un permiso para guardar información en el disco duro. La aplicación, desde el punto de vista de acceso al código, puede otorgar todos los derechos de acceso para realizar ciertas operaciones de Visual Basic .NET como escribir en el disco determinada información. Esta acción depende del lugar en el que se desea almacenar el archivo, dado que el sistema operativo puede negar el permiso necesario para realizar la operación. Si el director intenta guardar la información en una carpeta a la que no tiene derechos de acceso (a partir de los permisos de carpeta del sistema operativo), la operación no podrá concluirse y se generará un error cuando se ejecute el código. Lo anterior muestra el trayecto en las capas de seguridad o la serie de comprobaciones de cualquier acción que se intenta llevar a cabo en la aplicación o el sistema.

Generalizando la idea anterior es posible presentarlo en los diagramas de las figura 7.1.

Figura 7.1 Diagramas de Capas de seguridad

En estos diagramas se presenta las capas por las cuales una acción debe pasar desde el punto de vista de seguridad.

Tomando el modelo de las capas de seguridad se estructuran en este capítulo las herramientas y técnicas que permitirán completar la seguridad en un sistema como parte de una estrategia de seguridad.

En principio se presentan las técnicas y herramientas que permiten establecer las bases para asegurar la aplicación, como segunda parte se muestran las herramientas de seguridad presentes en el *.NET Framework* que complementan la seguridad implementada en la primera parte. En tercero y último lugar se presentan algunas de las herramientas que se encuentran dominadas por el sistema operativo, y que además de encargarse de robustecer el S.O. en cuestiones de seguridad, encargándose de proporcionar la confianza para ejecutar las aplicaciones y los sistemas en esta plataforma.

7.1. SEGURIDAD IMPLEMENTADA POR EL PROGRAMADOR

7.1.1. CONTROL DE EXCEPCIONES Y ERRORES

En el capítulo 5 ya se habló acerca de este tema, sin embargo en este capítulo se retomará y complementará la información, a fin de convertirla en una útil herramienta y técnica para los desarrolladores y programadores.

Los términos error y excepción se suelen utilizar indistintamente. De hecho, un error, que es un evento que sucede durante la ejecución del código, interrumpe el flujo normal del mismo y crea un objeto de excepción. Al producirse la interrupción, el programa intenta buscar un controlador de excepciones, que es un bloque de código que indica cómo reaccionar ante el problema y que ayudará a reanudar el flujo. Es decir, ***el error es el evento y la excepción es el objeto que crea dicho evento.***

Con la expresión "iniciar una excepción" los programadores indican que el método en cuestión encontró un error y reaccionó con la creación de un objeto de excepción que contiene información acerca del error y el momento y lugar donde sucedió. Entre los factores que causan el error y las posteriores excepciones se incluyen errores de usuario, de recursos y de lógicas de programación. Dichos errores se relacionan con el modo en que el código realiza una tarea determinada y no con el propósito de ésta.

7.1.1.1. EXCEPCIONES

Microsoft Visual Basic .NET ofrece dos formas de control de excepciones. La primera de ellas es el control no estructurado, que sigue las convenciones de control de excepciones de las versiones anteriores de Visual Basic. La segunda es el control estructurado.

Control estructurado frente al no estructurado y cuándo utilizar cada uno de ellos

El control de excepciones estructurado consiste en utilizar una estructura de control que contiene excepciones, bloques de código aislados y filtros para crear un mecanismo de control. Con ello se permite que el código realice una distinción entre diferentes tipos de errores y reaccione según las circunstancias. En el control de excepciones no estructurado una instrucción *On Error* al principio del código controla todas las excepciones.

El control estructurado resulta mucho más versátil, sólido y flexible que el no estructurado, por lo que en la mayoría de los casos se recomienda su uso. No obstante, el control no estructurado se puede utilizar cuando:

- Se actualiza una aplicación escrita en una versión anterior de Visual Basic.
- Se desarrolla una versión preliminar o borrador de una aplicación y no es importante que la ejecución del programa presente errores o que fracase estrepitosamente.
- Se conoce con anterioridad y exactamente la causa de la excepción.

No se puede combinar el control de excepciones de ambos tipos en la misma función. Si se utiliza una instrucción *On Error*, no se podrá emplear *Try...Catch* en la misma función.

Control de excepciones estructurado

Con este tipo de control se comprueban partes específicas del código y cuando se produce la excepción, el código del control de excepciones se adapta a las circunstancias causantes de dicha excepción. En aplicaciones de gran tamaño este método resulta mucho más rápido que el control de excepciones no estructurado; asimismo, permite una respuesta más flexible a los errores y una mayor fiabilidad de la aplicación.

La estructura de control *Try...Catch...Finally* resulta fundamental en el control de excepciones estructurado. Comprueba una parte del código, filtra las excepciones creadas mediante la ejecución de dicho código y reacciona de forma diferente en función del tipo de excepción iniciada.

Bloque *Try...Catch...Finally*

Las estructuras de control *Try...Catch...Finally* comprueban una parte de código y dirigen el modo en que la aplicación debe controlar las distintas categorías de error. Cada una de las tres partes que componen la estructura realiza una función específica en este proceso.

La instrucción *Try* proporciona el código en el que se están comprobando las excepciones.

La cláusula *Catch* identifica bloques de código que se encuentran asociados con excepciones específicas. Un bloque *Catch When* hace que el código se ejecute en circunstancias específicas. Una cláusula *Catch* sin otra *When* reacciona ante cualquier excepción. Por lo tanto, el código debe disponer de una serie de instrucciones *Catch...When* determinadas que reaccionen, cada una de ellas, ante un tipo específico de excepción, seguidas de un bloque general *Catch* que reaccione ante cualquier excepción que no se haya interceptado mediante las cláusulas *Catch...When* anteriores.

La instrucción *Finally* contiene código que se ejecuta independientemente de si ocurre o no una excepción en el bloque *Try*. Esta instrucción se ejecutará incluso después de *Exit Try* o *Exit Sub*. Este código suele realizar tareas de limpieza como el cierre de archivos o la eliminación de búferes.

El código que se muestra es el diseño de un controlador de excepciones simple en Visual Basic .NET.

```
Sub PruebaVBNET()
    Try
        ' Realizar una acción aquí que
        ' pueda generar un error.
    Catch
        ' Controlar excepciones que ocurren en
        ' el bloque Try aquí.
    Finally
        ' Ejecutar el código de limpieza aquí.
    End Try

End Sub
```

Código Fuente 7.A

Centrándome en las funciones de la cláusula Catch, esta puede adoptar tres formas: *Catch, Catch...As y Catch...When*.

Una cláusula *Catch* permite que el bloque de instrucciones asociado controle cualquier excepción. Las cláusulas *Catch...As* y *Catch...When* interceptan una excepción específica y permiten que el bloque de instrucciones asociado indique a la aplicación qué medidas adoptar. Asimismo, estas dos cláusulas se pueden combinar en una sola instrucción como, por ejemplo, *Catch ex As Exception When intResult <> 0*.

Si la excepción es el resultado de un error de recursos, se debe identificar el recurso en cuestión y proporcionar sugerencias para la solución de problemas. Si la excepción procede de un error de lógicas de programación, es muy probable que la cláusula permita que la aplicación se

cierre de la forma más elegante posible. Sin embargo, si un error de usuario ha sido la causa de la excepción, el código debe permitir al usuario corregirlo y poder continuar.

Las cláusulas *Catch* se comprueban en el orden en el que aparecen en el código. Por lo tanto, este tipo de cláusulas se debe enfocar desde lo específico a lo general a medida que progresan a través de la secuencia de código.

El objeto *Exception*

El objeto *Exception* proporciona información sobre cualquier excepción que se intercepte. Siempre que se inicia una excepción, se establecen las propiedades del objeto *Err* y se crea una nueva instancia del objeto *Exception*. Se deben examinar sus propiedades para determinar la ubicación del código, el tipo y el motivo de la excepción.

Algunas de las propiedades útiles del objeto *Exception* son:

La propiedad ***HelpLink*** puede contener una dirección URL que dirija al usuario a más información sobre la excepción.

Hresult obtiene o establece *HRESULT*, un valor numérico asignado a la excepción. Se trata de un valor de 32 bits que contiene tres campos: un código de gravedad, un código de servicio y un código de error. El primero de ellos indica si el valor devuelto representa información, una advertencia o un error. Con el código de servicio se identifica el área del sistema responsable de la excepción. El código de error es un número exclusivo asignado para representar al error.

La propiedad ***InnerException*** devuelve un objeto de excepción que representa una excepción que ya se encontraba en el proceso de control cuando se inició la actual. El código que controla la excepción externa puede utilizar la información de la interna para poder controlar la expresión externa con mayor precisión.

La propiedad ***Message*** contiene una cadena que es el mensaje de texto que informa al usuario de la naturaleza del error y de la mejor forma de solucionarlo. Durante la creación de un objeto de excepción se puede proporcionar la cadena que mejor se adapte a la excepción específica.

La propiedad ***Source*** obtiene o establece una cadena que contiene el nombre del objeto que inicia la excepción o el nombre del ensamblado donde ocurrió la misma.

StackTrace contiene un seguimiento de la pila que se puede utilizar para determinar en qué lugar del código ha ocurrido el error. Esta propiedad incluye todos los métodos llamados que precedieron a la excepción y los números de línea del código fuente donde se realizaron las llamadas.

La propiedad ***TargetSite*** obtiene el nombre del método que inició la excepción actual. Si el nombre no se encuentra disponible y el seguimiento de la pila no es ***Nothing***, la propiedad obtendrá el nombre del método de dicho seguimiento.

CREACIÓN DE EXCEPCIONES PROPIAS PARA EL CONTROL ESTRUCTURADO

Existen dos subclases definidas de excepciones en la clase de base *Exception*: *System.Exception* y *Application.Exception*.

System.Exception es la clase a partir de la cual *.NET Framework* deriva las clases de excepciones predefinidas de tiempo de ejecución en lenguaje común. Se inicia en el tiempo de ejecución en lenguaje común cuando ocurren errores no fatales. *System.Exception* no proporciona información sobre el motivo de la excepción.

Para obtener más información sobre las clases de excepciones predefinidas de tiempo de ejecución en lenguaje común, consulte la tabla 7.1 que contiene las clases de excepciones predefinidas, sus causas y sus clases derivadas.

CLASES DE EXCEPCIONES	MOMENTO DE INICIO	CLASE DERIVADAS
AppDomainUnloadedException	Se ha intentado tener acceso a un dominio de la aplicación descargado	Ninguna
ArgumentException	Se han proporcionado uno o varios argumentos no válidos a un método	ArgumentNullException ArgumentOutOfRangeException ComponentModel.InvalidEnum ArgumentException DuplicateWaitObjectException
ArithmeticException	Los errores ocurren en una operación aritmética o de conversión	DivideByZeroException NotFiniteNumberException OverflowException
ArrayTypeMismatchException	Se ha intentado almacenar un elemento de tipo incorrecto en una matriz	Ninguna
BadImageFormatException	La imagen de archivo de una DLL o el programa ejecutable no son válidos	Ninguna
CannotUnloadAppDomainException	Error al intentar descargar un dominio de la aplicación	Ninguna
ComponentModel.Design.Serialization.CodeDomSerializerException	La información de un número de línea está disponible para un error de serialización	Ninguna
ComponentModel.LicenseException	No se puede conceder una licencia a un componente	Ninguna
ComponentModel.WarningException	Una excepción se controla como advertencia en lugar de como error	Ninguna
Configuration.ConfigurationException	Error en un valor de configuración	Ninguna
Configuration.Install.InstallException	Error durante la fase de desinstalación o de confirmación de una instalación	Ninguna
ContextMarshalException	Error al intentar calcular la referencia de un objeto a través de un límite de contexto	Ninguna
Data.DataException	Los errores se generan al utilizar componentes ADO.NET	Data.ConstraintException Data.DeletedRowInaccessibleException Data.DuplicateNameException

		Data.InRowChangingEventException Data.InvalidConstraintException Data.InvalidExpressionException Data.MissingPrimaryKeyException Data.NoNullAlllowedException Data.ReadOnlyException Data.RowNotInTableException Data.StringTypingException Data.TypedDataSetGeneratorException Data.VersionNotFoundException
Data.DBConcurrencyException	Durante la operación de actualización, DataAdapter determina que el número de filas afectadas es igual a cero	Ninguna
Data.SqlClient.SqlException	SQL Server devuelve una advertencia o un error	Ninguna
Data.SqlTypes.SqlTypeException	Clase de excepción de base para Data.SqlTypes	Data.SqlTypes.SqlNullValueException Data.SqlTypes.SqlTruncateException
Drawing.Printing.InvalidPrinterException	Se ha intentado tener acceso a una impresora utilizando una configuración de impresora no válida	Ninguna
EnterpriseServices.RegistrationException	Se ha detectado un error de registro	Ninguna
EnterpriseServices.Serviced ComponentException	Se ha detectado un error en un componente de servicio	Ninguna
ExecutionEngineException	Existe un error interno en el motor de ejecución del tiempo de ejecución en lenguaje común	Ninguna
FormatException	El formato de un argumento no cumple con las especificaciones del parámetro del método invocado	Net.CookieException Reflection.CustomAttribute FormatException UriFormatException
IndexOutofRangeException	Se ha intentado tener acceso a un elemento de una matriz con un índice que está fuera de los límites de la misma	Ninguna
InvalidCastException	Conversión no válida o explícita	Ninguna
InvalidOperationException	Llamada al método no válida para el estado actual del objeto	Net.ProtocolViolationException Net.WebException ObjectDisposedException
InvalidProgramException	Los programas contienen metadatos o lenguaje intermedio de Microsoft no	Ninguna

	válidos	
IO.InternalBufferOverflowException	El búfer interno se desborda	Ninguna
IO.IOException	Error de E/S	IO.DirectoryNotFoundException IO.EndOfStreamException IO.FileLoadException IO.FileNotFoundException IO.PathTooLongException
Management.ManagementException	Error de administración	Ninguna
MemberAccessException	Error al intentar tener acceso a un miembro de clase	FieldAccessException MethodAccessException MissingFieldException MissingMemberException MissingMethodException
MulticastNotSupportedException	Se intentan combinar dos instancias de un tipo delegado que no se puede combinar donde ningún operando es una referencia nula	Ninguna
NotImplementedException	No se ha implementado un método u operación solicitada	Ninguna
NotSupportedException	No se admite el método invocado o se intenta leer, buscar o escribir en un flujo que no es compatible con la funcionalidad invocada	PlatformNotSupportedException
NullReferenceException	Se intenta eliminar una referencia nula a un objeto	Ninguna
OutOfMemoryException	No hay suficiente memoria para completar la ejecución de un programa	Ninguna
RankException	Una matriz se ha pasado a un método con el número incorrecto de dimensiones	Ninguna
Reflection.AmbiguousMatch Exception	El vínculo a un método tiene como resultado que varios métodos coincidan con los criterios de vinculación	Ninguna
Reflection.ReflectionType LoadException	El método Module.GetTypes determina que no se pueden cargar una o varias clases de un módulo	Ninguna
Resources.MissingManifest ResourceException	El ensamblado principal no contiene los recursos para la referencia cultural neutra, aunque éstos son necesarios debido a que no se encuentra un ensamblado satélite adecuado	Ninguna
Runtime.InteropServices.	Tipo de excepción de base para	ComponentModel.Design.

ExternalException	todas las excepciones de interoperabilidad COM y las de control de excepciones estructurado	CheckoutException ComponentModel.Win32Exception Data.OleDb.OleDbException Messaging.MessageQueueException Runtime.InteropServices.COMException Runtime.InteropServices.SEHException Web.HttpException
Runtime.InteropServices. InvalidComObjectException	Se utiliza un objeto COM no válido	Ninguna
Runtime.InteropServices. InvalidOleVariantTypeException	El contador de referencias encuentra un argumento de un tipo variant para el que no se pueden calcular las referencias en código administrado	Ninguna
Runtime.InteropServices. MarshalDirectiveException	El contador de referencias encuentra un atributo MarshalAsAttribute que no es compatible con el mismo	Ninguna
Runtime.InteropServices. SafeArrayRankMismatchException	El rango de una matriz de entrada SAFEARRAY no coincide con el especificado en la firma administrada	Ninguna
Runtime.InteropServices. SafeArrayTypeMismatchException	El tipo de una matriz de entrada SAFEARRAY no coincide con el especificado en la firma administrada	Ninguna
Runtime.Remoting.RemotingException	Error en ejecución remota	Runtime.Remoting.Remoting TimeOutException
Runtime.Remoting.ServerException	Se utiliza para comunicar excepciones cuando el cliente se conecta a una aplicación que no pertenece a .NET Framework y que no puede iniciar excepciones	Ninguna
Runtime.Serialization.SerializationException	Error durante la serialización o deserialización	Ninguna
Security.Crytography.CryptographicException	Error durante una operación de cifrado	Security.Cryptography. CryptographicUnexpected OperationException
Security.Policy.PolicyException	La directiva prohibe la ejecución del código	Ninguna
Security.SecurityException	Se ha detectado un error de seguridad	Ninguna
Security.VerificationException	Una directiva de seguridad requiere que el código sea seguro y el proceso de comprobación no puede realizar la operación	Ninguna
Security.XmlSyntaxException	Error de sintaxis en el análisis XML	Ninguna

ServiceProcess.TimeoutException	Ha caducado el tiempo de espera especificado	Ninguna
StackOverflowException	Desbordamiento de la pila de ejecución debido a que existen demasiadas llamadas al método pendientes	Ninguna
Threading.SynchronizationLockException	Se invoca un método sincronizado desde un bloque de código sin sincronizar	Ninguna
Threading.ThreadAbortException	Se ha realizado una llamada al método Abort	Ninguna
Threading.ThreadInterruptedException	Se ha interrumpido el subproceso mientras se encontraba en un estado WaitSleepJoin	Ninguna
Threading.ThreadStateException	Subproceso en una propiedad ThreadState no válida para la llamada al método	Ninguna
TypeInitializationException	Se inicia como un contenedor de una excepción que genera el inicializador de clase	Ninguna
TypeLoadException	Error al cargar el tipo	DllNotFoundException EntryPointNotFoundException
TypeUnloadedException	Se intenta tener acceso a una clase no cargada	Ninguna
UnauthorizedAccessException	El sistema operativo niega el acceso debido a un error de E/S o a un tipo específico de error de seguridad	Ninguna
Web.Services.Protocols.SoapException	El error ocurre como resultado de un método de servicio Web XML al que se llama a través de SOAP	Web.Services.Protocols. SoapHeaderException
Xml.Schema.XmlSchemaException		Ninguna
Xml.XmlException		Ninguna
Xml.Xpath.XpathException	Error al procesar una expresión Xpath	Ninguna
Xml.Xsl.XsltException	Error al procesar una transformación de lenguaje de hoja de estilo extensible (XSL)	System.Xml.Xsl.XsltCompileException

Tabla 7.1. En la tabla se incluyen las clases de excepciones predefinidas, sus causas y sus clases derivadas.

Es posible crear clases de excepciones propias en una aplicación heredándolas de la clase *Application.Exception*. Siguiendo las estructuras de las prácticas de codificación adecuadas haciendo terminar el nombre de clase de la excepción con la palabra "*Exception*"; por ejemplo, *OutOfMoneyException* o *TooMuchRainException*.

En el siguiente ejemplo se define una clase de excepción y tres constructores para la misma; cada uno de ellos adopta distintos parámetros[25].

```
Imports System
Public Class GardenException
    Inherits System.ApplicationException
    Public Sub New()
    End Sub
    ' Crea un Sub New para la excepción que permite establecer la _
    ' propiedad del mensaje cuando se inicia.
    Public Sub New(ByVal Mensaje As String)
        MyBase.New(Mensaje)
    End Sub
    ' Crea un Sub New que se puede utilizar cuando también se desea incluir _
    ' la excepción interna.
    Public Sub New(ByVal Mensaje As String, ByVal Inner As Exception)
        MyBase.New(Mensaje)
    End Sub
End Class
```

Código Fuente 7.B.

Este ***ejemplo*** de código es un bloque Try...Catch que en primer lugar busca *ArithmeticException* y, a continuación, las excepciones genéricas.

```
Imports System

Sub Main()
    Dim x As Integer = 0
    Try
        Dim y As Integer = 100 / x
    Catch ex As ArithmeticException
        MessageBox.Show(ex.Message)
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub 'Main
```

Código Fuente 7.C.

El siguiente ejemplo es un bloque Try...Catch...Finally asociado con una aplicación que abre un archivo para examinarlo. Se debe tener en cuenta que la instrucción Finally se ejecuta aunque Exit Sub aparezca en el código antes de la misma.

```
Imports System
Sub AbrirArchivo()
    Dim archivo As Object
    Try
        FileOpen(1, archivo, OpenMode.Input)
    Catch ex As Exception
        MsgBox(ex.Message)
        Exit Sub
    Finally
        FileClose(1)
    End Try
End Sub
```

Código Fuente 7.D.

[25] Cuando se utiliza la ejecución remota junto con excepciones definidas por el usuario, se debe comprobar que los metadatos para dichas excepciones se encuentren disponibles para el código que se ejecuta de forma remota, incluyendo aquéllas que ocurran a través de los dominios de la aplicación.

CONTROL DE EXCEPCIONES NO ESTRUCTURADO

El control de excepciones no estructurado se implementa utilizando el objeto Err y tres instrucciones: *On Error, Resume y Error*.

On Error establece un solo controlador de excepciones que intercepta todas las excepciones iniciadas; posteriormente, se podrá cambiar la ubicación del controlador, aunque sólo dispondrá de un controlador cada vez. Con este método se realiza un seguimiento de las excepciones iniciadas más recientemente, así como de la última ubicación del controlador de excepciones. En la entrada del método, tanto la excepción como la ubicación del controlador de excepciones se establecen en *Nothing*.

Para generar un error en tiempo de ejecución en el código se utiliza el método *Raise*. El método *Raise* del objeto *Err* se utiliza para generar errores en tiempo de ejecución. Siempre que ocurre una instrucción *Exit Sub*, *Exit Function*, *Exit Property*, *Resume* o *Resume Next* en una rutina de control de errores, las propiedades del objeto *Err* se restablecen a cero o a cadenas de longitud cero. El uso de algunas de ellas fuera de una rutina de control de errores no supondrá el restablecimiento de sus propiedades. Si es necesario realizar esta operación, se puede utilizar el método *Clear* para restablecer el objeto Err.

7.1.1.2. ERRORES

Los valores de las propiedades del ***objeto Err*** se determinan mediante el error que acaba de ocurrir. En la tabla 7.2 se incluyen estas propiedades y una breve descripción de cada una de ellas.

PROPIEDAD	DESCRIPCIÓN
Description	Mensaje de texto que proporciona una breve descripción del error.
Helpcontext	Entero que contiene el Id. de contexto para un tema en un archivo de ayuda.
Helpfile	Expresión de cadena que incluye la ruta completa al archivo de ayuda.
LastDLL	Código de error del sistema que se produce mediante una llamada a una biblioteca de vínculos dinámicos (DLL). Se trata de la última DLL a la que se llamó antes de que se produjera el error.
Number	Valor numérico que especifica un error.
Source	Expresión de cadena que representa al objeto o aplicación que generó el error.

Tabla 7.2. Errores

En el siguiente ejemplo se muestra el modo de utilización de algunas de estas propiedades en el control de errores no estructurado:

```
On Error Resume Next
Err.Clear
Err.Raise(33333)
Err.Description = "¡No se ha introducido ningún número!"
MsgBox(Err.Number)
MsgBox(Err.Description)
Msg = "Presione F1 o HELP para ver el tema " & Err.HelpFile & " para" & _
```

```
    " el siguiente HelpContext: " & Err.HelpContext
    MsgBox(Msg)
```

Código Fuente 7.E.

La Instrucción ***On Error GoTo*** activa una rutina para el control de excepciones y especifica la ubicación de la misma en el procedimiento. Se utiliza con una etiqueta o número de línea y dirige el código a una rutina de control de excepciones. Con -1, activa el control de errores en el procedimiento. Con 0, desactiva la excepción actual. Si no existe ninguna instrucción *On Error* y ningún método se encarga de controlar la excepción en la pila de llamadas actual, todos los errores en tiempo de ejecución que ocurran serán graves, por lo que la ejecución se detendrá y aparecerá un mensaje de error. En esta tabla 7.3., se muestran las formas en las que se puede utilizar la instrucción *On Error GoTo*.

INSTRUCCIÓN	FUNCIÓN
On Error Goto -1	Restablece el objeto **Err** a Nothing y desactiva el control de errores en la rutina.
On Error Goto 0	Restablece la última ubicación del controlador de excepciones a **Nothing** y desactiva la excepción.
On Error Goto <nombreEtiqueta>	Establece la etiqueta especificada como la ubicación del controlador de excepciones.
On Error Resume Next	Establece el comportamiento **Resume Next** como ubicación del controlador de excepciones más reciente.

Tabla 7.3 On Error

La instrucción ***Resume*** puede devolver por sí misma el control a la instrucción que ocasionó la excepción. La ejecución se reanudará en la misma línea que en un principio provocó la excepción.

La instrucción ***Resume Next*** reanuda la ejecución una vez que se ha producido una excepción. Especifica que, en caso de excepción, el control pasa a la instrucción inmediatamente siguiendo a la instrucción en la que ocurrió la excepción. Resume Next se puede utilizar para permitir errores sin importancia; ocurre un error en la instrucción pero la aplicación continúa ejecutándose y permite que el usuario corrija el error y prosiga. En el siguiente ***ejemplo*** se muestra un enfoque para el control de errores no estructurado. Cuando *sub CodigoFragil* localiza un error, la ejecución pasa a Salto que, a su vez, proporciona al usuario información sobre el error, en concreto sobre lo que se incluye en la propiedad *Description* del objeto *Err*:

```
Private Sub CodigoFragil()
    On Error GoTo Salto
    ' Código que realiza varias operaciones
    ' No continuar en el código de control de errores.
    Return
Salto:
    ' Proporcionar al usuario información sobre el error.
    MsgBox("Error inesperado:" & Err.Description)
    Return
End Sub
```

Código Fuente 7.F.

En el siguiente ejemplo se muestra cómo utilizar el objeto *Err* para crear un cuadro de diálogo de mensaje de error.

```
Dim MensajeError As String
' Crear un mensaje de error si se produce.
On Error Resume Next
Err.Raise (13) ' Generar error Type MisMatch.
' Comprobar si ha ocurrido un error. Si es así, mostrar mensaje.
If Err.Number <> 0 Then
   MensajeError = "Error # " & Str(Err.Number) & " generado por " _
      & Err.Source & vbCrLf & Err.Description
' Mostrar el mensaje como muy importante.
   MsgBox(MensajeError, MsgBoxStyle.Critical, "Error")
End If
```

Código Fuente 7.G.

Para finalizar con esta parte tan importante en la programación de sistemas se presenta la incorporación del control de excepciones estructurado que facilita a los programadores la administración de la notificación y generación de errores, así como la determinación de las causas de los mismos en tiempo de ejecución. *VB .NET* incluye una serie de características que lo convierten en un mecanismo de control más flexible que el utilizado en las versiones anteriores de Visual Basic.

El sistema de control de errores de .NET se basa, como ya mencione, en la clase *Exception*, que contiene información sobre el error actual, así como la lista de errores que lo puede haber desencadenado. Puede heredar de la clase *Exception*, creando sus propias excepciones con la misma funcionalidad que la clase base. Asimismo, si es necesario, puede crear funcionalidad extendida. La creación de su propia clase *Exception* permite a su código interceptar excepciones específicas, lo que aporta un alto nivel de flexibilidad.

Debido a que las clases de *.NET Framework* inician excepciones al encontrarse con errores en tiempo de ejecución, los programadores adquieren el hábito de interceptar y controlar las excepciones. De este modo, aumentan las posibilidades de que se realice con éxito el control de las excepciones iniciadas desde los componentes.

Tal vez si se desea extraer los errores de los procedimientos para indicar a los llamadores que se ha producido una excepción. Es probable que sólo desee pasar una excepción en tiempo de ejecución estándar proporcionada por *.NET Framework*, o bien, crear su propia condición de excepciones. En cualquier caso, debe utilizar la palabra clave *Throw* para extraer la excepción del bloque actual.

Puede utilizarse la palabra clave *Throw* de dos modos.

1. Devolver el error que acaba de ocurrir en el llamador desde un bloque *Catch*:

```
Catch e As Exception
  Throw
```

2. Generar un error desde cualquier código, incluido un bloque *Try*:

```
Throw New FileNotFoundException()
```

Código Fuente 7.H.

La primera técnica, iniciar la excepción que acaba de ocurrir, sólo funciona en los bloques *Catch*, como se vio en el tema de excepciones. La segunda de ellas, generar un nuevo error, funciona en cualquier bloque.

7.1.2. HERRAMIENTAS PARA VALIDACIÓN DE ENTRADA

Para las aplicaciones basadas en ***Windows Forms*** de *Visual Basic .NET* dispone de las siguientes herramientas que contribuyen a validar las entradas realizadas por los usuarios:

- Propiedad ***PasswordChar*** de *TextBox*: Puede utilizarse la propiedad *PasswordChar* para ocultar la entrada del usuario y, por tanto, impedir que las personas que merodeen por las cercanías puedan conocer las entradas confidenciales, tales como contraseñas o números de identificación personal (PIN).
- Propiedad ***MaxLength*** de *TextBox*: La propiedad *MaxLength* limita la entrada a un tamaño fijo o razonable. Si, por ejemplo, el campo apellido de una base de datos se encuentra limitado a 16 caracteres, se asigna el valor 16 a la propiedad *MaxLength* asociada con el cuadro de texto que representa al apellido.
- Propiedad ***CharacterCasing*** de *TextBox*: *CharacterCasing* ayuda a asignar un formato a la entrada. Por ejemplo, si se almacenan los ID de todos los productos en minúsculas sólo debería aceptar minúsculas cuando el usuario introduzca el identificador del producto.
- Validación de sucesos: Este suceso le ayuda a validar la entrada para las aplicaciones *Windows Forms*. El suceso *Validate* permite validar la entrada realizada por el usuario en un determinado control como, por ejemplo, un cuadro de texto, antes de que el foco de la aplicación se traslade a otro control.
- ***ErrorProvider***: Puede utilizarse esta clase para ayudar a mejorar su interfaz de usuario (TU). La clase *ErrorProvider* señala al usuario que la entrada contenida en el control se ha perdido o no es válida. Cuando un usuario pase el puntero del ratón sobre el icono de aviso, se mostrará en pantalla el mensaje de error asignado a *ErrorProvider*.

Otras herramientas para el control de la validación de entradas son:

HERRAMIENTA	DESCRIPCIÓN
Regex	Clase en *System.Text.RegularExpressions* que encapsula la máquina de expresiones regulares del *.NET Framework*
Controles de Validación	Juego de seis controles que validan la entrada de datos en el cliente y en el servidor: *RequiredFieldValidator*, *RegularExpression-Validator*, *RangeValidator*, etc.

HttpUtility.HtmlEncode	Codifica la entrada a HTML, convirtiendo caracteres potencialmente dañinos en secuencias de escape como <
Request.MapPath	Resuelve direccionamiento y opcionalmente valida que las rutas no violen los límites de aplicación
Validación de petición ASP.NET 1.1	Capacidad de ASP.NET 1.1 que automáticamnente rechaza peticiones que contienen ciertos caracteres y secuencias (p.ej., "<script")

Tabla 7.4 Herramientas

HERRAMIENTAS GENERALES DE VALIDACIÓN DE LENGUAJE

Visual Basic .NET dispone de un par de herramientas que no se limitan a validar la entrada realizada por el usuario; las expresiones regulares y del método *Parse* utilizado para convertir un tipo de dato en otro.

Expresiones regulares

Cada vez que se presenta un problema que se requiere validar, buscar o sustituir texto, la primera inclinación es desarrollar una expresión regular que permita realizar la tarea. Si se desconoce lo que son las expresiones regulares puede que parezca un poco enrevesada la sintaxis, además de ilegible y bastante intimidatoria. Invertir un poco de tiempo en aprender los operadores básicos puede ayudar a escribir un código compacto que valide la adecuación de los datos de cadena. Una especialidad de las expresiones regulares es su capacidad para validar (sin tener que revisar línea a línea su código) que un número de la seguridad social, un número de teléfono o el nombre de una persona, tienen el formato correcto. La Tabla 7.5 enumera las expresiones regulares que puede encontrar de utilidad en el código.

EJEMPLOS DE EXPRESIONES REGULARES.		
Expresión	**Formato**	**Expresión regular**
Nombre completo de una persona	Caracteres alfabéticos con un uso limitado de espacios, comas, puntos, apóstrofes y guiones	^[a -zA-Z]+ ([a -zA-Z])? (\')?([azA- Z]+)?(\-)?[a-zA-Z]+(\.)?(\ )?([a-zA-Z]+)?(\).?$
Número de Cuenta UNAM	########-#	\d{8}-\d{1}
Número telefónico.	(##) ####-####	((\(\d{2}\) ?)|(\d{2}-))?\d{4}- \d{4}

Tabla 7.5 Herramientas

Método Parse

El método *Parse* es un método de propósito general disponible para la mayoría de los tipos de datos, que permite convertir datos de un tipo a otro. En el proceso de conversión, el método *Parse* valida que los datos de origen se puedan convertir en el tipo de datos de destino. Si, por ejemplo, la aplicación permite que el usuario introduzca (en un control de entrada de forma libre,

por ejemplo, un cuadro de texto) datos formateados, tales como una fecha o una cantidad monetaria, incluyendo símbolos como, por ejemplo, el signo del dólar ($), comas (,), o puntos (.), es necesario contar con un método adecuado para convertir la cadena con formato en un número.

Como ejemplo, el código que se muestra a continuación convierte la cadena $100 en el valor entero 100, y, en el proceso, validar que el formato de la cadena contiene sólo un signo dólar ($) y una cantidad entera, y que no utiliza un separador decimal o de unidades de millar.

```
Dim intvalue As Integer = Integer.Parse("$100", NumberStyles.AllowCurrencySymbol)
```

Para convertir la cadena con formato en un tipo de dato numérico, es necesario seleccionar el tipo de dato que mejor se adapte al tipo de valor que desee obtener. Por ejemplo, el tipo de dato más adecuado para representar cantidades monetarias es el tipo de dato Decimal porque ha sido diseñado específicamente para dicho propósito. La Tabla 7.6 lista los métodos *Parse* y los tipos de datos asociados que puede utilizar para verificar entradas con formato numérico y de fecha/hora.

MÉTODOS *PARSE* PARA CADENAS CON FORMATO NUMÉRICO Y DE FECHA/HORA.	
Métodos *Parse*	**Descripción**
Byte.Parse, Decimal.Parse, Double.Parse, Integer.Parse, Long.Parse, Short.Parse, Single.Parse	Sólo permite el empleo de los dígitos del 0 al 9. Utilice el parámetro *NumberStyles* para especificar formatos numéricos de tipo general (por ejemplo, monetario) o indicadores que señalen cuáles son los símbolos que están permitidos (por ejemplo, el signo del dólar, separador de la unidad de millar y signos más o menos).
DateTime.Parse, DateTime.ParseExact	Valida una cadena de fecha/hora contra un conjunto estándar o exacto de formatos disponibles (dependiendo del método que se utilice) para una determinada región.

Tabla 7.6 Método Parse

En las entradas en **aplicaciones Web,** al igual que sucede con las aplicaciones basadas en *Windows Forms*, es necesario validar las entradas realizadas por los usuarios. A diferencia de lo que sucede con las aplicaciones basadas en *Windows Forms*, las aplicaciones Web pueden aceptar entradas provenientes de diversas fuentes, por lo que se debe utilizar otras herramientas para validar las entradas. Las entradas que puede recibir una aplicación Web son diversas y todas ellas son accesibles a través del objeto *Request*. Los tipos de entrada que acepta el objeto *Request* son:

- La colección *QueryString*, que contiene todos los parámetros de las cadenas de consulta.
- La colección *Form*, que contiene valores para todos los controles de entrada contenidos en el formulario Web, tales como cuadros de texto, casillas de verificación o listas desplegables. Entre estos valores se incluyen la propiedad *Text* de un cuadro de texto o la propiedad *Value* de una casilla de verificación. Si se utilizan controles *Web Form*, puede obtenerse el valor directamente del control, en lugar de utilizar la colección *Form*.

- La colección *Cookies*, que contiene todas las *cookies* enviadas desde el explorador *Web* cliente.
- La colección *ServerVariables*, que contiene todas las variables definidas por el servidor Web.
- *Params*, que permite acceder a todas las entradas del objeto *Request* que hayan sido denominadas previamente utilizando una colección adecuada.

Es necesario verificar todos los parámetros de entrada del objeto *Request* para garantizar que la entrada tiene una longitud razonable y que contiene la entrada esperada.

Los parámetros cuyo contenido se muestran al usuario, bien como parte de un mensaje de error o como una página *Web* normal, se pueden proteger de los ataques de programación de sitio cruzado sin más que pasar la entrada a través del método *Server.HtmlEncode*. Por ejemplo, debería comprobar todos los lugares de su código en los que se utilice cualquiera de las entradas del objeto *Request* que enumeramos anteriormente, por ejemplo *Request.QueryString*, tal como se muestra en el siguiente ejemplo:

```
lblBienvenido.Text = Bienvenido & Request.QueryString("NombreUsr")
```

Como mínimo, debe utilizar *Server.HtmlEncode* para garantizar que ningún contenido que vaya a mostrar el explorador cliente de Internet se va a procesar como código HTML. Por ello, el código mostrado anteriormente debería quedar como:

```
lblBienvenido.Text = Server.HtmlEncode(Request.QueryString("NombreUsr"))
```

7.1.3. TÉCNICAS Y HERRAMIENTAS DE PRUEBAS

Probar la aplicación antes de implementarla o distribuirla es la forma de ir por delante de cualquier atacante que pueda utilizar posteriormente la aplicación. Puede optarse por varios enfoques a la hora de probar la aplicación. Las ventajas y desventajas de cada uno de los enfoques posibles se muestran en la Tabla 7.7.

Los enfoques mostrados en la Tabla 7.7 son válidos para las pruebas de carácter general, así como para aquellas que intentan probar que la aplicación es segura. La diferencia fundamental entre las pruebas de carácter general y las pruebas de seguridad es el tipo de pruebas que debe ejecutarse y las herramientas que se utilizan para validar la aplicación. En el caso de probar la seguridad, el objetivo es simular a un atacante; para ello, deben desarrollarse pruebas que se lleven a cabo en las mismas acciones que se realizaría un atacante y utilizando las mismas herramientas. Con independencia de que las pruebas tengan carácter general o de seguridad, el objetivo es el mismo: intentar "romper" la aplicación.

TÉCNICAS DE PRUEBAS	VENTAJAS	DESVENTAJAS
Pruebas de autocomprobación	Captura rápidamente las regresiones. Las pruebas se integran en el código.	Requiere el desarrollo y la ejecución de una versión de depuración independiente de la versión final.
Pruebas ad hoc o manuales	Coste inicial bajo. Adecuada para escenarios que sean difíciles de automatizar. Técnica adecuada para localizar nuevos errores. Eficaz para validar la interfaz de usuario. Eficaz para la validación end-to-end de una aplicación.	Alto coste a largo piazo para el caso de que ejecute la misma prueba para cada una de las versiones de su aplicación.
Pruebas unitarias automatizadas	Evita las regresiones del código. Garantiza un cierto nivel de calidad basado en la calidad de las pruebas. Relativamente barata al repetir automáticamente todas las pruebas en cada una de las ediciones del producto.	. Costo inicial elevado de creación del sistema de pruebas. Costo de mantenimiento elevado si las características del producto cambian de manera significativa. .
Pruebas de fortaleza	Para descubrir vulnerabilidades en aplicaciones que se ejecuten en entomos de elevada carga, tales como condiciones de memoria escasa y ocupación elevada del CPU. Adecuadas para comprobar las vulnerabilidades relacionadas con los ataques de denegación de servicio.	En general se tienen que ejecutar durante un largo periodo de tiempo (incluso días) antes de que aparezcan los problemas. Cuando aparecen los problemas resulta difícil recrear las condiciones exactas en las que se ha producido el fallo sin volver a ejecutar la prueba, lo que requiere grandes periodos de tiempo.

Tabla 7.7 Pruebas

Autocomprobación

Visual Basic .NET dispone de un rico conjunto de funciones de depuración, tales como *Debug.Assert* y *Debug.Fail*. Estas instrucciones sólo se activarán en las versiones de depuración de la aplicación.

Pueden utilizarse estas instrucciones para validar ciertas hipótesis realizadas en el código sin que ello signifique una disminución del rendimiento en la versión operativa de la aplicación. Por ejemplo, si se desea asegurar que en un determinado lugar del código la cadena que representa al

nombre del usuario no se encuentra vacía y que tiene como máximo 20 caracteres, puede agregarse la siguiente instrucción:

```
Debug.Assert(NombreUsuario <> AndAlso NombreUsuario.Length <= 20,_
"Nombre de usuario no válido")
```

Código Fuente 7.I.

Estas instrucciones no deben incluirse en la versión comercial de la aplicación. Si por alguna razón la hipótesis no se cumple y el nombre de usuario no es válido, la versión comercial de la aplicación podría llegar a utilizar un nombre no válido para el usuario o, incluso, podría sufrir un error. En cualquier caso, tal vez no sea capaz de decir, sin más que ejecutar la aplicación comercial, dónde se encuentra exactamente el problema. Sin embargo, si ejecuta el mismo escenario utilizando la versión de depuración, se mostrará en la pantalla un aviso (un cuadro de mensaje mostrando la advertencia "Nombre de usuario no válido"). Si prueba el código con llamadas del tipo *Debug.Assert* para comprobar que sus variables *NombreUsuario* contienen un valor válido, puede conocerse con facilidad el punto exacto del código en el que el valor de *NombreUsuario* deja de serlo.

También puede incluirse código de prueba dentro de las subrutinas sin más que agregar el atributo Conditional(" DEBUG"), tal y como se muestra a continuación:

```
'Agregar Imports System.Diagnostics en la parte superior del módulo de código
    <Conditional("DEBUD")> Sub ProbarControlesTexto()
        'Agregar código que intente pasar una cadena no válida
        'a todos los controles de entrada de texto contenidos en la aplicación
        'para validar la forma en que estos controles manipulan el error.

End Sub
```

Código Fuente 7.J.

Las llamada a *ProbarControlesTexto* se incluye únicamente si se define el símbolo DEBUG, algo que ocurrirá cuando genere la aplicación en *Microsoft Visual Studio .NET* habiendo seleccionado la configuración de depuración. Es posible el uso de versiones de depuración para llevar a cabo autovalidaciones de la aplicación, algo que disminuirá el rendimiento de la aplicación. La subrutina P*robarControlesTexto* no se incluirá (no habrá ninguna llamada a esta subrutina) en las versiones comerciales dado que, en este caso, lo que desea es obtener un elevado rendimiento de ejecución.

Pruebas ad hoc o manuales

Las pruebas ad hoc, también conocidas como pruebas manuales, son la forma más directa de verificación que podrá realizar. La idea es ejecutar la aplicación e intentar romperla. El objetivo de los escenarios de prueba ad hoc es, por ejemplo, introducir datos que no sean válidos en todos los campos de entrada de datos y ver cómo responde la aplicación. Esta forma de proceder le proporciona el mejor y más realista punto de vista de la aplicación y del comportamiento. Las pruebas ad hoc son idóneas para probar las nuevas funciones con el fin de encontrar nuevos errores tan rápido como sea posible. También ayudan a localizar los errores que los clientes

encontrarán con toda probabilidad cuando utilicen la aplicación. El mayor inconveniente de este tipo de verificación es que le llevará bastante tiempo analizar todos los posibles puntos de fallo de la aplicación. Para verificar que nada ha cambiado entre dos versiones consecutivas, es necesario repetir los pasos cada vez que aparezca una nueva versión.

Pruebas unitarias automatizadas

Las pruebas unitarias automatizadas implican el desarrollo de código o de archivos de comandos para analizar una determinada característica de la aplicación y, más importante aún, para validar que las funciones trabajan en la forma esperada. Las pruebas manuales o ad hoc son la mejor forma de descubrir los nuevos errores. El desarrollo de pruebas automatizadas implica un aumento del costo inicial del proyecto y, además, tiene que analizarse y verificarse los resultados proporcionados por las funciones de soporte. Sin embargo, una vez que el sistema automatizado de pruebas se encuentre operativo, agregar nuevas pruebas sólo supondrá un pequeño incremento de los costos.

La principal ventaja de contar con un sistema automatizado de pruebas es que puede utilizase para probar cada nueva versión de la aplicación, estableciendo una línea base de medida de la calidad de la aplicación, evitando cualquier disminución de la calidad. La situación ideal es analizar todos los errores conocidos que se hayan resuelto antes en las aplicaciones, escribir los casos de prueba para cada error y ejecutar estos casos contra cualquiera de las versiones que se hayan podido desarrollar. Si los casos de prueba han sido escritos correctamente y las versiones superan el cien por cien de los mismos, se tiene la certeza que la calidad de la aplicación no ha disminuido entre dos versiones consecutivas.

Si la función se encuentra preparada para iniciar una *InvalidDurationException* (una excepción personalizada definida a nivel aplicación, que deriva de *System.Exception*, que representa una excepción por duración no valida) cuando la duración sea igual o menor que cero, puede desarrollarse una prueba que llame a la función y pase el valor 0 para el parámetro duración, tal y como se muestra a continuación:

```
Try
    CalcularPagos(100, 0)
    RegistrarPruehasFalladas()
Catch exExpected As InvalidDurationException
    RegistrarPruebasSuperadas()
Catch exExpected As Exception
    RegistrarPruebasFalladas()
 End Try
```

Código Fuente 7.K.

Implementar las funciones *RegistrarPruebasSuperadas* y *RegistrarPruebasFalladas*, es posible, de un par de formas distintas:

- Las funciones podrían escribir en una base de datos los resultados si se desea registrar los resultados obtenidos en las pruebas. Pueden también definirse las funciones *RegistrarPruebasSuperadas* y *RegistrarPruebasFalladas* para que escriban el nombre de la prueba ejecutada, la versión de la aplicación que se está verificando y el resultado de la prueba (Pasada o Fallada) en una tabla de una base de datos. Una vez que haya ejecutado todas las

pruebas automatizadas contra una versión de su aplicación, puede analizarse el contenido de esta base de datos y ver las pruebas que no hayan sido superadas.

- Las funciones pueden emitir un resultado del tipo Pasada o Fallada y escribirlo en un archivo de texto. Por ejemplo: las funciones escriban PASADA (en el caso de *RegistrarPruebasSuperadas*) o FALLADA (en el caso de *RegistrarPruebasFalladas*) en un archivo de texto de RESULTADOS.TXT. El ejecutar las pruebas, inspeccionar el contenido del archivo RESULTADOS.TXT para verificar que contiene los resultados PASADA esperados y almacenar el archivo como RESULTADOS.LOG, que se convertirá en su archivo de referencia. El ejecutar las mismas pruebas sobre cada una de las nuevas versiones de la aplicación y comparar los resultados utilizando una herramienta tal como ***WinDiff.Exe*** (incluida con Visual Basic .NET) para comparar RESULTADOS.TXT contra RESULTADOS.LOG. Cualquier diferencia existente entre los dos archivos se deberá considerar como un fallo y se asociará un error con la prueba que haya fallado.

Para desarrollar y ejecutar pruebas automatizadas podrá crearse un propio sistema de pruebas utilizando Visual Basic .NET, como lo he mostrado aquí, o utilizar una herramienta de pruebas de carácter comercial. Por ejemplo, utilizar la herramienta gratuita de pruebas denominada ***NUnit***, que ha sido diseñada específicamente para crear y ejecutar pruebas para aplicaciones desarrolladas en cualquier lenguaje .NET.

Pruebas de fortaleza

Las pruebas de fortaleza ayudan a identificar problemas o vulnerabilidades de la aplicación en condiciones de alta carga. Por ejemplo, pueden utilizarse herramientas de fortaleza para ver la forma en que su aplicación responde ante las siguientes situaciones:

- Cuando la memoria disponible es escasa o no existe.
- Cuando el espacio libre en disco es escaso o inexistente.
- Cuando varios subprocesos están ejecutando simultáneamente diferentes partes de su código (de utilidad cuando se desea probar componentes de subproceso múltiple).
- Cuando se realizan simultáneamente varias peticiones Web desde un único cliente Web o desde varios clientes Web (de utilidad para comprobar aplicaciones Web ASP.NET y servicios Web).

Las pruebas de análisis de la fortaleza conllevan dos beneficios principales:

- Identificar los problemas de seguridad en la aplicación. Por ejemplo, si bajo una condición de poco espacio libre en el disco la aplicación Web devuelve un error que revele la ruta de acceso y el nombre de un archivo que la aplicación Web no ha logrado guardar, el atacante podría utilizar esta información para conocer la forma en que se encuentran organizados los archivos dentro del servidor y lanzar otras formas de ataque.
- Detectar los cuellos de botella relacionados con el rendimiento y la escalabilidad. Al identificar este tipo de problemas se consigue que las aplicaciones sean más resistentes ante los ataques de Denegación de Servicio (DoS). Aunque con frecuencia necesite utilizar una herramienta de

este tipo para comprender mejor y resolver un problema de rendimiento o de escalabilidad, las herramientas de análisis de fortaleza sólo alertan del problema y no proporcionan la suficiente información sobre cómo resolverlo. Las herramientas de *profiling*, tales como Prueba del centro de aplicaciones de Microsoft (***Microsoft Application Center Test***), disponen de funciones que le permiten analizar el comportamiento de la aplicación bajo condiciones de alta carga.

Herramientas de pruebas

Las herramientas de prueba permiten analizar o probar la aplicación de diferentes formas. Los *hackers* utilizan diversas herramientas para crear un esquema de las aplicaciones que van a atacar. Con estas herramientas pueden conocer los archivos que forman la aplicación, el contenido de dichos archivos y los recursos externos utilizados (tales como las ubicaciones de red, bases de datos o direcciones Web). Además, el *hacker* utiliza herramientas para romper las defensas de la aplicación (por ejemplo, manipuladores de las entradas, reveladores de contraseñas y herramientas que generen sobrecarga en el sistema). Pueden utilizarse estas mismas herramientas u otras similares para probar la aplicación y comprobar la seguridad (o, al menos, ejecutar la herramienta, como lo haría un *hacker*, sobre la aplicación para ver lo que dicha herramienta descubre). La Tabla 7.8 lista algunas de las herramientas disponibles.

CATEGORÍA	EJEMPLO	DESCRIPCIÓN
Ingeniería inversa para descubrir la lógica de la aplicación	ILDasm, Anakrino, Link, Dump-Bin FileMon, Teleport Pro	Estas herramientas se utilizan para convertir binarios en código fuente o intermedio con el fin de descubrir la lógica de la aplicación, sus dependencias y los secretos almacenados en ella.
Manipulación de la página Web	NetMon, Netcat, Cookie Pal, Achilles	Estas herramientas se utilizan para analizar y manipular datos en bruto, tales como código HTML, cabeceras HTI'P y cookies que se envíen entre un cliente y un servidor.
Redirección de red	Netcat	Estas herramientas le permitirán a un hacker secuestrar un servidor de red, superar los cortafuegos y proporcionar el control total de un equipo de la red al hacker.
Revelación de contraseñas	JohnTheRipper, PWDump, LSADump2, LOphtCrack (en diferentes modelos, tal como LC4 de @stake)	Estas herramientas resultan de utilidad para revelar las contraseñas de usuario de sistemas operativos, bases de datos o aplicaciones Web.
Pruebas unitarias automatizadas	NUnit (herramientas de pruebas unitarias para .NET)	Estas herramientas proporcionan un entorno en el que podrá crear y ejecutar pruebas automatizadas.

Herramientas de prueba de perfiles y sobrecarga	Prueba del centro de aplicaciones de Microsoft (ACT), Microsoft SQL Server Profiler y ANTS.	Con estas herramientas podrá forzar fallos por sobrecarga en su aplicación con el propósito de identificar determinadas vulnerabilidades ante ciertos problemas de seguridad, incluyendo ataques de Denegación de servicio (DoS) y de descubrimiento de información.

Tabla 7.8 Herramientas

7.1.4. TÉCNICAS DE IMPLEMENTACIÓN

Es posible implementar (distribuir) una aplicación o componente Visual Basic .NET de diversas formas como, por ejemplo: utilizando *XCopy*, de forma automática, mediante Windows *Installer* o una instalación basada en archivos contenedores (*Cabinet*).

Las técnicas de instalación basadas en el empleo de *Windows Installer* y de los archivos contenedores son tecnologías tradicionales de implementación que se utilizan para distribuir aplicaciones y componentes basados en *Windows* (por ejemplo, controles ActiveX) y en *Microsoft Windows .NET*. Las técnicas basadas en *XCopy* y en modelos automáticos (no-touch) son técnicas de implementación que se han desarrollado expresamente para Visual Basic .NET.

Implementación basada en XCopy

La implementación basada en *XCopy* debe su nombre a la herramienta ejecutable desde la línea de comandos denominada *XCopy.exe*. La implementación basada en *XCopy* es una forma de copiar recursivamente un directorio que contiene una aplicación (y todos los subdirectorios contenidos dentro de dicho directorio) en la ubicación final donde se ejecutará la aplicación. .NET hace posible la implementación basada en *XCopy* porque los componentes y aplicaciones basados en .NET no requieren normalmente ningún tipo de registro especial (a diferencia de lo que ocurre con los componentes *ActiveX*) para poder ejecutarse. Si la aplicación basada en Visual Basic .NET está compuesta enteramente por componentes .NET es posible utilizar la implementación basada en *XCopy*. Cuando se copia una aplicación basada en Visual Basic .NET desde un equipo a otro utilizando esta técnica, debe conservarse la siguiente información:

- Los atributos de los archivos tales como sólo lectura, oculto, sistema, etc. Si utiliza XCopy.exe, debe emplear la opción /K para conservar los atributos de carpeta y de archivo, la opción /R para permitir la sustitución de los archivos de sólo lectura (en el equipo destino) y la opción /H para copiar archivos ocultos y del sistema.
- Las listas de control de acceso (ACL) de *Windows* relacionadas con la seguridad, para todos los archivos y carpetas que se vayan a copiar. Si utiliza *XCopy.exe*, emplee la opción /O para conservar los permisos.

Implementación automática

Esta técnica (también conocida como *no-touch*) implica permitir la posibilidad de ejecutar la aplicación o componente *Windows Forms* basado en *Visual Basic .NET* directamente desde una página *Web*; esto significa que la aplicación y sus componentes asociados se copiarán

directamente desde Internet (o desde la intranet, dependiendo de dónde se encuentre la página Web) en una caché de descarga especial de Internet contenida en el equipo y que se ejecutarán desde este lugar. Entre los beneficios de utilizar este tipo de implementación destacan:

- Disposición de un modelo centralizado de distribución para aplicaciones y componentes *Windows Forms*, en el que todos los clientes se actualizarán automáticamente con la versión más reciente de la aplicación o del componente instalado en un servidor Web.
- La aplicación o el componente tiene asignados permisos de seguridad de acceso mediante código *.NET*, basados en la zona de seguridad, como se verá más adelante en este tema.

Implementación mediante Windows Installer

Se puede utilizar una amplia variedad de herramientas, incluyendo el Asistente para implementación de *Microsoft Visual Studio .NET*, para crear un paquete de implementación de *Windows Installer* (.MSI).

Un paquete de *Windows Installer* es un programa de instalación completo, con una práctica interfaz de usuario (IU), que da al usuario a elegir entre distintas opciones de instalación tales como las funciones de la aplicación que desee instalar y la ubicación de los directorios en los que se almacenarán los distintos archivos de la aplicación. Debe utilizar un paquete de *Windows Installer* siempre que la aplicación deba cumplir alguno de los siguientes requisitos:

- El proceso de instalación va a contar con opciones entre las que podrá elegir el usuario.
- Cuando el usuario deba determinar la ubicación del directorio que contendrá la aplicación.
- Cuando se deba registrar algún componente .NET o *ActiveX*.
- Cuando se deban instalar componentes .DLL basados en .NET (ensamblados) en la caché del ensamblado global (GAC)
- Cuando desee instalar accesos directos en el escritorio o elementos de programa en el menú Inicio.
- Para reiniciar el sistema como parte del proceso de instalación con el fin de sustituir un archivo que el sistema operativo esté utilizando.
- Cuando desee desactivar un servicio del sistema operativo Windows requerido para instalar un componente.

Otro posible empleo del paquete de *Windows Installer* es empaquetar permisos de seguridad de código de acceso .NET para distribuirlos con la aplicación *Windows Forms*. Es posible usar un archivo .MSI distinto para implementar la aplicación o componente *Windows Forms* que requiera la actualización de la directiva de seguridad; también podrá utilizar la técnica de implementación automática que hemos descrito anteriormente.

Implementación de archivos Cabinet

Los archivos contenedores o *Cabinet* (.CAB) se utilizan normalmente como un sistema para distribuir los componentes *ActiveX* a través de Internet. Los archivos .CAB utilizan la misma tecnología subyacente que los archivos *Windows Installer* pero, en general, los archivos .CAB no dispondrán de una interfaz de usuario.

Si se desea incluir un componente *ActiveX* en una página *Web*, debe pensarse en utilizar un archivo .CAB para implementar el componente. En el caso de componentes .NET que se deseen incluir en una página Web, debe utilizarse el sistema de implementación automática, analizada anteriormente.

La Tabla 7.9 muestra un resumen de las técnicas de implementación disponibles, junto con algunas directrices sobre cuándo utilizar cada una de ellas.

TÉCNICA DE IMPLEMENTACIÓN	CUÁNDO UTILIZARLA	CARACTERÍSTICAS	¿SE EJECUTA EN EL SANDBOX?
XCopy	Cuando la aplicación sea un servicio Web oASP.NET implementado en un servidor Web.	No se requiere el empleo de un paquete de instalación sofisticado para distribuir una aplicación Web a un servidor Web. No se necesita detener el servidor Internet Information Services (IIS) de destino para copiar los nuevos componentes .NET en el servidor Web.	No. La aplicación .NET se copiará en el equipo de destino y se ejecutará en la zona de seguridad de Mi PC. La aplicación se ejecuta con plena confianza.
Automática (no-touch)	La aplicación o el componente es una aplicación o componente Windows Forms (por ejemplo, un control de usuario) que se implementará por Internet. La aplicación o el componente Windows Forms se ha diseñado para descargar de forma automática otros componentes dependientes (llamando a *Assembly.LoadFrom* y pasando una dirección URL) desde Internet cuando así se requiera.	No requiere ningún tipo de empaquetamiento. Se hará referencia directa a los componentes .EXE y .DLL de la aplicación mediante un vínculo con una página Web. Las aplicaciones *Windows Forms* que descarguen dinámicamente los componentes *Windows Forms* necesitarán una conexión con Internet para estar siempre presentes cuando se ejecute la aplicación. Todos los equipos cliente recibirán automáticamente la última versión de la aplicación *Windows Forms* cuando ésta se	Sí. A la aplicación .NET se le asignarán los permisos de seguridad de acceso mediante código basándose en la zona desde la que ha sido instalada. Si la aplicación se instala desde Internet, la aplicación se ejecutará en la zona Internet con una confianza baja (o sin confianza, dependiendo de la versión que se encuentre instalada de *.NET Framework*).

		encuentre disponible. Podrá distribuir sus aplicaciones basadas en *Windows Forms* desde un servidor Web central.	
Windows Installer (.MSI)	La aplicación contiene componentes .NET o *ActiveX* que requieran un registro especial. Se necesita añadir a la aplicación los permisos de la directiva de seguridad del código de acceso .NET. Hace falta reiniciar el sistema como parte de la instalación. Durante la instalación, hace falta sustituir ciertos archivos bloqueados (lo que exigirá la detención y el reinicio de los servicios del sistema operativo que se encuentren relacionados). Hace falta añadir ciertas opciones de configuración (por ejemplo, seguridad NTFS). Hace falta crear accesos directos al escritorio o elementos de menús. La aplicación contiene componentes o características opcionales que podrá elegir el usuario.	Proporciona un sistema para distribuir actualizaciones de la directiva de seguridad del código de acceso .NET para las aplicaciones que se implementen utilizando: técnicas de instalación automáticas, instalación basada en *Windows Installer* o en archivos *Cabinet*. Es la forma de implementación utilizada con mayor frecuencia por los sistemas tradicionales de distribución, tales como aquellos basados en software empaquetado. Los archivos .MSI se pueden distribuir a través de Internet, pero no se podrá proporcionar una protección de la seguridad del código de acceso una vez que hayan sido distribuidos por Internet. Es el tipo de implementación utilizada para distribuir el propio Visual Studio .NET.	No. La aplicación .NET se instalará en el equipo de destino y se ejecutará en la zona de seguridad Mi PC. El sistema de seguridad d acceso mediante código de .NET asignará plena confianza a la aplicación. Esto será así incluso aunque el archivo .MSI provenga de una página Web de Internet. El sistema de seguridad de acceso mediante código de .NET no es consciente de que dicha aplicación (cuando se ejecute en el equipo local) ha sido originada en un paquete de instalación .MSI proveniente de Internet.
Archivos contenedor o *Cabinet* (.CAB)	Se va a distribuir un componente .NET o *ActiveX* a través de Internet para su empleo en una página Web.	Si en el .CAB se incluye un componente .NET, la implementación se comportará de forma bastante similar a la técnica de implementación automática.	

Tabla 7.9 Herramientas

Se dice que las técnicas de implementación que ofrecen una protección basada en la seguridad de acceso mediante código se ejecutan en el *sandbox*. El término *sandbox* o caja de arena, cuando se aplica a *software*, hace referencia a un entorno aislado en el que una aplicación se puede ejecutar y desde la que no puede dañar a otras aplicaciones, al sistema operativo o a los datos (u otros recursos) almacenados en el equipo. El *sandbox* de seguridad de acceso mediante código de .NET es el entorno restrictivo en el que se ejecutan las aplicaciones .NET identificadas como ejecutadas en cualquier otra zona que no sea Mi PC. Para comprender ampliamente a que me refiero cuando hablo de zonas el tema siguiente permite tener un panorama más claro.

7.2. SEGURIDAD IMPLEMENTADA POR EL .NET FRAMEWORK

7.2.1. PERMISOS

En el capítulo 3 de este trabajo, se explicó a detalle las características de los permisos y grupos con los que trabaja .NET, en este tema se aplicaran muchos de esos conceptos para comprender los tipos permisos en zonas de seguridad con base a la tabla 7.10 y con las indicaciones: "✓" para definir que son todos los permisos, "×" sin permisos y "✶" permisos parciales..

NIVEL DE PERMISOS	LO QUE SE PERMITE A LAS APLICACIONES EN LA ZONA DE …		
	MI PC	INTRANET LOCAL	INTERNET Y SITIOS DE CONFIANZA
DnsPermission	✓ Realizar operaciones de Sistema de Nombres de Dominio (DNS) tal como convertir un nombre URL en una dirección IP.	✓ Permite a la aplicación de Visual Basic .NET obtener nombres de host (tal como www.mycompany.com) para una dirección IP determinada u obtener una dirección IP para un determinado nombre de host.	× No se concede ningún permiso DNS.
EventLogPermission	✓ Leer o escribir de/en el registro de sucesos.	✶ Permite a la aplicación de Visual Basic .NET leer y escribir desde/en el registro de sucesos. La aplicación no puede eliminar entradas.	× No se asigna ningún permiso sobre el registro de sucesos.
EnvironmentPermission	✓ Leer o escribir variables de entorno.	✶ El código de Visual Basic .NET puede leer la variable de entorno USERNAME. Cualquier intento de leer o escribir en cualquier otra variable de entorno provocará una	× No se asigna ningún permiso de variable de entorno.

		excepción de seguridad.	
FileDialogPermission	✓ Mostrar los cuadros de diálogo Abrir archivo y Guardar.	✓ El código de Visual Basic .NET puede mostrar todos los cuadros de diálogo de archivo disponibles, incluyendo los cuadros de diálogo Abrir archivo y Guardar archivo.	✶ El código de Visual Basic .NET puede mostrar el cuadro de diálogo Abrir archivo, pero no el cuadro de diálogo Guardar archivo.
FileIOPermission	✓ Leer, escribir o anexar archivos.	× No se concede ningún permiso	× No se concede ningún permiso
IsolatedStorageFilePermission	✓ Leer o escribir datos en un sitio especial y reservado en el disco duro de su PC.	✶ Las aplicaciones de Visual Basic .NET podrán leer y escribir archivos desde/en un área especial del disco. A cada usuario se le asigna un área de almacenamiento independiente para cada componente que forme la aplicación. El espacio en disco asignado a la aplicación no se encuentra limitado.	✶ En este caso se tienen las mismas restricciones que para el permiso de Intranet local, pero el almacenamiento se encuentra limitado a 10.240 bytes (10 kilobytes o 10 K) por usuario y por aplicación donde todos los componentes (.DLL) que conforman la aplicación comparten el mismo espacio limitado de almacenamiento aislado (10 K).
PrintingPermission	✓ Conectarse con impresoras locales o de red.	✶ Permite a la aplicación de Visual Basic .NET imprimir en la impresora predeterminada pero no en ninguna otra impresora conectada a su equipo. También permite la impresión mediante un cuadro de diálogo de impresión restringido.	✶ Permite que Visual Basic .NET pueda imprimir a través de un cuadro de diálogo de impresión, que es más restrictivo que el cuadro de diálogo disponible en la zona de Intranet local.
ReflectionPermission	✓ Consultar las clases, módulos, propiedades, métodos y sucesos que forman la aplicación.	✓La aplicación puede generar sobre la marcha aplicaciones .NET y leer tipos públicos desde una aplicación .NET.	× No se conceden permisos de reflexión.
RegistryPermission	Leer y escribir en el registro del sistema.	× No se conceden permisos	× No se conceden permisos
SecurityPermission	✓ Realizar operaciones relacionadas con la	✶ Permite la ejecución del código de Visual Basic	✶El código de Visual Basic .NET puede realizar

	seguridad tales como desactivar las verificaciones de permisos. Aquí se incluyen los denominados permisos de código no administrados tales como la capacidad para llamar a funciones de Windows API o para utilizar controles o componentes AciveX	.NET y garantiza, de forma temporal, una serie de permisos adicionales a los llamadores, permisos que no poseerá la aplicación llamada, aunque sí que tiene concedidos la propia aplicación de Visual Basic .NET. Observe que no se permite la posibilidad de llamar a código no administrado tal como funciones APIo componentes ActiveX.	llamadas a funciones API. No se permite el empleo de componentes ActiveX.
UlPermission	× Mostrar una UI y los tipos de UI que se pueden mostrar, tales como ventanas de nivel superior. También controla el acceso de las aplicaciones al portapapeles del sistema.	✶ Se garantizan todos los permisos UIPermissions (por ejemplo, la capacidad de mostrar cualquier tipo de ventana de nivel superior y acceso total al Portapapeles.	✶ El código de VB:NET sólo puede mostrar ventanas seguras de nivel superior, que son las que satisfacen ciertas restricciones de tal forma que no puede iniciar cuadros de dialogo del sistema, tales como los de inicio de sesión del usuario. Este permiso incluye acceso al portapapeles.
	PERMISOS DE PLENA CONFIANZA ASIGNADOS A LA ZONA DE MI PC		
DirectoryServicesPermission	Examinar, leer y escribir las entradas de Active Directory.		
MessageQueuePermission	Localizar las colas de mensajes disponibles, leer los mensajes contenidos en la cola o enviar y recibir mensajes.		
OleDbPennission	Acceder a un proveedor de OleDb o definir una contraseña en blanco en la cadena de conexión.		
PerformanceCounterPermission	Localizar, modificar o crear categorías de contadores de rendimiento.		
ServiceControllerPermission	Localizar, activar o desactivar servicios de Windows tales como el servicio SQL.		
SocketPermission	Leer desde o escribir en una determinada red basándose en el nombre del Host, puerto y transporte.		
SqlClientPermission	Utilizar una contraseña en blanco como parte de la cadena de conexión.		
WebPermission	Aceptar datos o transmitir datos desde/hacia un determinado URL.		

Tabla 7.10 Herramientas

7.2.2. CLASES DE SEGURIDAD .NET

El *.NET Framework* proporciona un conjunto rico de clases que se puede usar para proteger y firmar datos sensibles usando criptografía. Estas clases están en el espacio de nombre (*namespace*) *System.Security.Cryptography*. Las funciones de las clases incluyen las siguientes cuatro áreas:

- Encriptación y desencriptación

  Permite la encriptación y desencriptación de datos al utilizar un algoritmo simétrico o asimétrico específico. Los algoritmos simétricos soportados son DES, Triple DES, RC2 y Rijndael. Los algoritmos asimétricos soportados son RSA y DSA.

- Generación y verificación de firmas digitales

  Permiten la generación y verificación de firmas digitales para datos basados en un algoritmo específico. Los algoritmos soportados incluyen RSA y DSA.

- Generación de análisis

  Permite la generación de valores de análisis para datos. Los algoritmos de análisis soportados son MD5, SHA1, SHA256, SHA384 y SHA512.

- Firmas XML

  Realmente un subconjunto de las firmas digitales, las clases de Firma XML implementan las recomendaciones propuestas ante el W3C sobre Firmas XML. Estas clases residen en el espacio de nombre *System.Security.Cryptography.Xml.*

A continuación se presentan las más importantes para este trabajo

7.2.2.1. CIFRADO: ENCRIPTACIÓN Y DESENCRIPTACIÓN

Existen dos tipos principales de algoritmos criptográficos: simétrico y asimétrico.

- Simétrico: Los algoritmos criptográficos simétricos tienen una sola clave secreta que se usa tanto para la encriptación como para la desencriptación. El uso de un algoritmo simétrico requiere que sólo el emisor y el receptor conozcan la clave secreta. DES_CSP, RC2_CSP y TripleDES_CSP son implementaciones de algoritmos simétricos.
- Asimétrico: Los algoritmos criptográficos asimétricos, también conocidos como algoritmos de clave pública, requieren que cada entidad mantenga un par de claves relacionadas: una clave privada y una pública. Ambas claves son únicas para cada entidad.
  - La clave pública puede estar disponible a todos y se usa para codificar datos que se enviarán al receptor.

El receptor debe mantener en anonimato la clave privada y se usa para descodificar mensajes codificados utilizando la clave pública del receptor. RSA_CP es una implementación de un algoritmo de clave pública.

¿Qué es el cifrado? Antes de analizar la forma de utilizar el cifrado en Visual Basic .NET, debe tenerse un conocimiento general de lo que se entiende por cifrado. El cifrado está relacionado con el hecho de mantener seguros los secretos al codificar mensajes para hacerlos ilegibles. En términos de cifrado, el mensaje original recibe el nombre de Mensaje en claro o texto claro, el mensaje codificado se denomina Mensaje cifrado o texto cifrado, el proceso de convertir el Mensaje en claro en Mensaje cifrado se denomina cifrado y el proceso de recuperar el Mensaje claro a partir del Mensaje cifrado se denomina descifrado.

El cifrado no se utiliza sólo en el ciberespacio o en misteriosos trabajos gubernamentales. Puede encontrarse ejemplos sobre este tema en casi todas las actividades de nuestra vida diaria. Las computadoras nos permiten cifrar complejos mensajes en tiempo real, para que el cifrado sea eficaz, el emisor y el receptor deben ser las únicas partes que conozcan la forma en que se ha cifrado y descifrado el mensaje. *Microsoft Windows* y *.NET Framework* cuentan con algoritmos robustos para llevar a cabo el cifrado.

Suele ser un error muy frecuente pensar que los algoritmos de cifrado y las funciones ***hash*** deben ser secretos para resultar seguros. Los algoritmos de cifrado y las funciones *hash* (o de resumen) son de dominio público y el código fuente asociado se distribuye libremente por Internet. Sin embargo, siguen siendo seguros porque son irreversibles (en el caso de las funciones *hash*) o requieren que el usuario proporcione una clave secreta (en el caso de los algoritmos de cifrado). Siempre que sólo las partes autorizadas conozcan la clave secreta, el mensaje cifrado seguirá siendo seguro. El cifrado ayuda a garantizar tres cosas:

- Confidencialidad Sólo el receptor adecuado podrá descifrar el mensaje que haya enviado.
- Autenticación Los mensajes cifrados que reciba habrán sido generados por una fuente de confianza.
- Integridad Cuando envíe o reciba un mensaje no será alterado durante su trayecto.

Tres de los aspectos que ya se han mencionado y estudiado anteriormente, algunos mecanismos criptográficos son de un solo sentido; es decir, producen texto cifrado que no se puede descifrar. Un buen ejemplo de un cifrado unidireccional es el *hash*. Un ***hash*** es un número de gran tamaño, generado matemáticamente a partir de un mensaje en texto claro. Como el *hash* no contiene información sobre el mensaje original, éste no se puede deducir del *hash*. "¿Para qué se quiere utilizar el texto cifrado que no se puede descifrar?", puede uno preguntarse. Un *hash* resulta útil para verificar que alguien conoce un secreto sin tener que almacenar en realidad dicho secreto, así como para aprender a crear y utilizar un *hash* para verificar las contraseñas.

Un *hash*, como ya lo mencione, es un tipo de cifrado unidireccional. Algunas personas se refieren a las técnicas de ***hashing*** como cifrado; otras dicen que no se trata estrictamente de cifrado porque no se puede descifrar un *hash*. Los *hashes* son números de gran tamaño, generados sin más que codificar y condensar las letras de una cadena.

SHA- 1 es la abreviatura de ***Secure Hashing Algorithm*** (Algoritmo seguro de hashing). El "1' hace referencia a la revisión 1, que fue desarrollada en 1994. SHA-l acepta una cadena como entrada y devuelve un número de 160 bits (20 *bytes*). Como cualquier cadena se va a condensar en

un número de tamaño fijo, el resultado recibe el nombre de resumen *hash* o *hash digest*, donde *digest* (resumen) indica un tamaño menor. Los resúmenes *hash* son cifrados unidireccionales porque es imposible obtener la cadena original a partir del *hash*. Un resumen *hash* es como la huella digital de una persona. La huella digital identifica de forma única a una persona sin revelar ningún detalle más acerca de la misma (no se puede determinar el color de ojos, la altura o el sexo de una persona analizando su huella digital).

Los resúmenes *hash* resultan útiles para verificar que alguien sabe una contraseña, sin tener que almacenar dicha contraseña. Al almacenar contraseñas sin cifrar en una base de datos se abren dos agujeros de seguridad:

- Si un intruso consigue acceder a la base de datos, podrá utilizar dicha información para iniciar una sesión posteriormente en el sistema utilizando el nombre de usuario y la contraseña de algún otro.
- Las personas suelen utilizar las mismas contraseñas en distintos sistemas, por lo que al perder nuestras contraseñas estamos posibilitando que el intruso se conecte a otros sistemas.

Como la contraseña se utiliza exclusivamente para autenticar al usuario, no existe ningún motivo para almacenar la contraseña en la base de datos. En su lugar, se puede almacenar un resumen *hash* de la contraseña. Cuando el usuario se conecte al sistema, se creará un resumen *hash* de la contraseña que haya tecleado y se comparará con el resumen *hash* almacenado en la base de datos. Si algún intruso consigue acceder a la tabla de contraseñas, no será capaz de utilizar el resumen *hash* para iniciar la sesión en el sistema porque tendría que conocer la contraseña sin cifrar, que no se encuentra almacenada en ninguna parte.

¿Cómo funciona un resumen *hash*?

Si cada cadena proporciona un único resumen *hash*, ¿será posible descifrar el resumen *hash* y obtener la cadena original? Para responder a estas dos preguntas vamos a crear un sencillo algoritmo de *hash*. Es posible comenzar asignando a cualquier letra del abecedario un número, así a la A le corresponde el 1, a B el 2, a la C el 3, y así sucesivamente hasta la que le corresponderá el 26. A continuación utilizaremos estos valores para crear un *hash* sin más que sumar los números correspondientes a cada uno de los caracteres contenidos en la cadena. La cadena Visual Basic genera un *hash* de 24 porque V es la vigésimo segunda letra del abecedario y B es la segunda (22 + 2 = 24).

Conocido el *hash* con valor 24, ¿uno es capaz de obtener la cadena original? No. Este *hash* no indica la longitud de la cadena, cuál es el carácter inicial, o cualquier otro detalle sobre la cadena Original. Cuando diferentes cadenas producen el mismo valor *hash* se dice que ha habido una colisión. Un buen algoritmo de *hashing* deberá producir resultados que sean únicos y que estén libres de colisiones. SHA-1 produce resultados libres de colisione y codifica y condensa la cadena original de tal forma que se considera computacionalmente imposible obtener la cadena original.

Para comprender adecuadamente este tema se recomienda realizar la práctica relacionada al cifrado, donde se aplica esta poderosa herramienta.

7.2.2.2. OFUSCACIÓN

Como medida para dificultar la posibilidad de que un atacante realice una operación de ingeniería inversa sobre la aplicación, debe considerarse la posibilidad de ofuscar el código antes de distribuirse. Ofuscar es sinónimo de oscurecer. Cuando se aplica este sistema a una aplicación ya generada, el ofuscamiento cambia los nombres de las funciones y variables, reorganiza el código y oculta las constantes de tal forma que cualquiera que descompile la aplicación tiene grandes dificultades si intenta comprender cómo ha sido diseñada y escrita. El objetivo último del ofuscamiento es confundir a la utilidad de descompilación hasta el punto de que sea incapaz de descompilar el código.

Visual Basic .NET 2003 incluye una utilidad de ofuscamiento denominada ***Dotfuscator***, que ha sido desarrollada por *PreEmptive Solutions*. *Dotfuscator* ofrece una serie de servicios de ofuscamiento de nivel básico tales como el cambio de nombres de las variables y de los métodos privados. Si existe una función denominada CalcularImpuestos, que acepta un parámetro de tipo Decimal denominado Ingresos, el *Dotfuscator* cambiará el nombre de la función por a y el nombre del parámetro por A_O, haciendo que el que intente determinar cuál es el objetivo de la función lo tenga realmente difícil.

Por ejemplo, *Dotfuscator* cambiará el siguiente código de Visual Basic .NET

```
Private Function CalcularImpuestos(ByVal Ingresos As Decimal) As Decimal
End Function
```

por éste:

```
Private Function a(ByVal A_O As Decimal) As Decimal
End Function
```

PreEmptive Solutions dispone de otras versiones comerciales de *Dotfuscator* que incluyen funciones más avanzadas, tales como la reorganización de la lógica de ejecución de su aplicación, de tal forma que dificulta al máximo la tarea de la ingeniería inversa.

Las utilidades de ofuscamiento funcionan mejor con aquellas aplicaciones de Visual Basic .NET que cuentan con un gran número de funciones internas que realizan la mayor parte del trabajo de la aplicación. También ayuda si las funciones internas de la aplicación utilizan pocas clases de *Visual Basic .NET* o *.NET Framework* (aunque éste suele ser un requisito difícil de satisfacer porque las aplicaciones de Visual Basic .NET suelen llamar a numerosas funciones de *Visual Basic .NET* y *.NET Framework* para llevar a cabo las tareas que tengan encomendadas). El motivo es que una utilidad de ofuscamiento no puede ofuscar por completo las clases y funciones públicas contenidas en la aplicación porque siempre se deberán conservar los nombres de estas clases y funciones para que las aplicaciones externas las puedan localizar. Además, si el código utiliza clases de *Visual Basic .NET*, todas las llamadas a esas funciones no se modificarán porque la utilidad de ofuscamiento no puede cambiar el nombre a la función que ha sido llamada,

aunque ya existen utilidades de ofuscamiento más avanzadas que pueden generar código que enmascara el nombre de la función que ha sido llamada.

El objetivo del ofuscamiento no es mejorar la seguridad general de la aplicación. Después de ofuscar una aplicación se ejecutará exactamente de la misma forma en que lo hacía antes del ofuscamiento y tendrá los mismos problemas de seguridad que tuviera antes del ofuscamiento. Esto no quiere decir que al ofuscar la aplicación no vaya a conseguir una mejora parcial en la seguridad. Si puede evitar que un *hacker* realice un proceso de ingeniería inversa sobre la aplicación, podrá impedir que el *hacker* llegue a robarle su código fuente. Sin embargo, también deberá adoptar otras medidas preventivas para impedir la sustracción del código fuente y de los documentos relacionados. Para ello, por ejemplo, debe asegurarse que los servidores sólo pueden ser accedidos por aquellas personas que sean de confianza y que necesiten acceder al código fuente.

Ejecución de Dotfuscator

La herramienta no está plenamente integrada en Visual Studio .NET y la interfaz de usuario no resulta demasiado intuitiva. Los siguientes pasos muestran la forma en que podrá ejecutar *Dotfuscator*:

1. Generar la aplicación.

2. Seleccionar *Dotfuscator Community Edition* del menú Herramientas.

3. Si se desea continuar, seleccione Sí para aceptar la licencia; se le pedirá que registre el producto en *PreEmptive Solutions*.

4. Cuando pregunte por el tipo de proyecto, seleccionar *Create New Project* (Crear nuevo proyecto). En la ventana principal se mostrará la ficha *Setup*, que no contiene nada que nos vaya a ser útil.

5. Seleccionar el nivel de directiva que se desee distribuir.

6. Activar la ficha *Trigger*, hacer clic en el botón *Browse* (Examinar) y localizar los archivos binarios .EXE o .DLL de la aplicación Visual Basic .NET ya generada que se desee ofuscar. En general, puede localizarse en el directorio *Bin* de la aplicación.

7. Si se está generando una biblioteca de clase o un proyecto de control de usuario, es necesario activar la ficha *Options* y seleccionar la opción *Library*. De esta forma impedirá que *Dotfuscator* modifique los nombres de los métodos y de las clases *Public* que exponga el componente.

8. Activar la ficha *Build* (Generar) e introducir una ubicación para almacenar el archivo que haya ofuscado. Por ejemplo, puede crearse un nuevo directorio denominado ofuscado dentro de la carpeta que contenga a la aplicación, por ejemplo, C:\MiApli\Ofuscado.

9. Hacer clic en el botón *Build* (Generar). Pedirá que se guarde el proyecto ofuscado.

10. Analizar en detalle la ventana de salida para comprobar que no se han producido errores durante la generación. La aplicación ofuscada se ubicará en el directorio que eligió anteriormente.

Siempre que vuelva a generar la aplicación, tiene que volver a ejecutar el programa de ofuscamiento. Este programa guarda las opciones utilizadas en la generación y tan sólo se tiene que hacer clic en el botón *Build* para volver a ofuscar la aplicación.

7.2.2.3. GENERAR CERTIFICADOS Y VERIFICAR FIRMAS DIGITALES

Una buena alternativa es la distribución de software es por Internet. Distribuir el software por la Web resulta conveniente, tiene un bajo riesgo y resulta barato. Se trata de un fabuloso medio de distribución si es el dueño de una pequeña empresa de desarrollo de software que no tiene el volumen o el capital necesario para justificar la creación y distribución de software empaquetado. Sin embargo, la distribución por Internet carece de algunas de las bondades del software empaquetado como, por ejemplo, la posibilidad de contar con una caja cubierta de atractivos gráficos, del logotipo de su empresa y de un reluciente holograma (detalles que dan legitimidad a su software y generan confianza en el cliente). Entramos en el mundo del certificado y de la firma digital.

Un ***certificado digital*** es similar a otros certificados que prueban la identidad y que recibimos a lo largo de nuestra vida. El certificado digital contendrá la siguiente información:

• El nombre o la institución del emisor.

• El nombre y otra información relativa al dueño del certificado, que dependerá del tipo de certificado.

Los certificados se utilizan para identificar a una persona o una empresa. Por ejemplo, si desea obtener el pasaporte sólo lo conseguirá si proporciona alguna prueba de la identidad, prueba que puede ser el certificado de nacimiento. La parte emisora acepta el certificado de nacimiento como prueba porque tiene plena confianza en la organización que lo ha emitido como entidad emisora de certificados válidos que contienen información veraz. En caso necesario, la parte que acepta el certificado de nacimiento como prueba de identidad podría entablar contacto con la organización emisora para verificar sus registros y confirmar que el certificado es válido.

Certificado X.509

Puede asignar a la aplicación un certificado digital que, como sucede en el caso de un certificado de nacimiento, sirva como prueba de identificación. En la vida real se reciben una enorme variedad de certificados que demuestren la identidad, pero en un equipo informático tan sólo tendrá que utilizar un determinado tipo de certificado. Este tipo único es el certificado X.509. Es el estándar del certificado de autenticación para Windows. Los certificados X.509 se pueden utilizar en muchas situaciones, todas las cuales implican la autenticación (identificación) del usuario, aplicación o equipo frente a otro usuario, aplicación o equipo. Los certificados X.509, cuando se aplican a un paquete de instalación descargable por Internet, hacen las veces del logotipo y del holograma de la empresa que, en el otro caso, tendría que imprimir sobre la caja que contiene el software. Los certificados X.509 contienen información como el nombre de la empresa y la dirección de correo electrónico. Sus clientes podrán ver esta información antes de instalar su aplicación. La Figura 10.1 muestra un típico certificado X.509.

Firma con *Authenticode*

La firma con ***Authenticode*** es el proceso de introducir el certificado X.509 en los archivos binarios de su aplicación junto con un valor de resumen *hash* cifrado de su aplicación que la identifica de forma unívoca. La firma con *Authenticode* ha estado presente desde la versión 3.0 de Internet Explorer y ha permitido la descarga y la instalación de ciertos componentes (por ejemplo, *ActiveX*) desde Internet. Si se utiliza una aplicación compatible con *Authenticode*, tal como Internet Explorer, para descargar una aplicación firmada con *Authenticode* desde la *Web*, *Internet Exploret* realiza automáticamente la tarea de verificar la existencia del certificado X.509 y que la aplicación no ha sido modificada. La combinación del certificado X.509 y de la firma con *Authenticode* proporciona las siguientes garantías:

- Identidad del editor. Garantiza que el nombre y la dirección de la empresa que edila el software es legítima. Este paso requiere la existencia de una entidad independiente que sea de confianza para uno mismo y los clientes y que suministre el certificado X.509 para que responda de la identidad. Si un cliente confía en el certificado que ha emitido una empresa y confía también en que la información de contacto proporcionada es legítima, el cliente confiará en que cualquier *software* firmado con su certificado X.509 habrá sido editado por uno y no por otro. Si los clientes no tienen forma de identificar con cierto grado de veracidad al editor de la pieza de *software*, el cliente no puede saber nunca si la pieza de *software* ha sido en realidad editada por la empresa cuyo nombre se encuentra unido al *software*. Puede resultar bastante sencillo para cualquiera que tuviera malas intenciones editar software malévolo o defectuoso en nombre de otra empresa.

- Integridad de la aplicación. Se garantiza también que la aplicación no ha sido modificada en su camino desde el servidor al cliente. Esta garantía la proporciona el valor de resumen *hash* que identifica de forma unívoca a la aplicación, una forma de firma digital si así lo desea.

La firma de *Authenticode* se puede aplicar al ejecutable de la aplicación, a los componentes .DLL de la aplicación y al paquete de instalación (archivos .MSI o .CAB) en los que está contenida la aplicación.

Firmar con *Authenticode* implica el empleo de un certificado X.509 y también se puede decir que la firma con *Authenticode* depende del certificado X.509. Sin embargo, lo contrario no es cierto. No se puede afirmar que los certificados X.509 dependan de la firma con *Authenticode* o que estos certificados se apliquen exclusivamente a las aplicaciones Windows. Los certificados X.509 tienen numerosos empleos además de las aplicaciones firmadas con *Authenticode*. Por ejemplo, los certificados X.509 resultan de utilidad para autenticar a los usuarios que inician una sesión en un equipo utilizando una tarjeta inteligente. Además, podrá utilizar los certificados X.509 con *Internet Information Services* (IIS) para verificar que una petición proviene de un determinado cliente y para notificar que un servidor IIS garantiza dicha petición. Otro empleo será para que una aplicación cliente Web de un servidor IIS valide al servidor basándose en el certificado X.509 de este último.

Deberá firmarse con *Authenticode* los binarios que constituyen una aplicación o el paquete de instalación de la aplicación cuando vaya a distribuirse utilizando un canal de distribución no

seguro, como Internet. También debe incluirse el software que distribuya por otros medios como, por ejemplo, un anexo de un correo electrónico. Entre los tipos de aplicaciones y componentes que deberá firmar con *Authenticode* se incluyen:

- Todos los paquetes de instalación que contengan la aplicación Visual Basic .NET y cuya instalación se vaya a realizar a través de Internet como, por ejemplo, los archivos .CAB y .MSI de *Windows Installer*. En este caso, no se necesita firmar con *Authenticode* los binarios contenidos dentro del paquete de instalación; al firmar con *Authenticode* el paquete de instalación se garantiza que ninguno de los binarios contenidos en él se podrá modificar.
- Las aplicaciones *Windows Forms* de implementación automática (*no-touch*) en las que se permite la ejecución directa por Internet de la aplicación .EXE. Por ejemplo, si el usuario de la aplicación puede ejecutarla sin más que hacer clic en un botón contenido en una página Web, ejecutando el archivo http://MiEmpresa/MiAplicacionVBNet.EXE, todos los archivos .DLL que descargue dinámicamente la aplicación desde Internet también deben haber sido firmados con *Authenticode*.
- Los archivos ejecutables o los paquetes de instalación que se puedan descargar desde un sitio FTP.

Un detalle interesante es que la firma con *Authenticode* de una aplicación se verificará en las siguientes situaciones límites:

- Internet Explorer comprueba automáticamente la firma con *Authenticode* de cualquier componente *ActiveX* contenido en una página *Web* cuando se descargue el archivo .CAB que contiene el componente *ActiveX* y se utilice como parte de una página Web.
- El sistema operativo *Windows* comprueba siempre la firma de *Authenticode* de los controladores de dispositivos antes de instalarlos.
- Si utiliza un programa previo .EXE de instalación de *Windows* que a su vez lance un paquete *Windows Installer* (.MSI), el programa previo verifica normalmente la firma con *Authenticode* del archivo .MSI que se vaya a ejecutar.

En relación a esta última situación, la mayoría de los editores de software no utilizan programas previos SETUP.EXE para distribuir aplicaciones y componentes por Internet. Un problema relacionado con el empleo de los programas previos es que la propia aplicación SETUP.EXE puede ser comprometida en el camino al equipo donde se va a instalar el software. Por ejemplo, SETUP.EXE puede ser sustituida por una aplicación SETUP.EXE distinta en el camino al equipo de destino, aplicación que podría llevar a cabo la acción destructiva que quisiera. Aparte de estas tres situaciones mencionadas aquí, las firmas con *Authenticode* no se verifican de manera automática. ¿Para qué se firma con *Authenticode* una aplicación o un componente si luego la técnica de implementación que haya elegido no va a verificar de forma automática la firma con *Authenticode*?

Si un cliente, después de descargar la aplicación o componente de Internet, se encontrara con un problema poco habitual que implicara a la aplicación o componente *Windows Forms*, el cliente podría llevar a cabo una verificación manual de la firma con *Authenticode* y ver el certificado.

El cliente podría ejecutar una herramienta desde la línea de comandos tal como ChkTrust.Exe para verificar la firma digital de la aplicación o del componente. Si la verificación de la firma digital fallara o no se encontrara, sería un indicio de que la aplicación ha sido manipulada o modificada. En tal caso, se solicitara a el cliente que descargue una nueva copia del paquete de instalación (quizás desde otra ubicación más segura) y solicitar al cliente que compruebe la firma con *Authenticode* antes de proceder a instalar el paquete.

Si desea firmar con *Authenticode* la aplicación o componente, desarrolla un proceso de generación que siempre firme con *Authenticode* las aplicaciones como parte de este proceso. De esta forma no sólo garantiza la seguridad de la versión final, sino que se hace lo mismo para todas las versiones intermedias. Firmar con *Authenticode* las versiones no finales del producto también le proporcionará la oportunidad de descubrir y corregir cualquier problema que pueda surgir en la firma con *Authenticode* de la aplicación o en el empleo de la propia aplicación. Seguro que es posible esperar hasta el último momento para comprobar que está utilizando un certificado X.509 que ya ha caducado.

La ***firma con nombre seguro*** es el proceso de creación de un nombre único para la aplicación que esté compuesto por un nombre de ensamblado, un valor de resumen *hash* cifrado, el número de versión y un valor de referencia cultural. Los nombres seguros son nombres únicos a nivel global que se asignan a la aplicación o componente y que no podrá ser duplicado por nadie más. Los nombres seguros impiden que la aplicación sea sustituida por otra aplicación o componente. Una parte clave del nombre seguro es la firma de resumen *hash*. La firma es bastante similar a la firma de código *hash* aplicada a una aplicación firmada con *Authenticode*, pero con un aspecto diferenciador importante: los nombres seguros garantizan únicamente la integridad de la aplicación y no la identidad del editor. Si recuerda el análisis anterior con *Authenticode* proporciona ambas cosas: integridad de la aplicación e identidad del editor.

Las firmas con nombre seguro están compuestas por dos partes, que se incluyen en la aplicación:

- Un resumen *hash* que identifica de manera unívoca a la aplicación y que ha sido cifrado con una clave privada. En este caso, el resumen *hash* funciona como una suma de verificación; un número único calculado teniendo en cuenta todos los *bytes* contenidos en el archivo.
- La clave pública del par de claves pública/privada que se utilizará para descifrar el *hash* cuando se cargue la aplicación.

Cuando Visual Basic .NET carga una aplicación de nombre seguro, la clave pública se utilizará para descifrar el valor *hash*, volverá a calcular el *hash* correspondiente de la aplicación y se comparará este último valor con el valor *hash* que se ha descifrado. Si ambos valores coinciden la aplicación no habrá sido modificada; si, incluso, un solo *byte* del *hash* no coincide, será que la aplicación ha sido modificada o se ha corrompido. Si la aplicación se ha corrompido, se producirá una excepción de seguridad y la aplicación no se ejecutará.

Al validar la firma de nombre seguro, Visual Basic .NET garantiza la integridad de la aplicación con nombre seguro. Además, antes de cargar a través de Internet cualquier aplicación

Visual Basic .NET que haya sido firmada con *Authenticode* mediante un certificado X.509, se verificará la firma de *Authenticode* de dicha aplicación.

Utilizar nombres seguros con la aplicación proporciona los siguientes beneficios:

- Identidad única. Garantiza que ninguna otra aplicación o componente tendrá el mismo nombre seguro.
- Integridad de la aplicación. Garantiza que una aplicación no ha sido modificada después de haber sido compilada.
- Integridad de la versión. Garantiza que el nombre seguro de la aplicación, representado por el valor de resumen *hash*, se encuentra reservado para todas las versiones de la aplicación. Por ejemplo, nadie más podrá utilizar la clave privada única para firmar una aplicación utilizando el mismo nombre seguro (incluso aunque la aplicación de la otra persona tenga el mismo nombre de ensamblado, el mismo valor de referencia cultural y el mismo número de versión).
- Protección frente a falsificaciones. Los nombres seguros garantizan que nadie puede falsificar una aplicación o componente creando una aplicación o componente que tenga el mismo nombre seguro. Incluso si dos ensamblados desarrollados por el mismo proveedor comparten el mismo nombre, número de versión y referencia cultural, se garantiza la unicidad del *PublicKeyToken*, ya que cada fabricante utilizará una clave única para firmar un ensamblado con nombre seguro. La única forma de que ambos fabricantes (o un fabricante y el atacante falsificador) pudieran utilizar el mismo nombre seguro es que uno de los fabricantes (o el atacante) pudiera obtener de alguna forma la clave privada que utiliza el otro fabricante.

7.2.3. CONTENIDO DE UNA LIBRERIASEGURIDAD.VB

Una vez que se ha analizado la parte teórica de las herramientas de seguridad que proporciona el *Framework*, a continuación se presenta la utilidad del contenido de *LibreriaSeguridad.vb* que utiliza bibliotecas de `System.Security.Cryptography` y `System.Runtime.InteropServices` principalmente.

RESÚMENES HASH

El código mostrado a continuación crea un *hash* SHA- 1 de la cadena *strFuente* y asigna el resultado a la cadena *strMiHash*:

```
strMiHash = Hash.CrearHash(strFuente)
```

CIFRADO DE CLAVE PRIVADA

El código mostrado a continuación cifra la cadena *strMensajeClaro* utilizando la cadena *strLlave24* de 24 caracteres como clave y asigna la cadena cifrada a la cadena *strMensajeCifrado*:

```
strMensajeCifrado = LlavePrivada.Cifrar(strMensajeClaro, strLlave24)
```

El código mostrado a continuación descifra la cadena *strMensajeCifrado* utilizando la cadena *strLlave24* de 24 caracteres como clave y asigna la cadena descifrada a *strMensajeClaro*. Si la clave es incorrecta o si se ha modificado *strMensajeCifrado* desde la operación de cifrado, el cifrado no tendrá éxito y se provocará una excepción.

```
strMensajeClaro = LlavePrivada.Descifrar(strMensajeCifrado, strLlave24)
```

CIFRADO DPAPI

El código mostrado a continuación utiliza la Windows API `DatosProtegCifrado` para cifrar la cadena *strValor* utilizando como claves las credenciales del usuario que ha iniciado la sesión.

El valor cifrado se almacena en el directorio de datos de la aplicación personal del usuario con el nombre de <strNombreSetting>.txt.

```
Settings.SalvarEncriptado(strNombreSetting, strValor)
```

La siguiente línea de código utiliza la Windows API CryptUnprotectData para descifrar el valor almacenado en el archivo <strNombreSetting>.txt. El resultado se asigna a la cadena *strValor*:

```
strValor = Settings.CargarCifrado(strNombreSetting)
```

CIFRADO DE CLAVE PÚBLICA

El código mostrado a continuación crea y devuelve un nuevo par de claves pública/privada a la variable *strMiParLlaves*. El par de Llaves pública/privada es una estructura con ocho campos, incluyendo las Llaves pública y privada. Esta estructura es una cadena XML y debe mantenerse de forma confidencial porque la clave privada es la que se utiliza para descifrar los datos.

```
strMiParLlaves = LlavePublica.CrearParLlaves()
```

La siguiente línea de código extrae la clave pública de la clave privada *strLlavePrivada* y la devuelve en forma de una cadena a *strLlavePublica*. La clave pública se puede distribuir libremente.

```
strLlavePublica = LlavePublica.ObtenerLlavePublica(strLlavePrivada)
```

El código mostrado a continuación cifra la cadena *strMensajeClaro* utilizando la clave pública *strLlavePublica* y, posteriormente, asigna la cadena cifrada a la variable *strMiMensajeCifrado*. Tanto la clave pública como el par de claves pública/privada se pueden utilizar para cifrar la cadena. Esta función puede cifrar una cadena de 58 caracteres como máximo. Si se intenta cifrar una cadena de más de 58 caracteres el cifrado fallará.

```
strMiMensajeCifrado = LlavePublica.Cifrar(strMensajeClaro, strLlavePublica)
```

El código mostrado a continuación descifra la cadena *strMiMensajeCifrado* utilizando la clave privada *strLlavePrivada* (no se puede utilizar la clave pública para descifrar la cadena). La cadena descifrada se asignará a la variable *strMensajeClaro*. Para que el descifrado se realice con éxito necesitará pasar la cadena original cifrada y la clave privada, asociada. Si se intenta pasar una clave privada errónea o pasar la clave pública, el descifrado fallará provocándose una excepción. De igual manera, si el texto cifrado ha sido modificado desde que fuera cifrado, la operación de descifrado fallará provocándose una excepción.

```
strMensajeClaro        =        LlavePublica.Descifrar(strMiMensajeCifrado,
strLlavePrivada)
```

La aplicación directa de esta librería se utiliza en la práctica de Cifrado de este trabajo, de igual forma en dicha practica se presenta todo el código fuente de esta librería.

7.2.4. HERRAMIENTAS DEL EL SDK DE .NET FRAMEWORK

El SDK de *.NET Framework* proporciona utilidades de línea de comando, como se muestran en la tabla 7.11, que ayudan a realizar tareas relacionadas con la seguridad y probar los componentes y aplicaciones antes de implementarlos.

NOMBRE DE LA HERRAMIENTA	DESCRIPCIÓN
Herramienta de Política de seguridad de acceso a códigos (Caspol.exe)	Le permite ver y configurar la política de seguridad. Puede ver los permisos otorgados a un ensamble específico y los grupos de código a los que pertenece.
Herramienta de Prueba de certificados del editor de software (Cert2spc.exe)	Crea un Certificado de editor de software (SPC) de uno o más certificados X.509.
Herramienta de Administrador de certificados (Certmgr.exe)	Administra certificados, certifica listas de confianza (CTLs), y certifica listas de revocación (CRLs).
Herramienta de Verificación de certificados (Chktrust.exe)	Verifica la validez de un archivo firmado con un certificado Authenticode.
Herramienta de Creación de certificados (Makecert.exe)	Genera un certificado X.509 que se puede usar sólo para fines de prueba.
Herramienta de Ver permisos (Permview.exe)	Le permite ver los permisos solicitados de un ensamble.
Herramienta de PEVerify (Peverify.exe)	Determina si el proceso de compilación JIT puede verificar la seguridad de tipo del ensamble; también verifica la estructura en metadatos EXE.
Herramienta de Secutil (Secutil.exe)	Extrae información de clave pública de nombre fijo o certificados del editor Authenticode de un ensamble, en un formato que se puede incorporar en el código.
Herramienta de Establecer registros (Setreg.exe)	Cambia las configuraciones del registro correspondientes a certificados y firmas digitales.
Herramienta de Firma de archivos (Signcode.exe)	Firma un archivo ejecutable portátil (PE) con permisos solicitados, dándole más control sobre las restricciones de seguridad colocadas en sus componentes.

Herramienta de Nombre fijo (Sn.exe)	Ayuda a crear ensambles con nombres fijos, proporciona opciones para la administración de llaves, generación de firmas y verificación de firmas.
Herramienta de Almacenamiento aislado (Storeadm.exe)	Administra almacenamiento aislado, proporcionando la habilidad de enumerar los almacenes del usuario y eliminarlos.
Mscorcfg.msc	Herramienta gráfica para administrar la política de seguridad. Es un snap-in de MMC.

Tabla 7.11 Herramientas

7.3. HERRAMIENTAS Y UTILIDADES GENERALES DE SEGURIDAD EN WINDOWS

HERRAMIENTA	DESCRIPCIÓN
Microsoft Baseline Security Analizer (MBSA)	Como parte del Programa Estratégico de Protección de la Tecnología de Microsoft y en respuesta a la necesidad directa del consumidor de un método racionalizado de identificación de los fallos de configuración de seguridad más comunes, Microsoft ha desarrollado el Microsoft Baseline Security Analyzer (MBSA). Es una herramienta que realiza búsquedas centrales de errores comunes de configuración de la seguridad en equipos basados en Windows.
Servicios de Windows Rights Management 1.0 (RMS)	Estos servicios para Windows Server 2003 son una tecnología de protección de información con aplicaciones compatibles con RMS que proporcionan la plataforma necesaria para el sistema de administración de permisos de una organización.
Software Update Services (SUS)	Diseñado para simplificar el proceso de mantener la máquina basada en plataforma Windows con la última actualización de seguridad, Software Update Services habilita a los administradores para desplegar rápidamente y en forma segura las actualizaciones críticas disponibles paras sus servidores basados en Windows 2000. Software Update Services es una herramienta gratuita para que las empresas puedan crear un servidor de planificación y aprobación de parches de seguridad, envía un boletín por correo electrónico cada vez que hay disponibles nuevos parches de seguridad, actualizaciones críticas, rollups o service packs para su descarga. Microsoft Software Update Services (SUS) está diseñado para simplificar el proceso de mantener sistemas basados en Windows con la última actualización crítica de seguridad. SUS habilita a los administradores para desplegar rápidamente y de forma segura las actualizaciones de seguridad en sus plataformas de servidor Windows 2000 Server así como en las plataformas de escritorio Windows 2000

	Professional y Windows XP Professional.
HFNetChk	Un elemento especialmente importante a la hora de hacer que un sistema sea seguro es mantenerse al día en cuanto a los parches de seguridad.
Herramienta de Seguridad UrlScan	La herramienta de seguridad UrlScan previene la posibilidad de que peticiones potencialmente dañinas puedan alcanzar el servidor.
Office Update Inventory Tool	Desde una localización central, los administradores pueden ejecutar esta herramienta para averiguar cuáles son las actualizaciones que han sido aplicadas sobre Microsoft Office 200, Office XP y Office 2003, qué actualizaciones están disponibles para su instalación y qué actualizaciones se pueden aplicar solo como imágenes administrativas.
IIS Lockdown Tool	IIS Lockdown Tool permite configurar adecuadamente un servidor web de forma rápida y sencilla, donde el servidor provee únicamente los servicios que el administrador desea. Los clientes pueden usar esta herramienta para proteger instantáneamente sus sistemas contra las amenazas de seguridad.

Tabla 7.12 Herramientas

7.3.1. DETECCIÓN DE INTRUSIONES Y REGISTRO DE SUCESOS

La detección de intrusiones es el seguimiento y la notificación de una actividad no autorizada en un equipo o red monitorizados. Las intrusiones pueden provocar el cambio de los derechos de seguridad de los usuarios y archivos, la instalación de archivos que son "caballos de Troya" o el acceso inapropiado a la información. La detección de las intrusiones se lleva a cabo a través del examen de los registros del sistema y de la configuración e implementación de servidores de seguridad, software antivirus y sistemas especializados en detección de intrusiones (IDS, *Intrusion Detection Systems*).

Las herramientas de detección de intrusiones intentan monitorizar e informar de actividades malintencionadas, como son el acceso no autorizado a archivos y sistemas, ataques (distribuidos) de denegación de servicio, gusanos, archivos que son caballos de Troya, virus informáticos, problemas derivados de la saturación de los búferes, la redirección de aplicaciones, la suplantación de identidad y de datos, ataques DNS, ataques mediante correo electrónico, daños a contenido o datos, y la lectura no autorizada de información confidencial.

La detección de intrusiones implica algunas de las funcionalidades siguientes: alertas, registro, elaboración de informes y prevención. Las alertas son mensajes informativos en tiempo real y de alta prioridad que los procesos de detección de intrusiones envían a los administradores de una red con el fin de advertirles de que se ha producido un suceso de gran riesgo para la seguridad. Todos los sistemas de detección de intrusiones deben grabar los incidentes notables relacionados con la seguridad en un archivo de registro junto con la fecha y hora del suceso, el riesgo y otra información detallada. Una buena herramienta de detección de intrusiones debería proporcionar informes de administración que analicen los archivos de registro con el objeto de disponer de

estadísticas y conocer tendencias. Las herramientas avanzadas de detección de intrusiones consiguen identificarlas y evitar que el ataque tenga éxito.

Creación de un plan de detección de intrusiones

La creación de un plan de detección de intrusiones conlleva el establecimiento e implementación de directivas de seguridad, la documentación de referencias para el sistema y la configuración de herramientas que permitan monitorizar las actividades no autorizadas. La creación de un plan de detección de intrusiones incluye los pasos siguientes:

1. Inventariar los sistemas que hay que proteger
2. Crear una directiva de seguridad
3. Proteger los sistemas
4. Documentar referencias de sistemas
5. Usar y examinar los registros de sucesos
6. Activar la auditoría
7. Implementar herramientas y métodos de detección de intrusiones

1. Inventariar los sistemas que hay que proteger

Debe realizar un inventario de la red de modo que sepa lo que está protegiendo. Es especialmente importante observar qué sistemas carecen de la aplicación Visor de sucesos, NTFS o *Active Directory* puesto que la protección y administración se pueden dificultar. Considere el uso de ***System Management Server*** *de Microsoft* para automatizar la realización de inventarios y la detección de los dispositivos de red. Evalúe el riesgo de incidentes de seguridad y comunique sus conclusiones a la dirección.

2. Crear una directiva de seguridad

Los administradores de seguridad deben establecer directivas de seguridad por escrito, que describan las directrices predeterminadas para mantener protegidos los sistemas. Estas directivas deberían identificar quién puede tener acceso a qué activos informáticos, su uso aceptable, las directivas relativas a las contraseñas, restricciones de software y la configuración predeterminada de la seguridad de los sistemas. La mayor parte de las directivas de seguridad del sistema *Windows* se almacenan en el equipo local mediante la directiva de seguridad local.

3. Proteger los sistemas

Una vez identificados los sistemas, es crucial comprobar que están configurados de forma segura. Microsoft dispone de varios recursos de seguridad, herramientas, listas de comprobación y documentos de procedimientos que pueden ayudarle a reforzar la seguridad de los sistemas Windows. Las plantillas de seguridad son el método que prefiere Microsoft para proteger la configuración de seguridad de un sistema local. La herramienta Microsoft ***Baseline Security Analyzer*** analiza los puntos débiles de la seguridad de los sistemas Windows más comunes y realiza recomendaciones en consecuencia. Las guías de la *National Security Agency* ofrecen directrices relativas a los permisos de archivos y directorios. Considerese el uso de un sistema de administración de revisiones para conservar actualizados los sistemas.

4. Documentar referencias de sistema

El signo más común de intrusión es un comportamiento en el sistema o una modificación del mismo que se desvíe de lo esperado. Con el fin de saber lo que se considera normal, debe examinar los sistemas y el tráfico de red para elaborar una referencia en cuanto a la configuración y la actividad.

Uso de plantillas de seguridad para documentar y comparar

Puede utilizarse la herramienta Configuración y análisis de seguridad para ver y documentar las opciones cruciales de configuración de la seguridad. Puede utilizar la misma herramienta para comparar la seguridad actual con respecto a una plantilla predefinida.

Examen y documentación de los derechos de usuarios y grupos

Examinar y documentar las cuentas de usuario, grupos, recursos compartidos y permisos. Puede usarse el complemento Usuarios y grupos locales de Administración del equipo o la utilidad NT *User Manager* para examinar las cuentas de grupo y de usuario. Asimismo, puede usar los comandos *NET USER, NET GROUP y NET LOCALGROUP* para hacer una lista de los usuarios y los grupos. El *Kit* de recursos de Windows NT 4.0 (*Windows NT 4.0 Resource Kit*) y el Kit de recursos de *Windows 2000* (*Windows 2000 Resource Kit*) contienen docenas de utilidades para ayudar a los administradores a auditar los derechos y permisos. NETWATCH.EXE es una herramienta útil para mostrar los recursos y los usuarios conectados.

Examen de los programas y procesos activos

Use la Utilidad de configuración del sistema (MSCONFIG.EXE) para ver y documentar los programas de inicio automático en *Windows 98*, *Windows Millennium Edition* y *Windows XP*. Use el Administrador de tareas de Windows NT, Windows 2000 y Windows XP para ver las aplicaciones y procesos que se estén ejecutando en ese momento. El Kit de recursos de Windows 2000 (*Windows 2000 Resource Kit*) contiene varias utilidades para ver los procesos, como son PULIST.EXE y TLIST. Muchos administradores encuentran que PVIEW y SPYXX son herramientas útiles para examinar los servicios y procesos. Considérese la utilización de la utilidad WINDIFF para comparar un directorio o un sistema de referencia limpio frente a un sistema recién auditado.

Netstat y los puertos TCP

Documente los puertos TCP activos pertenecientes a los sistemas monitorizados. Use el comando NETSTAT para mostrar los puertos TCP activos. En los sistemas Windows XP y 2003, puede usar NETSTAT para mostrar los procesos implicados con los puertos activos.

Supervisión de red

Desde los primeros tiempos de Windows para Trabajo en Grupo 3.11, siempre ha estado disponible alguna forma del Monitor de red de Microsoft. Monitor de red es una herramienta de diagnóstico que captura los paquetes de red para examinarlos, realizar su seguimiento y

elaborar estadísticas. Monitor de red, u otra herramienta similar, es la principal utilidad para establecer referencias del tráfico de una red.

Supervisión del rendimiento

Windows siempre ha dispuesto de una utilidad de supervisión del rendimiento para monitorizar y realizar un seguimiento en tiempo real de diversos sucesos. La supervisión del rendimiento se puede utilizar para hacer un seguimiento de los objetos, procesos y procesadores con el fin de establecer umbrales de referencia. El Kit de recursos de Windows 2000 contiene un fabuloso capítulo acerca de la supervisión del rendimiento.

Deben conocerse y documentarse las averiguaciones que se realicen para establecer métodos de detección de intrusiones y detectar actividades no autorizadas cuando se produzcan en el futuro.

5. Usar y examinar los registros de sucesos

Los registros de sucesos son la herramienta principal de supervisión para realizar el seguimiento de la actividad de intrusión en un sistema en particular o contra él. Los administradores de red deberían examinarlos en busca de sucesos inesperados e investigar cualquier actividad sospechosa. Los registros de sucesos se pueden examinar de forma manual o mediante una herramienta de cotejo de registros de sucesos.

Registros de sucesos

Los sistemas Windows NT, Windows 2000, Windows XP y Windows 2003 comparten tres registros de sucesos comunes: de aplicación, de seguridad y del sistema. Tanto los administradores como los usuarios pueden tener acceso a la utilidad Visor de sucesos, en el menú Herramientas administrativas; Windows 2000, Windows XP y Windows Server 2003 tienen un complemento Visor de sucesos predeterminado para *Microsoft Management Console*. Otros registros, como los del Servidor DNS, el Servicio de replicación de archivos y el Servicio de directorio, pueden estar disponibles en función de la plataforma de Windows y de cuándo se haya instalado. Cualquier registro puede proporcionar una evidencia de una intrusión en la seguridad.

Automatizar la visualización de sucesos en varios equipos

Si tiene que supervisar más de un sistema, ir a cada equipo personalmente puede ser una tarea difícil desde un punto de vista administrativo. La utilidad Visor de sucesos le permitirá abrir los registros de sucesos de equipos remotos, si usted dispone de derechos administrativos. Microsoft proporciona otras dos herramientas para ver y administrar varios equipos a la vez.

EventCombMT

EventCombMT es una utilidad gratuita de Microsoft que se utiliza para buscar en registros de sucesos remotos a través de dominios diferentes y funciona con los sistemas Windows NT, Windows 2000, Windows XP y Windows Server 2003.

Microsoft Operations Manager

El uso de una consola de *Microsoft Operations Manager* (MOM) basada en el Web ofrece un completo conjunto de características destinadas a ayudar a los administradores a supervisar y administrar los sucesos y el rendimiento de servidores basados en Windows. Tiene un precio muy razonable y permite a los administradores de red centralizar las actividades de administración de auditorías y registros de sucesos. Se pueden crear reglas y directivas predeterminadas para administrar servidores y responder a los sucesos de los servidores que intenten infringirlas.

6. Activar la auditoría

Aunque no se activa de forma predeterminada en ninguna plataforma Windows, la auditoría se puede usar para supervisar los intentos fallidos y correctos de acceso a los objetos, como son los inicios de sesión, el acceso a archivos y directorios, los cambios de contraseña, el uso de las aplicaciones, los cambios en los derechos de seguridad, en el Registro o en las directivas, y los intentos de cerrar el sistema. Los sucesos de auditoría se graban en el registro de seguridad.

La auditoría es una herramienta eficaz, pero debe tener cuidado y no supervisarlo todo o se capturará tanta información que dejará de tener utilidad. La mayor parte de los administradores encuentran un buen equilibrio entre el costo y el beneficio que supone supervisar los sucesos fallidos, como puede ser el intento fallido de aumentar los derechos de usuario, y supervisar las actividades fundamentales, como el reinicio del sistema.

Los cambios de las cuentas de usuario se pueden apreciar si se activa la auditoría y se examina el registro de seguridad.

7. Implementar herramientas de detección de intrusiones

La detección de intrusiones se puede llevar a cabo mediante la supervisión de los sistemas en busca de actividad inesperada que se desvíe de las referencias establecidas. Toda actividad sospechosa debe investigarse. Use todas las herramientas comentadas anteriormente para realizar comparaciones e investigar. Otras herramientas especializadas de detección de intrusiones resultan de utilidad en la observación y alerta de comportamientos sospechosos que puedan no ser detectados inmediatamente en la comparación manual de las referencias.

Servidores de seguridad

Microsoft dispone de dos servidores de seguridad para supervisar e impedir los intentos de intrusión.

Servidor de seguridad de conexión a Internet

El Servidor de seguridad de conexión a Internet de Windows XP (*ICF, Internet Connection Firewall*) es un servidor de seguridad que inspecciona de forma completa el estado de los paquetes y que viene integrado en las ediciones *Home y Professional de Windows XP*. Puede utilizarse para impedir los intentos de establecer conexiones entrantes no autorizadas desde Internet.

ISA Server

ISA Server es el producto estrella de Microsoft para la detección de intrusiones. Como sucesor de la línea de productos Proxy Server de Microsoft, ISA se ha ganado la reputación de ser una sólida herramienta de seguridad de red perimetral. Al actuar como un servidor Web *proxy* con almacenamiento en caché y como servidor de seguridad, ISA Server puede realizar el filtrado de las capas de paquetes, circuito y aplicación además de la publicación del servidor, y puede formar parte de una solución de VPN. Varios grupos de renombre dedicados a la auditoría de la seguridad han encontrado que ISA constituye una eficaz herramienta de defensa para las redes.

Antivirus y software IDS

Pocas redes se podrían considerar seguras sin una protección antivirus en tiempo real, ya sea en cada estación de trabajo o en los dispositivos perimetrales de la red. Microsoft dispone de una lista de proveedores de software antivirus. Snort es un IDS común de código abierto. Aunque originalmente funcionaba con UNIX, se ha pasado a la plataforma Windows. Los detectores de vulnerabilidades, como *Microsoft Security Baseline Analyzer*, están demostrando ser una herramienta necesaria para cualquier administrador de seguridad. Otros proveedores de detectores de vulnerabilidades son *Internet Security Systems* y GFI. Además, considere la utilización de software que supervise cambios en áreas esenciales de los sistemas, como *Tripwire*.

7.4. HERRAMIENTAS DE SEGURIDAD

A continuación se presentan algunas herramientas que permiten verificar y probar la seguridad de un sistema. Es responsabilidad de cada uno evaluarlar el uso antes implementarlo o complementarla para grarantizar la seguridad en los sistemas o aplicaciones

ACTIVIDAD EN INTERNET
Air SmartGate
Cyber Snoop
Ianalyst
Internet Cleanup
Internet Manager
Internet Resource Manager
Internet Risk Management
NetFocus
System Activity Manager
Web Spy
WebTrends Firewall Suite
WinGuardian

ANÁLISIS DE RED
Abend-AID Fault Manager
Actiview Trouble Manager
AimIT
BindView
Centennial Discovery
Enterprise Security Manager
Event Log Monitor
Expert Observer
Kane Security Analyst
Link Analyst
NT Manage
NTRama
Sentinel Software Security
SPQuery
TripWire
WebTrends Netware Management

ANTI-ESPIONAJE
Ad-Search
Anticotillas Plus
FlashLock
Guideon
Hook Protect
Iprotect
PC Security Guard
Rainbow Diamond Intrusion Detector
Top Secret Office

ANTI-SPAM
Spam Buster
Spamkiller
SpmmerSlammer

ANTI-VIRUS Y TROYANOS
AntiViral Toolkit Pro
AVTrojan
AVX
BootProtect
Compucilina
eSafe Protect
Fobiasoft Guardian
F-Secure
Inoculate
InVircible Antivirus
iRis Antivirus
Kaspersky Anti-Virus
MAMSoft
Mcfee VirusScan
Norman Thunderbyte Virus Control
Norton Antivirus
Panda Antivirus Platinum
PC-cillin
Protector Plus
Quick HeaIRAV Anti Virus
Sophos
The Cleaner
Trojan Defense Suite
VirIT
ViruSafe Web

ARRANQUE
Access Denied
Boot Sentry
BootLocker
MindSoft Custody
ScreenLock
SCUA Security
Sentry
ThunderGuard
Xlock

AUDITORIA
FileAudit
Log Monitor
SecurityCharge

BACKUP
@Backup
Adsm
ArcServe
A.tQave
Backup ATM Ntwork
Backup Exec
Connected Online Backup
Data Recovery for NetWare
DtKpr
DICvlew
Drive Image
ImageCast
NetBackup
Nrtn
Novabaukup
Open File Manager
Replica
Retrospect
SurvivelT
Ultrabac

BLOQUEO Y RESTRICCIÓN
Absolute Security
AceControl
Anfibia Soft Deskman
CDLock
ChildProof
Clasp2000
Deskman
Desktop Locker 1.0
DesktopShield
DeviceLock Me
GS98 Access Control
ISS Complock
Lock n Safe
MausTrap
MicroManager
MindSoft GuardianShip
Mindsoft Restrictor
PC Lock
PC Restrictor
RedHand Pro
Securelt Pro
SecurityWizard
Smart98
StormWindow
System Security
TrueFace
Windows Security Officer
WinFile Vault
WinLock

CONTRASEÑAS
007 Password Recovery

Aadun
Absolute Security
Advanced Password Generator
Asterisco
Claves
ePassword Keeper
EXE Protector
Guardian
Info Keep
LockDown
Locker
MasterPass
Office Password
Open Pass
PassGo
PassGuard
Password Corral
Password Generator
Password Guardian
Password Keeper
Password Power
Password Tracker
Passwords
Planet.Keeper
Private Bookmarks
PwlTool
Qwallet
Random Password Generator
Secret Surfer
Software Safe
y-GO Universal Password
WMVault

CONTROL REMOTO
AMI Server Manager
ControllT
CoSesslon
Kane Security Monitor
LapLink
NetOp
NetSupport Manager
PcAnywhere
Proxy
ReachOut Enterprise
Remote Administrator
ServerTrak
Timbuktu Pro
TrendTrak

COOKIES
Cache & Cookie Washer
Cookie Crusher
Cookie Pal
CyberClean
The Watchman
Window Washer

DETECTORES DE AGUJEROS DE SEGURIDAD
Check Point RealSecure
Hackershield
Intruder Alert
LanGuard Network Scanner
Lucent RealSecure
NetProwler
NetRecon
PassMan Plus
SecureNet Pro Software
WebTrends Security Analyzer

ENCRIPTACLÓN DE COMUNICACIONES
Bbcom VPN
F-Secure VPN
Go Secure
GuardianPRO VPN
Intel VPN
KryptoGuard LAN and VPN
PGP
Power VPN
SafeGuard VPN
Sidewinder
SmartGate
SonicWall Pro
VPN 1 Internet Gateway
VPNWare System

ENCRIPTACIÓN DE SOFTWARE
ABI- CODER
Absolute Security
AutoEncrypt
BestCrypt
CodedDrag
Combo
CrypText
Cryptit
Cryptoldentify
Cryptoman
Data Safe
DataCloak
Easy Code
Easycrypto
Emerald Encryption
Encrypt IT!
Encrypted Magic Folders
Enigma

Enigma 98
File Protector
FileCrypto
FileDisk Protector
FlyCrypt
Folder Guard
Hideit! Pro
HotCrypt
InfoSafe
Interscope BlackBox
Invisible Secrets
Jumblezilia
Kremlin
Krypton Encoding System
MindSoft Shelter
Neocrypt
Norton Secret Stuff
NovaLock
Passworx
PCSafe
PGP
Quick:CRYPT
RSA Bsafe
SafeGuard LanCrypt
SafeSuite Realsecure
SECRETsweeper
SECURE
Secure Shuttle Transport
Security BOX
SecurityManager
ShyFile
SpartaCom Cryptogram
TEACrypt
Text Watchdog
The DESX Utility
ThunderCrypt
Unbreakable Encryption
WINZAP
Xcrypto

ESPIONAJE
2Spy!
Activity Monitor
AIot Monica
AppsTraka
ASCII Spy
AY Spy
Boss Everyware
Canary
Date Edit
Desktop Surveillance
El Espía
EventControl
IntraSpy
Key Logger
Keyboard Monitor
KeyKey
MyGuardian
Omniquad Detective
Password Revealer
PC Spy
RemoteView
Snooper
Spector
SpyAnywhere
Stealth Activity
Stealth Keyboard Interceptor
Stealth Logger
SupervisionCam
System Spy
Watcher
WinGuardian

FILTROS DE INTERNET
CommandView
Cyber Attack Defense System
Cyber Sentinel
Digital ID
e-Sweeper
Go Secure!
Mail-Gear
MailSweeper
MailVault
Message Inspector
Predator Guard
Private-I
Real Secure
Shields UP!
SigabaSecure
SmartFilter
WEBsweeper
World Secure Mail

FIREWALLS
Altavista Firewall
Blacklce
CheckPoint
CyberArmor
Elron Firewall
FireProof Firewall KIT System
GuardianPRO Firewall
GuardIT
HackTracer
MindSoft Firewall

Neo Watch
Netmax Firewall
Norton Personal Firewall
Raptor Firewall
Secure Connect Firewall
SmartWail
Sygate Personal Firewall
Tiny Personal Firewall
Watchguard Livesecurity SYS
WinRoute Pro
ZoneAlarm Pro

GESTIÓN DE ACCESOS
Absolute Protect
Access Manager Secondary Radius
Azza Air Bus
Border Protector
c2000
Cnet/2
Defender
E-Z Lock
GuardianPro Authentication
Hands 0ff Personal
Identity Protector
IKey
Lock Protector
Navis Access
Navis Radius Access Control
Palladium Secure Remote Access
Panda Security
Personal Protector
PrivateEXE
RSA SecurID
SafeGuard Easy
SafeWord Plus
SmartGuard
SmartLock
Steel-Belted RADIUS
UserLock
VicinID
WinFuel

MANTENIMIENTO
Dlskeeper
More Space
Partition Commander
Partition Magic
Security Setup
System Commander
Windows Commander

OCULTACIÓN
BlackBoard FileWipe
Boss
Camouflage
Don't Panic!
Hidden 7
Invisible Files
Sentry98
WebPassword
WinShred
WipeClean

RECUPERACIÓN DE DATOS
ConfigSafe Desktop
CoreSave
Easy Recovery
Easy Restore
Esupport
GoBack
Instant Recovery
Lost & Found
PictureTaker Personal Edition
SecondChance
Shredder
System Snapshot
Undelete

SEGURIDAD EN COMERCIO ELECTRÓNICO
Commerce Protector
CryptoSwift
ETrust
NetSecure
RSA Keon
SAFEsuite Decisions
Safety Net

7.5. CONCLUSIÓN

Probablemente el título de este capítulo exige mencionar más herramientas, sin embargo existen gran variedad de ellas que pueden proporcionar la misma solución solo que con distintas técnica. En este capítulo pretendí que al interesado le sirva para guiarse y tomar ideas y decisiones sobre el mecanismo o técnica que necesita implementar la aplicación. El lograr complementar las aplicaciones, con seguridad, también dependerá de un factor muy importante: "El objetivo de la aplicación".

Si el sistema trabajará con datos que son de suma importancia para uno mismo y los que proporcionaron estos datos (como direcciones, teléfonos, números de cuentas, etc.), debe hacerse de la completa idea de implementar mecanismos que mantengan a salvo la información y en este "mantener" se incluyen muchas de las características de la información (confidencialidad, disponibilidad, integridad, etc.). En cambio si el sistema es un programa de juegos, el tipo de seguridad es diferente, pero solo "diferente", y no nulo como podría pensarse, puesto que este tipo de sistemas puede requerir un mayor cuidado de la seguridad dentro del código, esto es los ciclos, en las variables e incluso en la protección del código mismo. Por lo tanto, puedo decir que el objetivo de la aplicación guiará en conjunto con este capítulo a lo largo del desarrollo para definir el tipo de seguridad que se requiere.

Así como los programas tienen sus objetivos este trabajo y en particular este capitulo también lo tienen. El mas importante es el de mostrar las técnicas y herramientas, por lo que este objetivo se cumple a la perfección. Si se consideran que en este trabajo de tesis también exciten practicas de desarrollo, dichas practicas se convierten en herramientas y técnicas para aplicarlas de manera directa al momento de programar y diseñar en Visual Basic .NET.

Algunas personas pueden diferir conmigo en realizar en esta conclusión una nueva estrategia de seguridad más, pues dirán que debería haberla incluida en el capítulo anterior, sin embargo, mi intención es la de mostrar que una estrategia de seguridad no solo la encontraremos por su nombre, sino que es posible encontrarla y construirla en cualquier momento siempre y cuando se tenga claro el objetivo de la estrategia.

Como parte del capítulo se presentaron algunas medidas que pueden adoptarse para aumentar la seguridad de la aplicación a la hora de distribuirla o implementarla en diversos equipos. Entre estas medidas puedo citar: la firma con *Authenticode*, las estructuras de excepciones y la manipulación de errores, etc. Como parte de esta conclusión quiero constituir una estrategia para la comprobación de la seguridad en las aplicaciones, que proporcione las medidas que se deben tomar para proteger las aplicaciones, considerando un orden para implantar dichas medidas, tal y como se desarrollaron las estrategias a lo largo del trabajo de tesis.

1. Crear una versión libre "*release*" de la aplicación, y no una versión de depuración "*debug*". Esto obligará a asegurarse de que todos los valores de las constantes preprocesadas, que se comprobarán en el código utilizando la instrucción *If . . . Then*, se definen apropiadamente. De ser posible utilícese el atributo *AssemblyDelaySign* si entre los objetivos se encuentra asignar posteriormente un nombre seguro a la aplicación.
2. Ofuscar la aplicación si así lo desea.

3. Firmar la aplicación con un nombre seguro.
4. Firmar con *Authenticode* la aplicación utilizando un certificado X.509. Este paso no será necesario si se opta por firmar con *Authenticode* el propio paquete.
5. Crear el paquete de distribución (o implementación) para la aplicación. Verifiquese que incluye todos los binarios (.EXE o .DLL) con nombres seguros en el paquete de implementación. Siempre que re-genere un binario que tenga asignado un nombre fuerte, debe repetirse todos los pasos para volver a firmar con nombre fuerte el binario y volver a crear el paquete de implementación.
6. Firmar con *Authenticode* el paquete de implementación. Puede realizarse esta operación cuando genere el propio paquete de implementación (distribución).
7. Si la aplicación requiere actualizar la directiva de seguridad de .NET, crear el paquete de implementación que contenga las actualizaciones de la directiva de seguridad de .NET.
8. Analizar todos los archivos que formen el paquete de implementación en busca de virus, incluyendo el propio paquete de implementación.
9. Comprobar que la aplicación se instala y se ejecuta adecuadamente en todos los entornos de destino.
10. Comprobar que todos los archivos se han firmado en la forma adecuada.

Aunque debe implementarse la aplicación de una vez, la implementación (o distribución) se debe tratar como un proceso. De forma que se busquen los posibles errores que pueda contener el código, así como los errores que puedan encontrarse en el proceso de implementación. Deben diseñarse y llevarse a cabo el proceso de distribución al mismo tiempo que se diseña e implementa la aplicación. Cada vez que crea una nueva versión de la aplicación, debe volver a ejecutarse los pasos del proceso de implementación y crear un paquete de distribución para dicha versión.

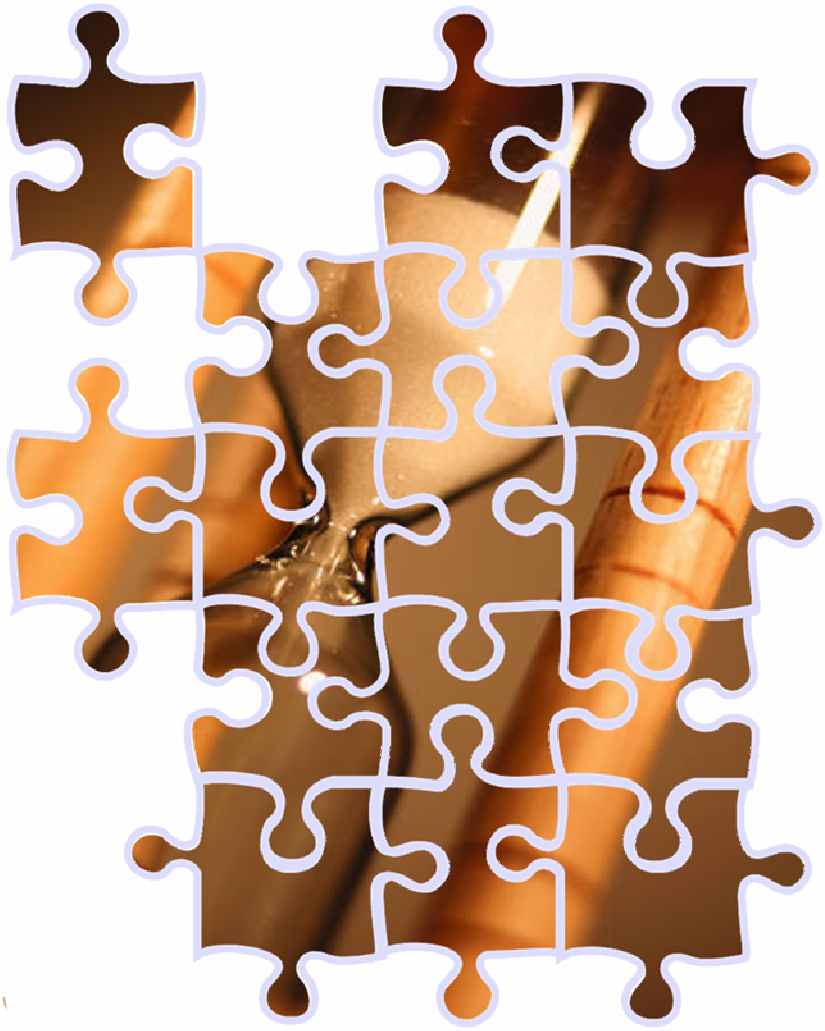
Configurar la aplicación que esté desarrollando de la misma forma en que espera que los solicitantes la vayan a instalar y probar. Así, puede comprobarse simultáneamente el comportamiento de la aplicación y el proceso de implementación, evitando cualquier sorpresa cuando se produzca la versión final.

Como se puede observar esa estrategia aplica muchos de los conceptos que a lo largo del capítulo se trataron y estudiaron.

Por otra parte debe tenerse conciencia que la gran cantidad de software que se genera en torno a la seguridad es muy amplio y aprenderla a manejar es prácticamente imposible, por lo que se anexa una lista de nombres de software que se pueden aplicar en distintos casos y dependerá de las necesidades e interés que cada programador y administrador investigar del paquete.

Capítulo 8

Conclusiones



8. CONCLUSIONES

Muchas son las conclusiones que a mi mente llegan y que realmente me cuestan trabajo exponerlas y ligarlas entre sí, cierto es que son comunes a este trabajo y a los nuevos pensamientos adquiridos en el desarrollo de la tesis. Pues bien, considerando los procesos y las estrategias de análisis, se presentaron, estudiaron y evaluaron las estructuras, métodos y herramientas que ofrecen los programas de Microsoft Visual Studio .NET, a la par se construyeron herramientas y mecanismos para lograr mayores niveles de seguridad en el uso y aplicación de estrategias de seguridad.

Les presenté una explicación muy amplia de los principales conceptos de seguridad informática existentes en .NET. Apoyado de un lenguaje de programación muy poderoso y completo como lo es Visual Basic .NET en el entorno de este trabajo terminado.

Se identificaron las principales problemáticas de seguridad a las que están expuestos los programadores, a los que se enfrentan y deben solucionar cuando desarrollan aplicaciones en Visual Basic .NET. También se diseñaron y establecieron mecanismos y estructuras idóneas, a partir de un conjunto de reglas y estrategias, que permitieron robustecer la seguridad en aplicaciones con Visual Basic .NET.

Propuse una técnica para establecer las estrategias de seguridad antes y después del diseño. Lo anterior fue complementado con la propuesta de herramientas y técnicas que permiten completar la seguridad en las aplicaciones de Visual Basic .NET.

En cuanto a los temas de seguridad, se estableció la importancia de tener los sistemas protegidos ante posibles ataques. Desde el punto de vista del usuario o del administrador de sistemas, es muy importante tener el sistema actualizado día a día, de manera que la última versión de un programa es mucho mas probable que tenga corregidos los posibles fallos de seguridad de versiones anteriores.

Se aterrizó en que la gran mayoría de los ataques que se producen son debido a un error de programación en una aplicación y que al diseñar los sistemas, los programadores podemos tratar de disminuir en gran medida los fallos de seguridad de nuestros programas con un poco de más atención. Ahí es donde estiba la importancia de haber mostrado cómo proteger los programas en la medida de lo posible. A partir de todas las problemáticas en torno a la programación, fue posible comprender porqué se dice que el programa perfecto no existe y en consecuencia no existe el sistema 100% seguro, pero no obstante esta premisa, se estudiaron técnicas, herramientas, mecanismos y sobre todo estrategias que permiten escribir código seguro para que la posibilidad de aprovecharse de algún fallo, para atacar la seguridad del sistema, sea el menor posible y no produzca pérdidas de datos e información que es lo más importante en las instituciones y empresas.

En este trabajo presento conceptos y ejemplos que son de utilidad a los programadores y diseñadores interesados en incursionar en el área del diseño de programas de computación con seguridad y en consecuencia con calidad. Las posibilidades y oportunidades presentadas por la plataforma .NET no se deben despreciar ya que dentro de ella es posible desarrollar y aplicar

herramientas, de fácil uso y disponibles para ingenieros en computación conscientes de la necesidad de sistemas seguros.

A medida que las redes e Internet se convierten en el ambiente *standard* entre empresas, la utilización de estas herramientas de programación en redes es cada vez mayor. Esto incrementa la necesidad de la eficiencia en los procesos de diseño arquitectónico para ayudar a resolver problemas específicos como la seguridad, de una manera rápida y eficiente.

Si es necesaria mi opinión sobre el impacto de la plataforma .NET, respecto a la seguridad informática en sus aplicaciones, desde una postura de desconocimiento técnico de la misma, situación en la que me encontré antes de comenzar este trabajo, hubiera dicho que si era posible que .NET tuviera un hueco en cuestiones de seguridad, por el simple hecho de la posición negativa en que se encuentran las estrategias y mercadotécnias que posee Microsoft.

Ahora he adquirido una visión técnica de la plataforma y de la seguridad que ofrece, y tengo la certeza que me encuentro frente a un posible exitoso proyecto que es consciente de las necesidades que exige la seguridad informática dentro de .NET. Me encuentro ante una nueva plataforma de desarrollo de componentes y de aplicaciones Web que incorporan una serie de nobles ideas, superando en muchos aspectos a otras plataformas y sistemas con lo que había trabajado de software libre. Así, el *framework* para el desarrollo de aplicaciones es superior al proporcionado en sistemas como Java o J2EE, tanto por su diseño como por su rendimiento.

Microsoft, como empresa, es consciente de las necesidades que se exigen en cuestiones de seguridad, reafirmando el hecho de que el futuro desarrollo del software pasa por el proceso técnico basado en sus componentes. El modelo de componentes de .NET supone un gran salto hacia el frente, con respecto a la anterior tecnología de Microsoft, como lo fue el modelo COM. Se ha logrado una integración entre los distintos lenguajes nunca vista hasta ahora. Asimismo, se reafirma el hecho de que la orientación a objetos y los sistemas de componentes, si bien son independientes entre sí, están estrechamente relacionados.

Otro foco de atención lo constituyen los dispositivos móviles en el futuro (el presente), en gran parte de los accesos a Internet que se producirán por medio de dichos dispositivos, lo que los convierte en una fuente potencial de acceso a los servicios Web. Motivo por el cual lanzó la versión de .NET para los dispositivos móviles y donde, evidentemente, también exigirán medidas de seguridad aun más completas y complejas que las existentes.

De manera particular, lo expuesto en este trabajo, espero haya quedado claro en la idea de las diferencias existentes entre el control de excepciones estructurado y no estructurado, así como las ventajas que se derivan de las capacidades del control estructurado de Visual Basic .NET. Generalmente, este tipo de control cumple con las necesidades del usuario, aunque en ciertas situaciones resulte más conveniente optar por el método no estructurado.

Aunque mi recomendación fue asegurarse de que las excepciones estén controladas, no es conveniente abusar a la hora de iniciarlas, ya que pueden ocasionar problemas de rendimiento. El control de excepciones resulta eficaz pero su inicio se debe reservar para cuando aparezcan verdaderas condiciones de excepción.

Como resultado de esta tesis profesional, es posible tomar y crear juicios acerca del uso de esta tecnología. Uno de estos juicios es el de evaluar las capacidades de procesamiento que los equipos y sistemas deben tener, principalmente a partir de las consideraciones dentro de un análisis de los datos que se manejan en el sistema.

Este análisis debemos realizarlo a partir de una revisión exhaustiva de la información, donde se debe asegurar o restringir el acceso, así como que debe estar disponible para todos los usuarios que se apoyan en este sistema. Para lograr este fin debemos diseñar y establecer las herramientas o estrategias que permitan cubrir las necesidades del sistema o bien programar y planear nuevas herramientas o estrategias que contengan los límites necesarios para proteger los datos necesarios.

Continuando con estas propuestas, será necesario establecer los privilegios que tendrá cada usuario o grupo de usuarios dentro del sistema. Apoyándose en estrategias de seguridad construidas durante el diseño del sistema y llevados acabo con anterioridad. Con ello es posible crear grupos de usuarios o roles de seguridad que contenga el número exacto de personas que pueden acceder a la información disponible para las asignaciones o tareas.

El diseño de sistemas debe tener características suficientes para asegurar la comunicación con el usuario y en muchos casos con otros sistemas. Aprovechando las bondades del lenguaje de programación Visual Basic .NET se puede lograr con las encriptación y manejo de llaves ofrecidas por los paquetes y extensiones que se estudiaron a lo largo del este trabajo.

El uso de esta tecnología permite establecer certificados y firmas digitales por lo que es conveniente el uso de los mismo para verificar que los objetos pasados por los sistemas son de un contenido confiable y no obtener entradas maliciosas o algún otro problema de los mencionados en este trabajo.

En cuanto a los accesos realizados por intrusos externos al sistema o no permitidos, es posible neutralizarlos mediante herramientas como *firewall* cuando estén bien configurados, para restringir el paso de paquetes de Internet hacia sistemas en cuestión. El uso de encriptación u ofuscación no es garantía de seguridad en este tipo de ataques y es por esto que la implementación de este firewall es recomendable para colocar algunos límites mas, agregando a los sistemas o programas algunas líneas extras de defensa.

Regresando a los beneficios que este trabajo se propuso lograr, es importante mencionar que permite crear estrategias de seguridad considerando herramientas y técnicas para robustecer los sistemas, el uso más claro es el de los algoritmos de encriptación para aumentar la seguridad de los sistemas y de esta manera facilitar el uso y aplicación dentro de las propias estrategias.

En general, el uso de este tipo de tecnología es factible siempre y cuando se tomen las medidas de seguridad necesarias para lograr un sistema confiable. Las ventajas que se ofrecen en esta propuesta de tesis, son varias al poder conectar diversos equipos y periféricos a la red o sistemas sin la necesidad de la configuración de los mismos. También se tienen las ventajas de aparición de nuevos dispositivos que se pueden usar con esta tecnología para aprovechar las posibilidades que se ofrecen.

Para concluir con los temas abarcados en esta tesis, quiero decir que los sistemas que utilizan este tipo de tecnología son fáciles de aplicar, que requieren poco mantenimiento durante el funcionamiento del mismo y los sistemas sobre los cuales se ejecutan, sólo en casos especiales, requieren configuración al utilizar la tecnología .NET para la correcta instalación o ejecución.

Gracias a las ventajas ofrecidas por la plataforma .NET, es realmente segura la instrumentación en sistemas dentro de las instituciones. Ciertamente es una tecnología que requiere de una inversión monetaria, sin embargo proporciona a cambio funcionalidad, estabilidad y cobertura de requerimientos específicos por los sistemas de la misma plataforma. A su vez, permite el establecimiento de permisos y restricciones al uso de información y servicios incorporados por Microsoft.

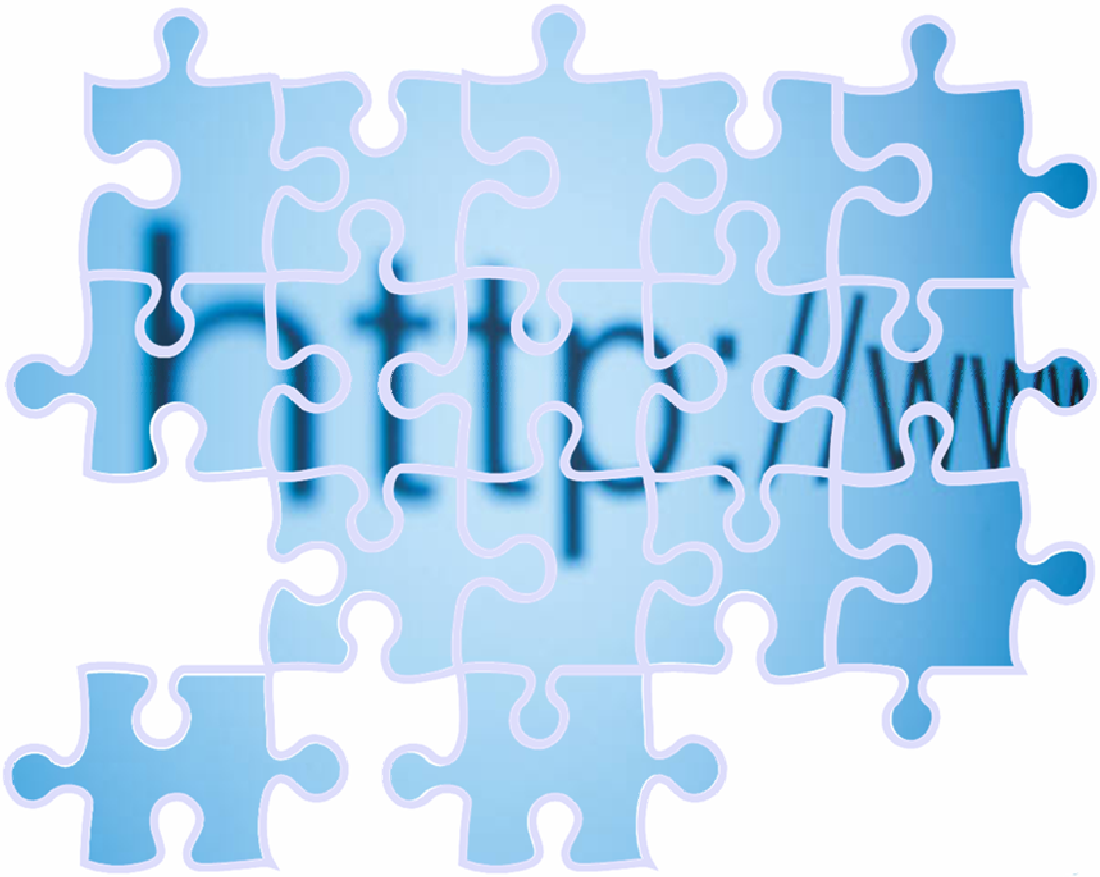
Para comprender y estudiar el uso de dispositivos emergentes es necesario un estudio por separado que requieren precisiones de clases específicas para su uso y por lo tanto otra extensión al sistema dentro de los servidores.

La propuesta de tesis desea también mostrar que en el caso de optar por esta tecnología, es necesario tomar en consideración los requerimientos y necesidades mencionadas durante el desarrollo de esta propuesta, principalmente acerca del establecimiento de medidas de seguridad estudiadas. Es necesario tomar su tiempo para realizar una estrategia y un análisis dentro del diseño, la programación e implementación de un sistema basado en leguajes como Visual Basic .NET dentro de la Tecnología .NET.

Por último, deseo terminar este orden de ideas y de conclusiones, manifestando que la verdadera seguridad informática no nos es posible conseguirla hasta que cada uno, como programador, opte por prácticas y desarrollos de una programación segura. Hasta que las estrategias y técnicas de programación segura estén lo suficientemente extendidas y aprendidas, al grado tal que desde los aprendices hasta los más experimentados programadores, todos seamos capaces de realizar búsquedas exclusivas de errores y huecos de seguridad que invadan nuestro mundo de la seguridad informática en la programación.

Al hablar de la práctica y desarrollo de programación segura, me refiero como tal a un proceso, que cuanto más se aplique y se utilice, nos encontraremos mas cómodos dentro de él, será, en consecuencia, más fácil de aplicar y constuir estrategias sin perderse en el camino de los objetivos por lo cuales se está programando. Al final, espero que este tipo de trabajos proporcionen información relacionada a la seguridad entre los programadores, de forma tal que permita ayudar y colaborar, si se presenta algún problema, explicando como punto de partida que la seguridad es importante tanto para programadores como para usuarios.

Bibliografía y Fuentes de Información



BIBLIOGRAFÍA Y FUENTES DE INFORMACIÓN UTILIZADAS EN EL PRESENTE TRABAJO POR CADA CAPÍTULO

CAPÍTULO 2 ~ PLATAFORMA MICROSOFT .NET ~

📖 **BÜHLER** Erich R. ***Visual Basic .NET Guía de migración y actualización***, Primera Edicion, McGraw-Hill, Madrid, España 2002. ISBN:84-481-3271-8

🗐 **CARPE GARCÍA** Francisco. ***Estudio de la plataforma .NET***, Facultad de Informática, Universidad de Murcia, 11 de Diciembre de 2001

🗐 **CONARD, DENGLER, FRANCIS, GLYNN, HARVEY, SHORT**, ***Introducing .NET***, Wrox, 2000.

🎬 **LARRIERA** Gustavo, ***Microsoft .NET Framework Algunos aspectos internos de la plataforma***, Microsoft MVP, 2002

💿 **MICROSOFT**, Comunidad Academica .NET, ***MicroNet*** **http://www.microsoft.com/mexico/msdn/cdacademico**

🗐 **RICHTER**, ***Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web***, Octubre 2000. **http://msdn.microsoft.com/msdnmag.**

📖 **SHAPIRO** Jeffrey R. ***Visual Basic .NET. Manual de referencia***, McGrawHill, España 2003, ISBN: 84-481-3713-2

CAPÍTULO 3 ~ PRINCIPALES CONCEPTOS DE SEGURIDAD EN .NET ~

🗐 **BURKE** S. ***Writing Custom Designers for .NET Components***, Microsoft MSDN, Junio 2001. **http://msdn.microsoft.com/library.**

🗐 **DON** Box, ***The Security Infrastructure of the CLR Provides Evidence, Policy, Permissions, and Enforcement Services***, Septiembre 2002, **http://msdn.microsoft.com/msdnmag/issues/02/09/default.aspx**

🗐 **FOUNDSTONE, INC. AND CORE SECURITY TECHNOLOGIES**, ***Analysis of Security in the Microsoft.NET Framework***, **http://www.corest.com, http://www.foundstone.com**

📖 **HOWARD** Míchael, **LEBLANC** David. ***Writing Secure Code***, Microsoft Corporation Press, EE.UU. 2003, ISBN 0-7356-1588-8

📖 **JAMSA** Kris. ***Superutilidades para Visual Basic .NET***, Primera edición, McGrawHill, Madrid España 2003. ISBN: 84-481-3710-8

💿 **MICROSOFT**, Comunidad Academica .NET, ***MicroNet***
http://www.microsoft.com/mexico/msdn/cdacademico

📖 **ROBINSON** Ed, **BOND** Michael. ***Security for Microsoft Visual Basic .NET***, McGrawHill, Washington EE.UU. 2003, ISBN: 0-7356-1919-0

🗐 **SANGHAVI** K. ***Code Access Security***, Noviembre 2001. **http://www.csharptoday.com.**

🗐 **VILCINSKAS** Markus, ***Security Entities Building Block Architecture***, Microsoft Solutions Framework,
http://www.microsoft.com/technet/security/bestprac/bpent/sec2/secentbb.mspx

CAPÍTULO 4 ~ CARACTERÍSTICAS Y ELEMENTOS DEL ENTORNO DE DESARROLLO CON VISUAL BASIC .NET ~

📖 **BLANCO** Luis Miguel. ***Programación en Visual Basic .NET***, Grupo Eidos, Madrid España 2002,

📖 **BÜHLER** Erich R. ***Visual Basic .NET Guía de migración y actualización***, Primera Edicion, McGraw-Hill, Madrid, España 2002. ISBN:84-481-3271-8

🎬 **FREIBERGER** Joerg M, **SEARA** Daniel A. ***Visual Basic.NETMicrosoft .NET*** Developer Tools Readiness Kit, Buenos Aires – ARGENTINA NDSoft

📖 **JAMSA** Kris. ***Superutilidades para Visual Basic .NET***, Primera edición, McGrawHill, Madrid España 2003. ISBN: 84-481-3710-8

💿 **MICROSOFT**, Comunidad Academica .NET, ***MicroNet***
http://www.microsoft.com/mexico/msdn/cdacademico
💿 **MICROSOFT DEVDAY 2004**,***Guia y Curso de Microsoft Visual Studio.NET,*** Comunidad MSDN, **http://www.microsoft.com/latam/vstudio**

📖 **SHAPIRO** Jeffrey R. ***Visual Basic .NET. Manual de referencia***, McGrawHill, España 2003, ISBN: 84-481-3713-2

🗐 **SOM** Guilermo, ***Curso de iniciación a la programación con Visual Basic .NET***, **http://www.elguille.info/NET/cursoVB.NET/indice.htm**

CAPÍTULO 5 ~ PROBLEMAS DE SEGURIDAD PARA PROGRAMADORES DE VISUAL BASIC .NET ~

BROWN Keith, ***Beware of Fully Trusted Code***, Microsoft MSDN, Abril 2004, **http://msdn.microsoft.com/msdnmag/issues/04/04/SecurityBriefs/default.aspx**

GUMBINER Gary M., ***Como implementar seguridad en .NET***, DEVT1-04, Microsoft Corporation, **http://msdn.microsoft.com.**

HOWARD Michael, **BROWN** Keith, ***Defend Your Code with Top Ten Security Tips Every Developer Must Know***, Septiembre 2002, **http://msdn.microsoft.com/msdnmag/issues/02/09/SecurityTips/default.aspx**

HOWARD Míchael, **LEBLANC** David. ***Writing Secure Code***, Microsoft Corporation Press, EE.UU. 2003, ISBN 0-7356-1588-8

MÁRQUEZ R. Carlos Alexei, ***Escribiendo Código Seguro: Trustworthy Computing*** , Seminario Microsoft DevDays 2004

Mejorando la Seguridad de Aplicaciones Web, Amenazas y Contramedidas, **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp**

REYNOLDS-HAERTLE Robin. ***Problemas de seguridad para los programadores de Visual Basic .NET y Visual C# .NET***, Microsoft Corporation, Enero de 2002 **http://www.microsoft.com/spain/msdn/articulos/archivo/280602/voices/vbtchSecurityConcerns.asp**

ROBINSON Ed, **BOND** Michael. ***Security for Microsoft Visual Basic .NET***, McGrawHill, Washington EE.UU. 2003, ISBN: 0-7356-1919-0

SCHMIDT Jeff. ***Guía Avanzada Seguridad en Microsoft Windows 2000***, Prentice Hall, Madrid 2001, ISBN: 84-205-2973-7

TANENBAUM Andrew S. ***Sistemas Operativos Modernos***, Prentice Hall, México 1993, ISBN: 0-13-588187-0

TIEMANN Michael, **MOORE** Sandra, **FOX** Tammy. ***Guía del Administrador RedHat***, Grupo Anaya, Madrid 2003, ISBN: 84-415-1498-4

CAPÍTULO 6 ~ ESTRATEGIAS PARA ROBUSTECER LA SEGURIDAD EN APLICACIONES VB.NET ~

BENSON Christofer. ***Security Strategies***.
Inobits Consulting (Pty) Ltd. 2000 Microsoft Solutions.
http://www.microsoft.com/technet/security/bestprac/bpent/sec1/secstrat.mspx

BENSON Christopher. ***Security Planning***,
Inobits Consulting (Pty) Ltd . 2000 Microsoft Solutions,
http://www.microsoft.com/technet/security/bestprac/bpent/sec1/secplan.mspx

BORGHELLO Cristian F. ***Tesis Seguridad Informática: Sus Implicaciones e Implementación***, Argentina 2001,
www.cfbsoft.com.ar

BROWN Keith, ***Beware of Fully Trusted Code***,
Abril 2004,
http://msdn.microsoft.com/msdnmag/issues/04/04/SecurityBriefs/default.aspx

BUDD Christopher. ***Security Threats***,
2000 Microsoft Corporation.
http://www.microsoft.com/technet/security/bestprac/bpent/sec1/secthret.mspx

FERNANDEZ Carlos M***. Seguridad en Sistemas Informáticos***.
Ediciones Díaz de Santos S.A. España 1988.

GUTIÉRREZ J.A., ***Una Introducción a la Seguridad en Sistemas Informáticos***,
http://webdiis.unizar.es/~spd/seguridad.pdf
http://www.solocursos.net/seguridad_informatica_slckey14063

HOWARD Michael, **BROWN** Keith. ***Defend Your Code with Top Ten Security Tips Every Developer Must Know***, Septiembre 2002,
http://msdn.microsoft.com/msdnmag/issues/02/09/SecurityTips/default.aspx

HOWARD Míchael, **LEBLANC** David. ***Writing Secure Code***,
Microsoft Corporation Press, EE.UU. 2003, ISBN 0-7356-1588-8

MÁRQUEZ R. Carlos Alexei, ***Escribiendo Código Seguro: Trustworthy Computing*** , Seminario Microsoft DevDays 2004

MICROSOFT DEVDAY 2004,***Guia y Curso de Microsoft Visual Studio.NET,***
Comunidad MSDN, **http://www.microsoft.com/latam/vstudio**

🗐 **REYNOLDS-HAERTLE** Robin. ***Problemas de seguridad para los programadores de Visual Basic .NET y Visual C# .NET***,
Microsoft Corporation, Enero de 2002**,**
http://www.microsoft.com/spain/msdn/articulos/archivo/280602/voices/vbtchSecurityConcerns.asp

🕮 **ROBINSON** Ed, **BOND** Michael. ***Security for Microsoft Visual Basic .NET***,
McGrawHill, Washington EE.UU. 2003, ISBN: 0-7356-1919-0

🗐 **STRASSMANN** Paul A. ***El arte de presupuestar: como justificar los fondos para seguridad Informatica***.
http://www.nextvision.com

🕮 **TIEMANN** Michael, **MOORE** Sandra, **FOX** Tammy. ***Guía del Administrador RedHat***, Grupo Anaya, Madrid 2003, ISBN: 84-415-1498-4

🎬 **VALENZUELA** R Víctor. ***Auditoría Computacional TEMA 7 Seguridad Informática***

🗐 **VILCINSKAS** Markus. ***Security Entities Building Block Architecture,***
Microsoft Solutions**,**
http://www.microsoft.com/technet/archive/security/bestprac/bpent/bpentsec.mspx

CAPÍTULO 7 ~ HERRAMIENTAS Y TÉCNICAS PARA ASEGURAR PROGRAMAS Y SISTEMAS DE *VISUAL BASIC .NET* ~

🕮 **BÜHLER** Erich R. ***Visual Basic .NET Guía de migración y actualización***,
Primera Edicion, McGraw-Hill, Madrid, España 2002. ISBN:84-481-3271-8

🗐 **DHAWAN** Priya, ***Comparación de rendimiento: Opciones de diseño de Seguridad, Generación de aplicaciones distribuidas con .NET***,
Microsoft Developer Network, Octubre de 2002,
http://www.microsoft.com/spanish/msdn/articulos/archivo/121202/voices/bdadotnetarch15.asp

🗐 **FRANCIS** Cat, ***Introducción al control de excepciones en Visual Basic .NET***,
Equipo de Visual Studio, Microsoft Corporation, Febrero de 2002
http://www.microsoft.com/spanish/msdn/articulos/archivo/120402/voices/vbtchexceptionserrorsinvisualbasicnet.asp

🗐 **GETZ** Ken, ***Control de errores en Visual Basic .NET***,
Actualización a Microsoft .NET, MCW Technologies, Febrero de 2002,
http://www.microsoft.com/spanish/msdn/articulos/archivo/260402/voices/errhandlvbnet.asp

📖 **JAMSA** Kris. ***Superutilidades para Visual Basic .NET***, Primera edición, McGrawHill, Madrid España 2003. ISBN: 84-481-3710-8

🗐 **MICROSOFT**, ***Forma rápida de detectar intrusiones y registrar sucesos***, 19 de enero de 2004,
http://www.microsoft.com/spain/technet/recursos/articulos/welcome3.asp

💿 **MICROSOFT**, Comunidad Academica .NET, ***MicroNet***
http://www.microsoft.com/mexico/msdn/cdacademico

📖 **ROBINSON** Ed, **BOND** Michael. ***Security for Microsoft Visual Basic .NET***, McGrawHill, Washington EE.UU. 2003, ISBN: 0-7356-1919-0

📖 **SCHMIDT** Jeff. ***Guía Avanzada, Seguridad en Microsoft Windows 2000***, Prentice Hall, Madrid 2001, ISBN: 84-205-2973-7

📖 **SHAPIRO** Jeffrey R. ***Visual Basic .NET. Manual de referencia***, McGrawHill, España 2003, ISBN: 84-481-3713-2
📖 **TANENBAUM** Andrew S. ***Sistemas Operativos Modernos***, Prentice Hall, México 1993, ISBN: 0-13-588187-0

🕸 **HERRAMIENTAS DE SEGURIDAD**
http://www.microsoft.com/latam/technet/seguridad/

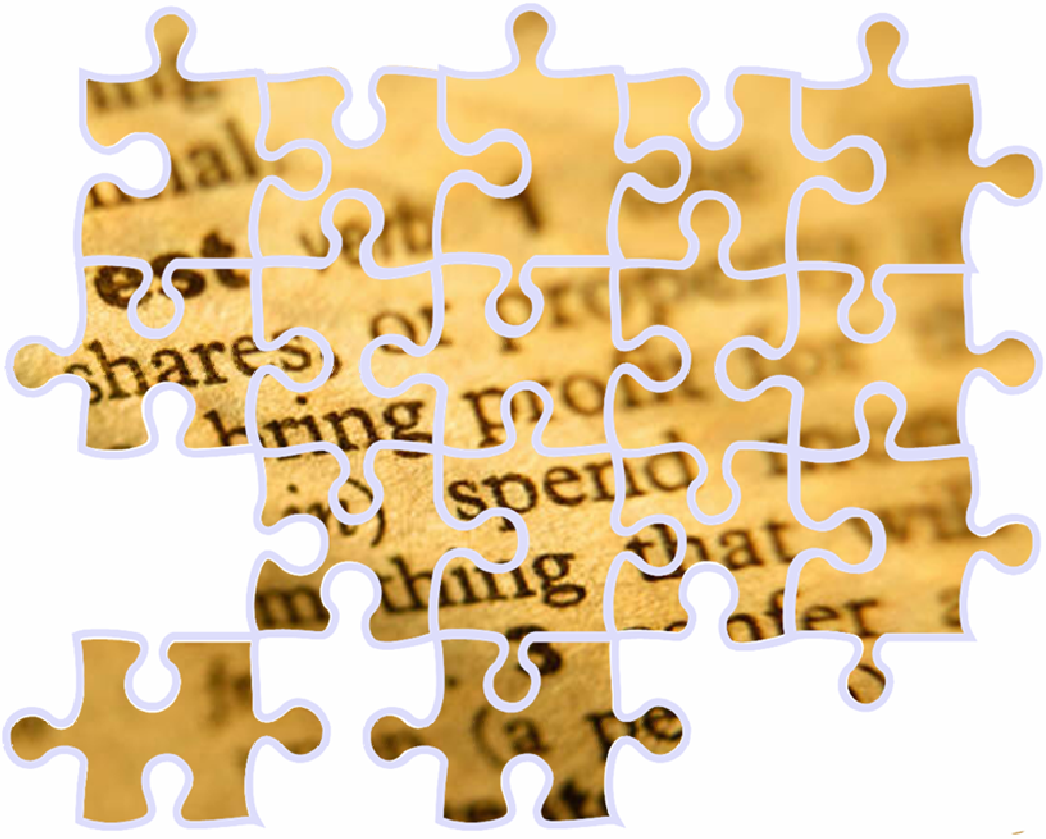
🕸 **CENTRO DE SEGURIDAD MSDN**
http://msdn.microsoft.com/security
http://msdn.microsoft.com/net
http://msdn.microsoft.com/vstudio/nextgen/
http://msdn.microsoft.com/library/

🕸 **CERT**
http://www.cert.org/stats/

Símbolos utilizados para representar el tipo de fuente:

- 📖 Libro, Manual o Tesis
- 🗐 Artículo o Nota
- 🎬 Presentación o Diapositivas
- 💿 Material Multimedia (CD-ROM)
- 🕸 Sitio Web

Glosario



GLOSARIO DE TÉRMINOS DE SEGURIDAD INFORMÁTICA Y PROGRAMACIÓN EN VISUAL BASIC .NET

A	
acceso (access)	Con respecto a la privacidad, es la habilidad de un individuo para ver, modificar y refutar lo completa y precisa que pueda ser la *información personal identificable (PII)* reunida sobre él o ella. Acceso es un elemento de las *Prácticas Honestas de Información*.
actualización (update)	Es el término que se utiliza para identificar todos los diferentes tipos de paquetes que pueden hacer que un sistema esté al día, incluyendo *hotfixes*, *acumulados*, *Service Packs*, y otros paquetes que incluyan características. Las actualizaciones se caracterizan por la severidad del tema que tratan. Algunas actualizaciones son críticas mientras que otras son recomendadas.
actualización de Windows (Windows update)	1. Es un sitio Web de Microsoft mantenido por el grupo de productos Microsoft Windows con la finalidad de proporcionar actualizaciones para los componentes de Windows. 2. Es una aplicación en Windows que permite a un usuario descargar archivos de Internet que son necesarios para mantener a su PC al día.
acumulado (rollup)	Es un conjunto acumulativo de *hotfixes* empaquetados juntos para una fácil implementación. Un acumulado tiene como objetivo un área o un componente específico del producto.
administración con privilegios mínimos (least privilege administration)	Es una práctica de seguridad recomendada en la cual a cada usuario se le proporciona solamente los *privilegios* mínimos necesarios para llevar a cabo las tareas que está autorizado a realizar.
administración de derechos digitales DRM	Es una tecnología que proporciona protección permanente a la información digital utilizando *encriptación*, *certificados* y *autenticación*. Los destinatarios o usuarios autorizados deben obtener un permiso para poder utilizar los materiales protegidos – archivos, música, películas – de acuerdo con los derechos y las reglas de negocio definidas por el propietario del contenido.
administración del cambio (change management)	Es la práctica de administrar los cambios con la ayuda de métodos de prueba y técnicas para poder evitar nuevos errores y minimizar el impacto de los cambios.
alerta de virus Microsoft (Microsoft virus alert)	Es un anuncio que describe un virus específico, el impacto de ataques potenciales en el software de Microsoft y da sugerencias para prevenirlos o recobrarse de tales ataques.
alterar (tamper)	Modificar información maliciosamente.

anonimato (anonymity)	Condición en la que la verdadera identidad de un individuo es desconocida.
antivirus	Es el software diseñado específicamente para la detección y prevención de virus conocidos.
array (matrices)	Los arrays (o matrices) son un tipo de variable que permiten tener más de un elemento, (o valor en su interior), a los que se pueden acceder mediante un índice. Un array también es el tipo en el que se basan todas las matrices o arrays.
asistente de privacidad (Privacy Wizard)	Es una herramienta de software desarrollada por Microsoft que ayuda a los negocios a elaborar políticas de privacidad basadas en principios ampliamente aceptados.
ataque local (local attack)	Ataque dirigido a la PC en el cual el atacante ha iniciado una *sesión interactiva*.
ataque por servicio denegado - DoS (denial of service attack)	Es un asalto computarizado llevado a cabo por un atacante para sobrecargar o congelar un servicio de red, como un servidor Web o de archivos. Por ejemplo, un ataque puede causar que el servidor esté tan ocupado tratando de responder, que ignorara cualquier petición legítima de conexión.
ataque remoto (remote attack)	Es un ataque que tiene como objetivo una PC diferente en la que el atacante ha iniciado una *sesión interactiva*. Por ejemplo, un atacante puede iniciar una sesión en una estación de trabajo y atacar a un servidor en la misma red o en una diferente.
autenticación (authentication)	Es el proceso de verificar que alguien o algo es quien o lo que dice ser. En redes de equipos públicos y privados (incluyendo Internet), la autenticación se lleva a cabo comúnmente a través de contraseñas de inicio de sesión.
autorización (authorization)	Con referencia a la computación, especialmente en los equipos remotos en una red, es el derecho otorgado a un individuo o proceso para utilizar el sistema y la información almacenada en éste. Típicamente la autorización es definida por un administrador de sistemas y verificado por el equipo basado en alguna identificación del usuario, como son un código o una contraseña.
autorización basada la función (role based authorization)	Es un tipo de autorización que utiliza funciones para determinar los derechos de acceso y *privilegios*. Una función es una categoría simbólica de usuarios que comparten el mismo privilegio de seguridad.
aviso (notice)	Es un principio de privacidad que requiere la divulgación razonable a un consumidor de las prácticas de uso y recolección de *información personal identificable (PII)* de una entidad. Típicamente, esta divulgación de información es llevada a cabo a través de un aviso o de una política de privacidad. Obtenga más información sobre el aviso en las *Prácticas Honestas de Información*.

B	
BBBOnLine	Es un programa, establecido por el *Better Business Bureau*, que con su sello de privacidad certifica que sitios Web cumplen con los estándares de básicos de privacidad. Microsoft es un patrocinador de BBBOnLine.
boletín de seguridad Microsoft	Documento que describe un tema de seguridad específico en algún producto Microsoft y envía al lector a un archivo descargable que resuelve el problema.
bomba de correo (mail bomb)	Es una cantidad excesiva de información en correo electrónico enviada a la dirección de un usuario en un intento por hacer que el programa de correo electrónico del usuario colapse o para evitar que el usuario reciba mensajes legítimos.
búfer (buffer)	Es una región de la memoria reservada para servir como receptáculo intermediario en el cual la información es temporalmente retenida antes de su transferencia entre dos ubicaciones o dispositivos.

C	
caballo de troya (troyan horse)	Es un programa computacional que aparentemente es útil pero que en realidad causa daño.
calidad en el servicio (QoS)	Es un conjunto de estándares y mecanismos que aseguran la calidad en la transmisión de información.
capa de Sockets Seguros (SSL)	Es un protocolo para establecer un canal de comunicaciones encriptado que ayuda a prevenir la interceptación de información crítica, como números de tarjeta de crédito en World Wide Web y en otros servicios de Internet.
centro de descarga Microsoft	Es un sitio Web de Microsoft que proporciona actualizaciones de productos y revisiones de seguridad para los productos Microsoft.
certificado (certificate)	Es un archivo *encriptado* que contiene información de identificación del usuario o servidor, la cual es utilizada para verificar la identidad y ayudar a establecer un vínculo de seguridad mejorada.
cifrado (Chipre)	Es un método de *encriptación*, que utiliza típicamente una *clave* predefinida y un algoritmo para transformar *texto simple* en *texto cifrado*.

clases / cbjetos	Prácticamente todo lo que manejemos en el entorno .NET es una clase u objeto, de hecho todas las clases derivan de una clase u objeto básico: la clase System.Object
clases abstractas	Son clases que exponen un interface el cual hay que usar en las clases que se hereden de dicha clase abstracta.
clave (key)	En *encriptación* y *firmas digitales*, es un valor utilizado en combinación con un algoritmo para encriptar o decriptar información.
clave privada (private key)	Una de las dos *claves* en la *encriptación de clave pública*. El usuario mantiene la clave privada secreta y la utiliza para encriptar *firmas digitales* y para decriptar los mensajes recibidos.
clave pública (public key)	Una de las dos *claves* en la *encriptación de clave pública*. El usuario da a conocer esta clave al público y cualquiera puede utilizarla para encriptar mensajes que serán enviados al usuario y decriptar la firma digital del usuario. Compare con *clave privada*.
Código de autentificación de mensajes	Es un algoritmo que permite a un receptor asegurarse que un bloque de información ha conservado su integridad desde el momento en que se envió hasta el momento en que se recibió.
Common Language Runtime (CLR)	El CLR (Common Language Runtime) es el motor en tiempo de ejecución del .NET Framework, es decir la parte del "entorno" que se encarga de ejecutar el código de los lenguajes del .NET Framework.
condición de desincronización (race condition)	Es una condición causada por el tiempo de los eventos dentro o entre los componentes de un software. Típicamente, las condiciones de desincronización están asociadas con errores de sincronización que proporcionan una ventana de oportunidad durante la cual uno de los procesos puede interferir con otro, posiblemente introduciendo una vulnerabilidad de seguridad.
constante	Valores numéricos o de cadena que permanecen constantes, sin posibilidad de cambiar el valor que tienen. En caso de que necesitemos cambiar el valor, usaremos las variables.
contraseña (password)	Es una cadena de caracteres que el usuario escribe para verificar su identidad en una red o en una PC local.
colecciones	Serie de datos que están guardados en una lista, array (omatriz) o una colección propiamente dicha y que permite interactuar con los elementos de las mismas, pudiendo añadir, recuperar, eliminar uno o todos, saber cuantos elementos hay, etc.
cookie	Un pequeño archivo de texto que se almacena en la PC del usuario, el cual contiene información sobre el mismo que es relevante para un sitio Web, como sus

	preferencias.
credenciales (credentials)	Información que incluye la identificación y prueba de identificación que se utiliza para obtener acceso a los recursos locales y de red. Ejemplos de credenciales son el nombre de usuario y contraseñas, *tarjetas inteligentes* y *certificados*.
criptografía (cryptography)	Es la utilización de códigos para convertir información por medio de una *clave* para que sólo un receptor específico pueda leerla. La criptografía es utilizada para permitir la *autenticación* y el *no repudio*, y para ayudar a preservar la confidencialidad y la integridad de datos.
cumplimiento (enforcement)	Principio de privacidad que proporciona mecanismos para asegurar el cumplimiento de las *Prácticas Honestas de Información*, normas a las que pueden recurrir los individuos afectados por incumplimiento y que establece las consecuencias para las organizaciones incumplidas. Los métodos para el cumplimiento incluyen una revisión llevada a cabo por terceros independientes, como *BBBOnLine*.

D	
datos encriptados (encrypted data)	Es la información que ha sido convertida de texto simple a *texto cifrado*.
declaración de privacidad (privacy statement)	Documento que describe la posición de la compañía en cuanto a privacidad, detallando qué información reúne su sitio Web, con quién se comparte la información y cuántos usuarios pueden controlar el uso de su información personal.
decriptación (decryption)	Es el proceso de convertir los *datos encriptados* de regreso a su forma original.
descarga (download)	Es la transferencia de la copia de un archivo desde una PC remota a otra que lo pide por medio de un módem o por red.
Directiva de Protección de la Información de la Unión Europea (EU Data Protection Directive)	Ley de la Unión Europea que establece que la información personal proveniente de países de la UE puede ser transferida a otros países siempre y cuando proporcionen un nivel aceptable de protección a la privacidad. Una organización debe informar a los individuos el por qué la información sobre ellos es reunida, cómo contactar a la organización para preguntas y quejas, las terceras partes ante las cuales la organización divulgará la información, y las opciones que proporciona la organización para limitar la exposición de cierta información. Debe informarse apropiadamente y ofrecerse la opción de elección para permitir que los individuos seleccionen (*opt-in*) o de-seleccionen (*opt-out*) proporcionar información específica que la organización planea rastrear. Véase también el *Acuerdo de Puerto seguro*.
divulgación (disclosure)	Es un componente del principio de *aviso*, por el cual una compañía debe poner a disposición sus practicas de manejo de información, incluyendo avisos de cómo

	reúne, utiliza y comparte *información personal identificable* (PII).
divulgación de información (information disclosure)	La exposición de información a individuos que normalmente no tendrían acceso a ella.

E	
elección (choice)	Es la habilidad de un individuo para determinar cómo la *información personal identificable (PII)* reunida de él o ella puede ser utilizada, especialmente para propósitos más allá de aquellos para los cuales se proporcionó la información originalmente. Elección es un elemento de las *Prácticas Honestas de Información*.
elevación de privilegios (elevation of privileges)	Proceso mediante el cual el usuario engaña al sistema para que le otorgue derechos no autorizados, usualmente con el propósito de comprometer o destruir el sistema.
encapsulación	La posibilidad de ocultar el código usado para implementar un método o cualquier otro procedimiento o función de forma que lo único que interese sea el interfase expuesto por la clase u objeto.
encriptación (encryption)	Se refiere al proceso de convertir datos en *texto cifrado* para evitar que terceras personas lo puedan ver o acceder.
encriptación de clave pública (public key encryption)	Es un esquema asimétrico de *encriptación* que utiliza un par de claves: la *clave pública* encripta la información y la clave secreta correspondiente la decripta. Para *firmas digitales,* el proceso es inverso: el emisor utiliza la clave secreta para crear un numero electrónico único que puede ser leído por cualquiera que posea la clave publica correspondiente, la cual verifica que el mensaje es verdaderamente del emisor. Véase también *clave privada, clave pública*.
entidad principal de seguridad (security principal)	Alguien que tiene una cuenta con un *identificador de seguridad* asignado automáticamente para controlar el acceso a los recursos.
escalamiento (upgrade)	Es un paquete de software que reemplaza una versión instalada de un producto con una versión más nueva del mismo producto. Típicamente, el proceso de escalamiento deja intacta la información existente del cliente y sus preferencias mientras que reemplaza el software existente con una nueva versión.
escalar (upgrade)	Es cambiar a una nueva, usualmente más poderosa o sofisticada versión de un producto.
evento	Los eventos son procedimientos (SUB) que se ejecutan normalmente cuando el

	sistema Windows los provoca, por ejemplo, al hacer click en una ventana o en cualquier objeto de la ventana, cuando se cambia el tamaño de una ventana, cuando se escribe en una caja de textos, etc.
expresiones	Una expresión es una secuencia de operadores y operandos que describe un cálculo. Normalmente una expresión se evalúa en tiempo de ejecución. Existen expresiones numéricas y alfanuméricas o de caracteres.
expresiones lógicas	Las expresiones lógicas son expresiones pero cuyo resultado es un valor "lógico" (verdadero o falso).Este tipo de expresiones se usan normalmente con instrucciones que normalmente necesitan un valor verdadero(true) o falso (false)

F	
filtro (filter)	Patrón o máscara a través de la cual la información es pasada para separar elementos específicos. Por ejemplo, un filtro utilizado en correo electrónico o al recobrar mensajes de un grupo de noticias puede permitir a los usuarios el descartar automáticamente mensajes que vienen de usuarios específicos.
firewall	Una combinación de hardware y software que proporciona un sistema de seguridad, usualmente para ayudar a evitar el acceso de externos no autorizados a una red interna o Intranet.
firma digital (digital signatura)	Es la información que es incluida con un mensaje o es transmitida separadamente que se utiliza para identificar y autentificar al emisor y la información del mensaje. Una firma digital también puede confirmar que el mensaje no haya sido alterado.
formulario (ventana)	Un formulario es una ventana de Windows la cual usaremos para interactuar con el usuario, ya que en dicha ventana o formulario, estarán los controles y demás objetos gráficos que mostraremos al usuario de nuestra aplicación. Los formularios también son llamados "formas" o Forms en su nombre en inglés.
function (Función)	Los procedimientos FUNCTION son como las funciones delvb.NET, es decir, realizan una tarea, al igual que un Sub, pero siempre suelen devolver un valor, resultado del código que se ha ejecutado en su interior. A las funciones no se les puede asignar valores, a diferencia de las Propiedades.

G	
gusano (s) - worm	Es un programa que se mantiene y duplica solo, usualmente consume memoria, causando por tanto que el equipo deje de responder. Compare con *virus*.

H	

handles	En VB.NET se usa Handles, seguido del nombre del evento, para indicar qué evento es el que se maneja en el procedimiento indicado. El formato suele ser: Sub Nombre(parámetros) Handles Objeto.Evento
herencia	La posibilidad de que una clase herede las propiedades y métodos de otra clase de forma que se puedan usar con la nueva clase de igual forma que si se hubiesen escrito directamente en ella.
HFNETCHK	Herramienta de línea de comandos que permite al administrador verificar el estado de una revisión en una PC con Microsoft® Windows NT 4.0, Windows 2000 y Windows XP en una red desde una ubicación central.
hotfix	Es un paquete sencillo acumulativo compuesto por uno o más archivos utilizados para corregir un defecto en el producto. También conocido como *QFE*, *revisión* y *actualización*.

I	
identificador de seguridad SID	Valor que identifica únicamente a cada usuario, grupo, cuenta e inicio de sesión en una red.
identificador único global GUID	Es un valor de 16 bits, generado a partir del identificador único en un dispositivo, de la fecha y hora actual, y de una secuencia de números. El GUID se utiliza para identificar un dispositivo, componente, usuario o sesión en particular.
imitación (spoofing)	Es la práctica de hacer que una transmisión aparezca como venida de un usuario diferente al usuario que realizó la acción.
información de clickstream (clickstream data)	Es la información que los usuarios generan mientras se mueven de página en página y hacen clic en objetos dentro de un sitio Web, usualmente se almacena en archivos de registro. Los diseñadores de sitios Web pueden utilizar la información de clickstream para mejorar las experiencias de los usuarios con un sitio.
información personal identificable PII	Cualquier información relativa a un individuo identificado o identificable. Tal información puede incluir nombre, país, dirección física, dirección de correo electrónico, número de tarjeta de crédito, número de Seguro Social, número de identificación gubernamental. También se le conoce como información personal o datos personales.
información confidencial (sensitive data)	Desde la perspectiva de la Unión Europea, es la, *información personal identificable (PII)* que se refiera a raza u origen étnico, opiniones políticas, creencias religiosas o filosóficas, preferencias sexuales o membresía a sindicatos de comercio. Dentro de los Estados Unidos, la información confidencial también incluye información sobre

	salud, finanzas e hijos.
infraestructura de clave pública PKI	Generalmente se refiere a las leyes, políticas, estándares y software que regulan o manejan los *certificados* y claves públicas y privadas.
Ingeniería de corrección rápida QFE	Es un equipo dentro de Microsoft que produce *hotfixes*. La mayoría de estos equipos se refieren a sí mismos como equipos de Ingeniería Sostenida. QFE también se utiliza como un sinónimo para hotfix o para especificar que un hotfix no será distribuido a través de un sitio Web público.
inicio de sesión en red (network logon)	Es el proceso de iniciar una sesión en una PC por medio de una red. Típicamente, un usuario inicia una sesión interactivamente primero en un equipo local, después proporciona sus credenciales de inicio de sesión en otro equipo en la red, como un servidor, que el usuario está autorizado para utilizar. Compare con *inicio de sesión interactiva*.
inicio de sesión interactiva (interactive logon)	Es el proceso de iniciar una sesión en una PC local utilizando un teclado. Compare con *inicio de sesión en red*.
instancia	Para poder usar una clase u objeto, hay que crear una instancia del mismo. Es decir, debe declararse una variable y a esa variable asignar el objeto o clase en cuestión para que pueda usarse. Es como si se tuviera que darle vida al objeto par poder usarlo.
interface	Se dice que las propiedades y métodos expuestos por una claseforman el interface de la misma.
interface / implements	Los interfaces a diferencia de las clases es que no hay queescribir código para los métodos o propiedades que expone, simplemente se indica la "declaración". Usando Implements, se pueden usar esas interfaces en lasclases, aunque hay que escribir el código de cada método opropiedad implementado.

L	
llamada a procedimiento remoto RPC	Es un mecanismo de comunicación que permite a una aplicación cliente y servidor el comunicarse entre ellos mismos a través de llamadas de funciones enviadas desde el cliente hacia el servidor.

M	
Me (this)	La palabra clave (o instrucción) Me hace referencia a la clase actual. Por ejemplo Me.Width se refiere a la propiedad Width de la clase actual. En C# en lugar de Me es this.
método	Un procedimiento (Sub, Function -función) que se usa pararealizar una tarea específica en la clase o módulo.
Microsoft .NET Passport	Es un servicio de autenticación en línea de Microsoft utilizado en Internet el cual permite que la autenticación de identidad de un individuo sea compartida entre socios participantes. La clave de acceso Microsoft .NET permite al usuario utilizar una sola credencial de identidad, como nombre y clave de acceso. Después de conceder permiso al socio la primera vez, la clave de acceso .NET no requiere que el usuario inicie la sesión una y otra vez.
módulo	Los módulos, al igual que las clases, son "espacios" en los cuales se incluyen declaraciones de variables, procedimientos, funciones, etc. Pero a diferencia de las clases, el código contenido en un módulo siempre está disponible de forma directa, sin necesidad de crear una "instancia" de dicho módulo.
MyBase	La palabra clave MyBase se comporta como la clase de la que ha derivado la clase actual, es decir si una clase deriva de una(o hereda a otra) clase, MyBase se referirá a dicha clase base, de esta forma es posible acceder a los métodos, propiedades y eventos de la clase de la que se deriva (o hereda) la clase actual.

N	
no repudio (nonrepudiation)	Es la habilidad de identificar quien ha llevado a cabo varias acciones en una PC, para que los usuarios no puedan negar las responsabilidades de las acciones que ellos llevan a cabo. Generalmente utilizado en el sentido de crear una huella de auditoría indiscutible para identificar la fuente de una transacción comercial o acciones maliciosas.
número de identificación personal - PIN	Es un número de código de acceso único asignado, como en las tarjetas de cajero automático, al usuario autorizado.

O	
opt in	Es la aceptación explícita a participar. Típicamente, se utiliza en los programas de mercadeo y ofertas, en donde una acción (como el uso de información personal mas allá del propósito original y primario para el cual fue obtenida) no se lleva a cabo a menos de que un individuo exprese su consentimiento. Es un elemento de *elección*.

	Véase también *Prácticas Honestas de Información*.
opt out	Es declinar explícitamente la participación. Típicamente, se utiliza en programas de mercadeo y ofertas, en donde una acción (como el uso de información personal mas allá del propósito original y primario para el cual fue obtenida) se lleva a cabo a menos de que un individuo decline explícitamente. Es un elemento de *elección*. Véase también *Prácticas Honestas de Información*.

P	
parámetro	Los métodos o propiedades pueden tener parámetros, (uno ovarios), los cuales le indicarán los valores que deben usar para la tarea que debe realizar. Por ejemplo, un método Contar podría recibir un parámetro con el valor de las veces que tiene que contar.
parámetros opcionales	Algunos procedimientos que aceptan parámetros, pueden tener también parámetros opcionales, los cuales, como su nombre indica, pueden o no ser incluidos en la llamada al procedimiento. Los parámetros opcionales tienen unos valores por defecto, el cual se usará en caso de que no se especifique.
perfil de usuario (user profile)	Configuraciones que definen las preferencias de personalización de un usuario en particular, tales como las configuraciones de escritorio, conexiones de red persistentes, *información personal identificable (PII)*, utilización de un sitio Web y otros comportamientos y datos demográficos.
permisos (permissions)	Regla asociada con un objeto, como un archivo, para regular qué usuarios pueden obtener acceso al objeto y de qué manera. El dueño del objeto otorga o niega los permisos.
Plataforma para el Proyecto de Preferencias de Privacidad - P3P	Una especificación de privacidad abierta, desarrollada y administrada por el Consorcio World Wide Web (WWW) que cuando se implementa, permite a las personas tomar decisiones informadas sobre cómo desean compartir la información personal con los sitios Web.
polimorfismo	La posibilidad de usar en clases diferentes propiedades o métodos con el mismo nombre de forma que cuando se usen nonos preocupe a que clase pertenece. Por ejemplo el objeto básico del que derivan todas las clases de.NET tiene una propiedad llamada *ToString*, ésta propiedad estará implementada de forma diferente en diferentes clases, pero se usa de la misma forma, sin importar que objeto se este usando.
política de privacidad (privacy policy)	Son los requerimientos de una organización para cumplir con las regulaciones y directivas de privacidad.

prácticas de mercadeo dudosas (deceptive trade practices)	Son productos o servicios engañosos o distorsionados para los consumidores y clientes. En los Estados Unidos, estas prácticas están reguladas por la Comisión Federal de Comercialización a nivel federal y típicamente a nivel estatal por la Oficina del Fiscal General de Protección al Consumidor.
Prácticas Honestas de Información (Fair Information Practices)	Son las bases para prácticas recomendadas de privacidad, tanto en línea como sin conexión. Son las prácticas originadas en la Ley de Privacidad de 1974, la legislación que protege la información personal reunida y mantenida por el gobierno de los Estados Unidos. En 1980, estos principios fueron adoptados por la Organización de Cooperación y Desarrollo Económico e incorporados en sus Lineamientos para la Protección de la Información Personal y el Transporte de los Flujos de Información. Fueron adoptados más tarde en la Directiva de Protección a la Información de la Unión Europea en 1995, con modificaciones. Las Prácticas Honestas de Información incluyen *aviso, elección, acceso, retransferencia, seguridad,* integridad de datos y remedios.
principios de Puerto seguro (Safe Harbor Principles)	Son los siete principios acordados entre los EE.UU. y la UE para la transferencia de *información personal identificable (PII)* desde la UE hacia los EE.UU., a los cuales una compañía debe adherirse si se registra para Puerto Seguro. Los siete principios están categorizados en los siguientes temas: *aviso, elección, acceso, retransferencia, seguridad,* integridad de datos y *cumplimiento*..
privacidad (privacy)	El control que los clientes tienen sobre la recolección, uso y distribución de su información personal.
privacidad comprometida (privacy compromise)	Escenario en el cual un individuo no autorizado puede obtener acceso a información personal o confidencial sobre otro usuario.
privilegios (privileges)	Es el permiso otorgado a un usuario para llevar a cabo una tarea específica, usualmente una que afecta a todo un sistema computacional en lugar de a un objeto en particular. Los privilegios son asignados por un administrador, a usuarios individuales o a grupos de usuarios, como parte de las configuraciones de seguridad de una PC.
procedimiento	Un método, función o propiedad de una clase o módulo.
programación orientada a objetos (OOP / PPO**)**	Una forma de programar basada en la reutilización de código mediante herencia, encapsulación y polimorfismo.
Property (Propiedad)	A diferencia de los métodos, las propiedades se usan para "configurar" la forma que tendrá la clase. Algunas veces es difícil diferenciar un método de una propiedad, pero por convención los métodos realizan tareas. Por ejemplo, el ancho de un objeto es una propiedad, mientras que mostrar el objeto se realizaría con un método. A las Propiedades se les puede asignar valores y pueden devolverlos,

	(como las funciones). Aunque también pueden existir propiedades de solo lectura, (solamente devuelven valores), o de solo escritura, (sólo se les puede asignar valores, pero no los devuelven).

R	
rechazo (repudiation)	La habilidad de un usuario para negar haber llevado a cabo una acción que otras partes no puedan refutar. Por ejemplo, un usuario que borre un archivo puede con éxito negar haberlo hecho si no existe ningún mecanismo (como archivos de auditoria) que pueda contradecir su declaración.
recinto (sandbox)	Es un mecanismo de protección utilizado en algunos ambientes de programación que limita el acceso que tiene un programa a los recursos del sistema. Un recinto restringe un programa a una serie de *privilegios* y comandos que le dificultan o imposibilitan el causar algún daño a la información del usuario.
Red Privada Virtual - VPN	Red de información privada que hace uso de una red pública, como Internet, al *encriptar* información en un nodo y utilizar procedimientos de seguridad que proporcionan un túnel a través del cual la información puede pasar a otro nodo.
reemplazo total de archivo (full file replacement)	Es una tecnología utilizada en *hotfixes* que reemplaza los archivos instalados actualmente con nuevos archivos.
Regularización (throttling)	Es un método para prevenir el *ataque por servicio negado* al limitar el número de peticiones que pueden hacerse a un sistema. También se le conoce como *agrupación* (pooling).
retransferencia (onward transfer)	Es la transferencia de *información personal identificable (PII)* por parte del receptor de la información original a un segundo receptor. Por ejemplo, la transferencia de PII desde una entidad en Alemania a una entidad en los Estados Unidos constituye una retransferencia de esa información. La retransferencia es discutida en las *Prácticas Honestas de Información*.
retransmisión de correo (mail relaying)	Práctica en la cual un atacante envía mensajes de correo electrónico desde otro sistema de servidor para utilizar sus recursos y/o para hacer aparecer que los mensajes se originaron desde otro sistema.
revisar (patching)	Método para actualizar un archivo que reemplaza solamente las partes que han sido cambiadas, en lugar del archivo completo.
revisión (patch)	Es un sinónimo de *hotfix* que es utilizado genéricamente en la industria del software. En algunos casos el término se ha utilizado para indicar un método utilizado para aplicar el *hotfix*, donde una revisión no puede reemplazar todo el archivo pero en

	lugar de eso modificará el archivo existente.
revisión privada (private fix)	Es un *hotfix* no oficial que puede no estar completamente probado o empaquetado. Es entregado al cliente antes de la prueba final para verificar que resuelve el problema.

S	
salvaguarda (safeguard)	Es una tecnología, política, o procedimiento que contrarresta una amenaza o protege un valor.
saturación del búfer (búfer overrun)	Condición que resulta de agregar más información a un búfer de la que fue diseñado para almacenar. Un atacante puede explotar esta vulnerabilidad para apoderarse de un sistema.
seguridad computacional (computer security)	Es la disciplina, técnicas y herramientas diseñadas para ayudar a proteger la confidencialidad, integridad y disponibilidad de información y sistemas.
Seguridad de la capa de transporte TLS	Es un protocolo que ayuda a proporcionar a las comunicaciones, privacidad y seguridad entre dos aplicaciones que se comunican a través de una red. También permite que los clientes *autentifiquen* servidores y que los servidores autentifiquen a los clientes.
Service Pack	Es un conjunto acumulado de todos los *hotfixes* creados y las correcciones para errores encontrados internamente desde la publicación del producto. Los Service Packs pueden contener también un número limitado de peticiones del cliente para cambios de diseño o características. Éstos son ampliamente distribuidos y por tanto probados arduamente.
Service Release	Es un conjunto de *hotfixes* que pueden contener nuevas características. Típicamente, una publicación de servicio se integra para crear un producto completamente empaquetado el cual reemplaza un producto de venta en el mercado. El Service Release de un OEM (fabricante de equipo original) es distribuido solamente a un OEM, usualmente para el soporte de nuevo hardware.
servidor proxy (proxy server)	Componente de un *firewall* que maneja el tráfico de Internet hacia y desde una red de área local (LAN) y puede desempeñar otras funciones, como el almacenaje de un documento y control de acceso.
sistema de encriptación de archivos EFS	Es una tecnología de *encriptación* basada en archivos que permite al usuario encriptar archivos y carpetas en un disco de volumen NTFS para mantener su confidencialidad.

sobrecarga (Overload)	Se dice que un método está sobrecargado cuando existen distintas versiones de dicho método en la clase. Por ejemplo métodos con el mismo nombre que reciban parámetros de distintos tipos.
Sub	Un procedimiento SUB es como una instrucción, es decir, realiza una tarea (ejecuta el código que haya en su interior),pero no devuelve un resultado.
suplantación (impersonation)	Es la habilidad de un proceso para ejecutarse en un contexto de seguridad específica – como un usuario específico, por ejemplo – y por tanto acceder recursos autorizados para ese contexto de seguridad. La suplantación es utilizada en las aplicaciones Web para proporcionar un contexto de seguridad mejorada para peticiones anónimas.

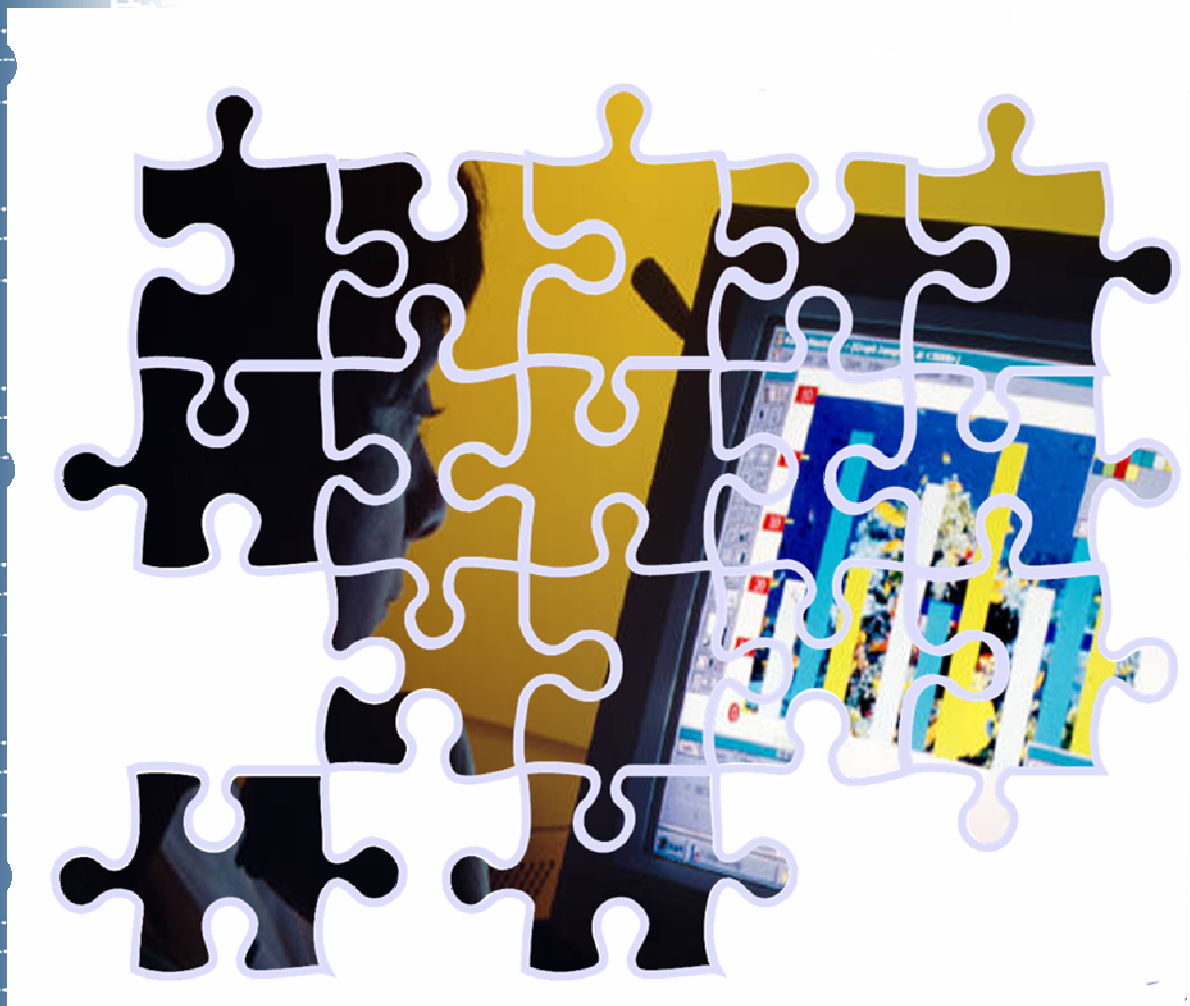
T	
tarjeta inteligente (smart card)	Dispositivo del tamaño de una tarjeta de crédito con un microprocesador integrado que se utiliza con un código de acceso para permitir una *autenticación* basada en *certificados*.
texto cifrado (ciphertext)	Es el texto que ha sido encriptado con un algoritmo y clave de *encriptación*.
texto simple o no cifrado (plaintext)	Un mensaje que no está *encriptado*. Los mensajes de texto simples también son conocidos como mensajes de texto no cifrado.
token de acceso (access token)	Es una estructura de datos que contiene información de autorización para un usuario o un grupo. Un sistema utiliza un token de acceso para controlar el acceso a objetos seguros y para controlar la habilidad de un usuario para llevar a cabo varias operaciones relativas al sistema en una terminal.
transferencia de datos (data transfer)	Como un principio clave de la privacidad, es el movimiento de *información personal identificable (PII)* entre entidades, tal como una lista de clientes que se comparte entre dos compañías diferentes.
transparencia (transparency)	Estándar que requiere que la estructura para procesar la información personal sea abierta y entendible por el individuo, cuyos datos están siendo procesados. Es ino de los objetivos de las *Prácticas Honestas de Información*, que requiere que la compañía informe a los usuarios qué información personal reúne la compañía y cómo la utiliza.
TRUSTe	Organización que ofrece programas de sello privado en línea que certifican a los sitios Web elegibles y sujetos a los sitios Web a un estándar de privacidad básico.

	Esta organización guardiana de las claves de privacidad juega un papel de cumplimiento importante en la disputa y resolución de temas de seguridad.

U	
usos secundarios de la información (secondary data uses)	Es el uso de la información personal para propósitos diferentes de aquellos para los que la información fue proporcionada. Las *Prácticas Honestas de Información* establecen que una persona puede proporcionar información personal para un propósito específico sin temor de que pueda ser utilizada después para algún propósito no relacionado su conocimiento o consentimiento.
usuario malicioso (malicious user)	Persona que tiene acceso a un sistema y representa una amenaza para éste. Un ejemplo es alguien que trata de *elevar sus privilegios* para obtener acceso a información no autorizada.

V	
variable	Son "espacios" de memoria en la que se almacena un valor. Se usarán para guardar en memoria los valores numéricos o de cadena de caracteres que nuestro programa necesite.
virus	Programa que trata de esparcirse de una PC a otra, usualmente a través de correo electrónico, adjuntándose a si mismo a un programa huésped. Puede dañar el hardware, software o datos.

Prácticas



PRÁCTICA #1: USO DEL DESENSAMBLADOR MSIL

OBJETIVO

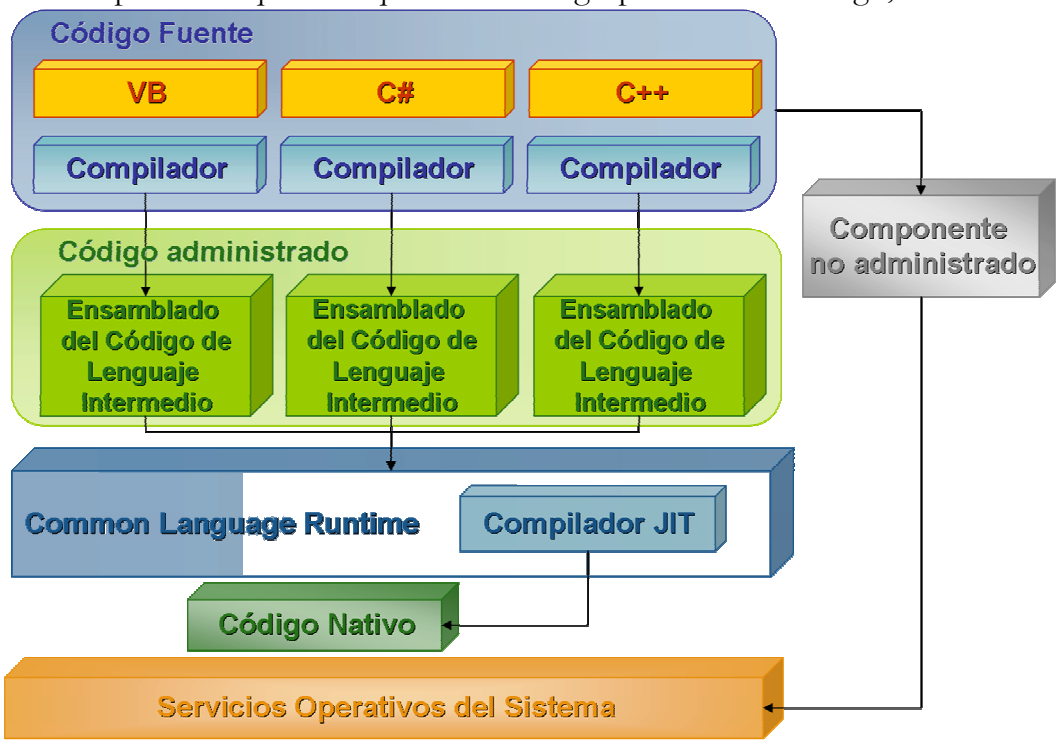
Después de completar esta práctica, podrá utilizar el Desensamblador MSIL para ver el código MSIL.

INTRODUCCIÓN

La unidad base de una aplicación basada en .NET se conoce como **ensamble** o ***assembly***. Un ensamble es la unidad de versión e implementación. Por lo general cuando de compila el código fuente en un EXE o en un DLL (también puede crearse módulos, que más adelante se pueden vincular en un EXE o en un DLL). Éstas son las formas más sencillas de un ensamble.

Los compiladores generan ensambles que contienen *Microsoft Intermediate Language*. Sin embargo, aparte del código MSIL, el compilador inserta **metadatos** en el ensamble. Los **metadatos** son una colección de información que describe todos los tipos, clases, métodos, campos, eventos y demás contenidos en el ensamble. Esto es de alguna manera similar a la idea de una biblioteca de tipos. Sin embargo, a diferencia de un servidor COM que puede o no tener un recurso de biblioteca de tipos incrustada, un ensamble y sus metadatos son inseparables. Esto, por supuesto, significa que el ensamble es autodescriptivo.

En muchos casos, un ensamble podría considerarse como un solo EXE o DLL. En algunas situaciones, un solo ensamble hace que la implementación sea mucho más sencilla, ya que todos los componentes que se requieren están agrupados. Sin embargo, un número de ensambles (por ejemplo, DLLs) se pueden vincular entre sí en un ensamble (un ensamble de archivos múltiples). Para lograr esto puede utilizarse la herramienta Vinculador de ensambles, AL.EXE. En algunas situaciones, por ejemplo en las aplicaciones basadas en el Web, el hecho de



que los ensambles estén contenidos dentro de archivos separados puede ser una gran ventaja, ya que sólo se pueden descargar los módulos que se requieren.

Antes de poder ejecutar el código MSIL, se debe convertir a instrucciones binarias nativas. La compilación normalmente la realiza por un compilador JIT. El siguiente diagrama ilustra el proceso que se utiliza para compilar y ejecutar el código administrado, esto es, el código que utiliza el CLR.

El código fuente escrito en Visual Basic .NET, C# o en algún otro lenguaje que se dirija al CLR, primero se transforma en MSIL a través del compilador de lenguaje apropiado. Antes de ejecutarse, este código MSIL se compila en código nativo en un compilador JIT para cualquier procesador en el que se ejecute el código. El compilador JIT no compila todo el código en una sola vez. La función predeterminada es compilar en JIT cada método cuando se invoca por primera vez, pero también es posible compilar "previamente en JIT" el código IL utilizando el Generador de imagen nativa (NGEN.EXE). Con esta opción, todos los métodos se compilan antes de cargar la aplicación, por lo que se elimina el costo de la compilación JIT en cada invocación de método inicial.

Obsérvese que todos los lenguajes que se dirigen al CLR deben exhibir prácticamente el mismo rendimiento. Mientras que algunos compiladores pueden producir mejor código MSIL que otros, es poco probable que surjan grandes variaciones en la velocidad de ejecución.

ESCENARIO

En esta práctica, se verá el archivo ejecutable portátil (PE) creado para un proyecto Visual Basic .NET. Para este fin se utilizará el Desensamblador MSIL.

El Desensamblador MSIL es una herramienta compañera para cualquier compilador compatible con .NET, incluyendo C# y los compiladores de Visual Basic, así como el Ensamblador MSIL. El Desensamblador MSIL toma un archivo PE que contiene código de lenguaje intermedio y lo muestra en una aplicación Windows, o utilizando las opciones de línea de comando, crea un archivo de texto.

DESARROLLO

Para ver el código MSIL en el Desensamblador MSIL

1. Haga clic en **Inicio**, apunte a **Programas**, apunte a **Microsoft Visual Studio.NET 2003**, apunte a **Visual Studio.NET Tools**, y luego haga clic en **Visual Studio.NET Command Prompt**.
2. De ser necesario, cambie a la unidad que contenga los archivos de práctica.
3. En el Indicador de comando, escriba **cd** <*letra de la unidad*>**:\Practica1-MSIL\ExeParaElMSIL** y presione **ENTER**.
4. En el Indicador de comando, escriba **ildasm Excepciones.exe** y presione **ENTER**.

Se abrirá una ventana ILDASM nueva con una estructura de árbol.

5. Amplíe el nodo **EjemplosExcepciones**.
6. Amplíe el nodo **frmExcepciones**.

Aparecen los Manifiestos, Campos, Métodos y Propiedades en la clase. Obsérvese que los triángulos rojos que señalan hacia la derecha indican los manifiestos, el diamante azul indica un campo, los rectángulos magenta indican los métodos y los rectángulos magenta marcados con "S" indican los métodos estáticos. Puede hacer doble clic en cualquier nodo para obtener más información acerca de ese elemento.

7. Haga doble clic en **MANIFEST**.

Aparece una ventana que muestra los contenidos del proyecto y sus referencias.

8. Realice sus conclusiones contemplando la utilidad que se puede dar a esta herramienta.

PRÁCTICA #2: MANEJO DE EXCEPCIONES

OBJETIVO

Después de terminar esta práctica, podrá manejar excepciones utilizando la instrucción **Try...Catch...Finally**.

INTRODUCCIÓN

Implementar el manejo completo de errores en Visual Basic siempre ha sido un reto. Un esquema consistente de manejo de errores significaba una gran cantidad de código duplicado. El manejo de errores utilizando la instrucción **On Error GoTo** en ocasiones alentaba el desarrollo y mantenimiento de aplicaciones grandes. Manejar varios errores con varias combinaciones de **Resume** y **Next** resultaba rápidamente en código ilegible, y, cuando no se planeaban completamente las rutas de ejecución, frecuentemente generaba errores. A continuación se presentan un ejemplo de código que utilizaba la instrucción **On Error GoTo**:

```
Variable = False
On Erro GoTo ErrManipulado
<Código que puede fallar>
GoTo Etiqueta
ErrManipulado:
If CondiciónPuedeManipularse Then
<código de error manipulado>
else
Variable2 = True
End if
Etiqueta:
<código de la etiqueta>
If Variable2 then err.Raiser err
```

Visual Basic soporta ahora el manejo estructurado de excepciones, utilizando una versión mejorada de la sintaxis **Try...Catch...Finally** que soportan otros lenguajes como C++. El manejo estructurado de excepciones combina una estructura de control moderna, similar a las instrucciones **Select Case** o **While** de Visual Basic, que incluye excepciones, bloques protegidos de código y filtros. El manejo estructurado de excepciones facilita crear y mantener programas con manejadores de errores robustos y completos.

```
Try
     <código que puede fallar>
Catch CondiciónManipulable
     <código de error manipulado>
Finally
     <código que siempre se ejecuta>
End Try
```

La instrucción **Try** administra el código entre la instrucción **Try** y el primer **Catch**. La instrucción **Catch** filtra errores, los cuales se derivan normalmente de una **System.Exception**

común. La instrucción **Finally** siempre se ejecuta y puede ser utilizada para limpieza. Visual Basic .NET aún soporta la instrucción **On Error GoTo**. Sin embargo, el uso de esta instrucción generará una ejecución significativamente más lenta.

ESCENARIO

En este ejercicio aprenderá a escribir código para manejar errores de tiempo de ejecución. Verá tres formas de codificar las instrucciones **Try...Catch** para manejar excepciones.

DESARROLLO

Para abrir el proyecto

En el Explorador de Windows, navegue a <*Letra de la unidad*>:\Practicas\Practica2-Excepciones\TryCatch y haga doble clic en **EjemploExcepciones.sln** para abrir el proyecto.

Para entender el código

1. En el **Explorador de soluciones**, amplíe el proyecto **Excepciones** si se requiere, y después haga doble clic en **frmExcepciones.vb**.

   Para ver tres ejemplos del manejo de excepciones en la aplicación que se está ejecutando, haga clic en los botones **Ejemplo 1**, **Ejemplo 2** y **Ejemplo 3**. Ahora veamos el código escrito para cada uno de estos ejemplos.

2. En el menú **Ver**, haga clic en **Código** y después desplácese hacia abajo para ver el código para el evento **Click** del botón **Ejemplo 1**. Este botón se llama **btnEjemplo1**.

   En este ejemplo, todo el evento **Click** está dentro de la instrucción **Try...Catch**. Observe que el código para la instrucción **z = CInt(y / x)** causará una excepción ya que requiere operar 10/0. El control entonces se moverá hacia la instrucción **Catch** y después presentará el mensaje **Excepción Capturada**. Como resultado, la instrucción **MessageBox.Show("Fin del Bloque Try")** nunca se ejecuta.

3. Desplácese hacia abajo un poco más para ver el código para el evento **Click** del botón **Ejemplo 2**. Este botón se llama **btnEjemplo2**.

   En este ejemplo, de nuevo, todo el evento **Click** está dentro de la instrucción **Try...Catch**. En la instrucción **Try**, observe que una excepción del tipo **ArgumentException** se lanza manualmente utilizando la instrucción **Throw New ArgumentException()**. Como en el ejemplo anterior, en cuanto se lanza la excepción, el control se mueve a la instrucción **Catch** y presenta el mensaje **Excepción OverFlow Capturada**. Como resultado, la instrucción **MessageBox.Show("Fin del bloque Try")** nunca se ejecuta.

   Observe que una instrucción **Finally** se ha codificado a continuación. El código en la instrucción **Finally** siempre se ejecutará. La instrucción **Finally** normalmente se utiliza para llevar a cabo tareas de limpieza que se deben realizar cuando se lanza una

excepción. Las tareas de limpieza incluyen cerrar un archivo o socket cuando ya no se requieren.

4. Desplácese hacia abajo para ver el código para el evento **Click** del botón **Ejemplo 3**. Este botón se llama **btnEjemplo3**.

   La instrucción **Try** puede tener más de una instrucción **Catch** asociada con ella. Una instrucción **Catch** puede ser genérica o específica. Una instrucción **Catch** genérica deberá capturar cualquier error que ocurra en la instrucción. Una instrucción **Catch** específica deberá capturar un tipo específico de error.

   En este ejemplo, observe que la instrucción **Catch** maneja únicamente **OverflowException**. La instrucción **z = CInt(y / x)** causará que se lance una excepción **OverflowException** que será capturada por la instrucción **Catch excOverflow As OverflowException**. Todos los demás tipos de excepciones serán capturadas por una instrucción **Catch** genérica **Catch exc As Exception**.

Para compilar y ejecutar el proyecto

1. En el menú **Depurar** haga clic en **Inicio**.
2. Haga clic en el botón **Ejemplo 1**.

   Observe que aparece **Excepción Capturada**. Haga clic en **Aceptar** para cerrar el cuadro de mensajes.

3. Haga clic en el botón **Ejemplo 2**.

   Observe que aparece **Excepción Overflow Capturada** en la consola. Haga clic en **Aceptar** para cerrar el cuadro de mensajes. Aparece otro cuadro de mensajes que muestra **El código Finally siempre se ejecuta**. Haga clic en **Aceptar** para cerrar el cuadro de mensajes.

4. Haga clic en el botón **Ejemplo 3**.

   Observe que aparece **Overflow Exception Caught** en la consola. Haga clic en **Aceptar** para cerrar el cuadro de mensajes. Aparece otro cuadro de mensajes que muestra **El código Finally siempre se ejecuta**. Haga clic en **Aceptar** para cerrar el cuadro de mensajes.

5. Realice sus propias conclusiones contemplando las aplicaciones en los sistemas que tiene el control de excepciones y errores.

PRÁCTICA #3: HEREDAR UNA FORMA

OBJETIVO

Una vez terminada esta práctica, puede crearse de una forma y heredarla dentro de otra aplicación.

INTRODUCCIÓN

Al igual que con todos los objetos en .NET Framework, las formas son instancias de clases. .NET Framework también le permite heredar desde las formas existentes para agregar funcionalidad o modificar un comportamiento existente. Cuando agrega una forma puede elegir si la hereda de una clase **Form** desde una forma que creó anteriormente (la cual, a su vez, puede haber sido derivada de **Form**).

La herencia visual utilizando Windows Forms puede mejorar la productividad del desarrollador y facilitar la reutilización del código. Por ejemplo, una organización puede definir una forma base estándar que contiene elementos tales como el logotipo corporativo y quizás una barra de herramientas común. Esta forma la pueden utilizar los desarrolladores a través del proceso de herencia y después ampliarla para satisfacer los requerimientos de aplicaciones especificas al tiempo que promueven una interfaz común a lo largo de toda la organización. El creador de la forma base puede especificar qué elementos se pueden ampliar y cuáles deberán utilizarse como están, asegurando que se reutilicen apropiadamente.

Crear nuevas Windows Forms al heredar desde las formas base es una forma útil de duplicar sus mejores esfuerzos sin tener que recrear completamente una forma cada vez que la necesita.

La referencia deberá incluir el espacio de nombre que contiene la forma, seguida por un punto, después el nombre de la forma base misma (que se maneja automáticamente en Visual Studio):

```
[Visual Basic]
Imports Namespace1
Public Class Form2 Inherits Namespace1.Form1
```

ESCENARIO

En este ejercicio creará una aplicación con una forma **WinBaseForm**. Después creará otra aplicación que herede la forma **WinBaseForm**.

DESARROLLO

Para crear la forma base

1. Abra **Microsoft Visual Studio.NET**.

2. Haga clic en **Archivo**, apunte a **Nuevo** y después haga clic en **Proyecto**.
3. En el cuadro de diálogo **Nuevo proyecto**, seleccione **Proyectos de Visual Basic** en el panel **Tipos de proyecto**.
4. Seleccione **Biblioteca de clase** en el panel **Plantillas**.
5. Escriba **WinBaseForm** en el cuadro **Nombre**, y haga clic en **Aceptar**.
6. En el **Explorador de soluciones**, haga clic en el botón alterno sobre **Class1**, y después haga clic en **Eliminar**.
7. Haga clic en **Aceptar** para eliminar la clase permanentemente.
8. Haga clic en el botón alterno sobre **WinBaseForm**, haga clic en **Agregar**, y después haga clic en **Agregar Windows Forms**.

   Aparecerá el cuadro **Agregar nuevo elemento**, con la plantilla **Windows Form** seleccionada.
9. Escriba **WinBaseForm.vb** en el cuadro **Nombre** y haga clic en **Abrir**.
10. De la pestaña **Windows Forms** en la Caja de herramientas, arrastre y suelte el control de **Botón** sobre la forma.
11. Repita el Paso 10 para agregar otro botón a la forma.
12. Haga clic en el botón alterno sobre **Button1** y haga clic en **Propiedades**.
13. Cambie el valor del atributo **Texto** a **Aceptar**.
14. Cambie el valor del atributo **Nombre** a **Aceptar**.
15. Haga clic en el botón alterno sobre **Button2** y haga clic en **Propiedades**.
16. Cambie el valor del atributo **Texto** a **Cancelar**.
17. Cambie el valor del atributo **Nombre** a **Cancelar**.
18. Haga doble clic sobre el botón **Aceptar** y después agregue la siguiente línea de código al subprocedimiento **OK_Click**:

```
        me.Close
```

19. Haga clic en el botón alterno sobre **WinBaseForm** en **Explorador de soluciones**, y haga clic en **Construir** para construir un proyecto.

Para crear una forma que herede WinBaseForm

1. En Visual Studio .NET, haga clic en **Archivo**, apunte a **Agregar proyecto**, y después haga clic en **Nuevo proyecto**.
2. En el cuadro de diálogo **Nuevo proyecto**, seleccione **Proyectos de Visual Basic** en el panel **Tipos de proyecto**.
3. Seleccione **Biblioteca de clase** en el panel **Plantillas**.
4. Escriba **MainForm** en el cuadro **Nombre**, y haga clic en **Aceptar**.

5. Haga clic en el botón alterno sobre **Class1** en la ventana del **Explorador de soluciones**, y después haga clic en **Propiedades**.
6. Cambie el Nombre de archivo a **MainForm.vb**.
7. Para agregar una referencia, haga clic en el botón alterno sobre **Referencias** en el **Explorador de soluciones** y haga clic en **Agregar referencias**.
8. Haga clic en la pestaña **Proyectos**.
9. Haga doble clic en **WinBaseForm** y después haga clic en **Aceptar**.
10. En el **Explorador de soluciones**, haga clic en el botón derecho sobre **MainForm.vb**, y después haga clic en **Ver código**.
11. En la ventana de código, cambie la definición de clase por la clase **MainForm** para heredar la clase **WinBaseForm**, como aparece en el siguiente ejemplo de código:

```
Public Class MainForm
    Inherits WinBaseForm.WinBaseForm
```

12. Haga doble clic en **MainForm.vb** para abrir la forma.

    La forma ahora muestra los botones Aceptar y Cancelar que se heredaron de la forma **WinBaseForm**.

13. Por ultimo realice sus conclusiones contemplado las aplicaciones en le mundo real de la programación.

PRÁCTICA #4: ATAQUE DE DENEGACIÓN DE SERVICIO (DOS)

OBJETIVO

Una vez concluida esta práctica, identificará los lugares que pueden ser vulnerables en las aplicaciones, reproduciendo las condiciones necesarias y similares para producir un ataque DoS.

INTRODUCCIÓN

ATAQUES DE DENEGACIÓN DE SERVICIO (DOS)

La intención de los ataques de Denegación de servicio (DoS) es forzar el bloqueo parcial o completo de una aplicación. Los ataques DoS no intentan destruir los datos de la aplicación, si no que su objetivo es hacer que la aplicación deje de proporcionar los servicios que tuviera encomendados. Los ataques DoS suelen ejecutarse contra redes y aplicaciones basadas en redes. Las aplicaciones basadas en Visual Basic .NET, las aplicaciones Web basadas en ASP.NET y las aplicaciones de servicio Web que se ejecutan en Internet son las más vulnerables a los ataques DoS. Los ataques DoS pueden ser de varios tipos. El código de Visual Basic .NET suele padecer con más facilidad ataques basados en los bloqueos de la aplicación y en el acaparamiento de memoria, de CPU y de otros recursos.

Los ataques DoS son los que tienen una defensa más complicada. Cuando se trata de varios millones de computadoras que accedan simultáneamente a su aplicación Web, no puede evitarse el ataque, si bien podrá intentar mitigar los efectos negativos. Por ejemplo, si la aplicación Web es capaz de detectar que se realizan más peticiones de las que se puedan controlar de forma razonable, la aplicación puede ser capaz de responder con una página HTML distinta notificando a los usuarios que el sitio Web está experimentando un volumen de tráfico superior al normal e indicando a los visitantes que vuelvan a intentar la conexión un poco más tarde. Esta solución resulta más favorable que el hecho de que la aplicación (o el sitio Web mostrado por la aplicación) parezca encontrarse bloqueada porque la página Web no se muestre en una cantidad de tiempo razonable.

Si la aplicación no responde bien ante una demanda inusualmente elevada de usuarios (sin que tenga que tratarse de un ataque), deberá intentar trabajar en el sentido de conseguir que la aplicación sea capaz de controlar los picos de carga. A continuación se muestran algunas formas de enfrentarse a cada uno de los tipos de ataque DoS.

Fallo de la aplicación - Pasar datos no esperados a una aplicación con la intención de hacerla fallar.

Puede resolverse a partir de la validación de todas las entradas; utilizando los controladores Try ... Catch en la forma apropiada para capturar y controlar todas las excepciones que se produzcan, probar el código y atacándolo para realizar una auditoría de seguridad del código de la aplicación y del diseño de la misma.

Acaparamiento de memoria - Pasar largas cantidades de datos a una aplicación para saturar la memoria del equipo en el que se ejecuta dicha aplicación.

Una solución es limitar el tamaño de la entrada y rechazar toda entrada repetitiva. Escribir los datos almacenados en la memoria en un archivo o base de datos para ahorrar memoria. Liberar juiciosamente ciertos objetos (por ejemplo, las matrices) que consuman gran cantidad de memoria, asignando el valor *Nothing* a la variable objeto cuando los datos dejen de ser necesarios.

Acaparamiento de CPU - Pasar datos a una aplicación para provocar que la aplicación entre en un bucle infinito o que provoque que la aplicación realice un uso intensivo de CPU durante un periodo de tiempo excesivo.

Para poder solucionar el problema debe identificarse y resolver todos los casos presentes en el código en los que un bucle o una llamada recursiva a una función puedan conducir a un bucle infinito cuando se reciba una entrada inesperada.

Acaparamiento de recursos - Intentar agotar un recurso limitado como, por ejemplo, el espacio disponible en disco. Una forma de realizar esta tarea es pasando grandes cantidades de datos a una aplicación y, posteriormente, guardarlos en el disco y forzar una condición de poco espacio en disco o sin espacio en disco.

Para responder a este tipo de ataques debe controlarse apropiadamente los archivos, conexiones a bases de datos y referencias a objetos que la aplicación ya no sean necesarios. Revisar el código para garantizar que las entradas no verificadas no van a provocar que la aplicación asigne una cantidad de recursos innecesaria o un recurso con un tamaño excesivo.

Acaparamiento del ancho de banda de la red - Reclutar la ayuda de muchas máquinas para efectuar peticiones repetitivas y simultáneas a una aplicación o sistema con la intención de saturar al servidor de la aplicación con las solicitudes de la red.

Fallo del sistema - Pasar datos no esperados a una aplicación o directamente al sistema (o a la aplicación del sistema) a través de un punto de entrada (por ejemplo, un puerto de la red) con la intención de provocar un fallo en el sistema.

ESCENARIO

En este ejercicio se crea una aplicación cliente que envía largas cantidades de datos a una aplicación servidor para saturar la memoria del equipo en el que se ejecuta dicha aplicación.

DESARROLLO

Cree una aplicación de biblioteca de clases con una clase *Public* denominada *Productos* que contiene un método *Public* denominado *AgregarProducto*, de la siguiente manera

Para crear la forma Servidor

20. Abra **Microsoft Visual Studio.NET**.
21. Haga clic en **Archivo**, apunte a **Nuevo** y después haga clic en **Proyecto**.

22. En el cuadro de diálogo **Nuevo proyecto**, seleccione **Proyectos de Visual Basic** en el panel **Tipos de proyecto**.
23. Seleccione **Biblioteca de clase** en el panel **Plantillas**.
24. Escriba **ServidorProducto** en el cuadro **Nombre** en una localidad de nombre **AtaqueDoS** y haga clic en **Aceptar**.
25. En el **Explorador de soluciones**, haga clic en el botón alterno sobre **Class1**, y después haga clic en **Renombrar,** cambie el nombre de la clase por Productos.vb.
26. Agregue con la siguiente estructura[26] dentro de Productos.vb:

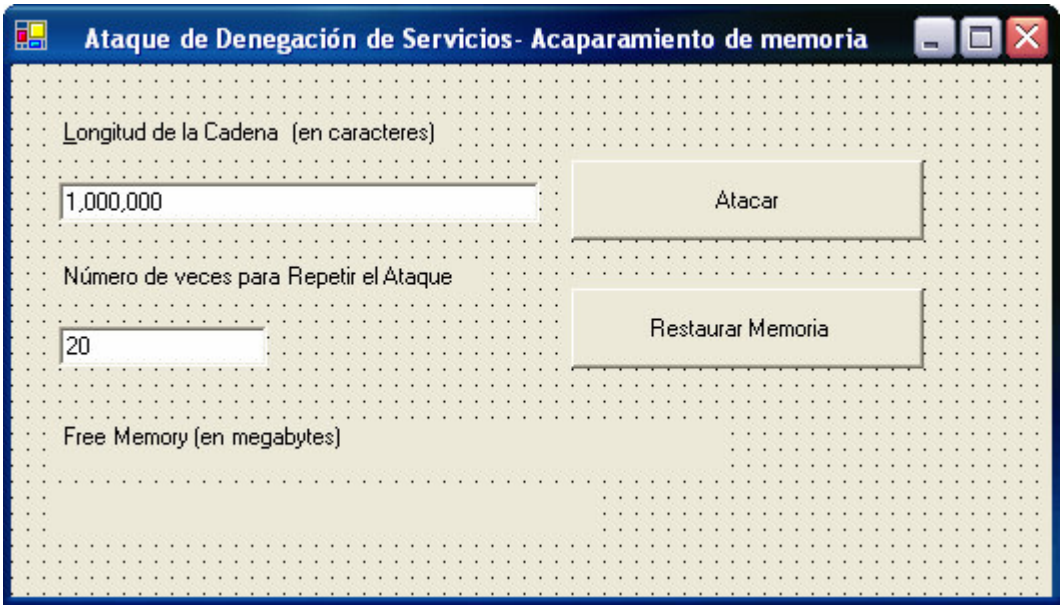
```
Public Class Productos
    Private m_NombresProductos As New Collection
    Public Sub AgregarProducto(ByVal NombresProductos As String)
        m_NombresProductos.Add(NombresProductos)
    End Sub
End Class
```

Para crear la forma Cliente

27. Haga clic en **Archivo**, apunte a **Nuevo** y después haga clic en **Proyecto**.
28. En el cuadro de diálogo **Nuevo proyecto**, seleccione **Proyectos de Visual Basic** en el panel **Tipos de proyecto**.
29. Seleccione **Aplicaciones Windows** en el panel **Plantillas**.
30. Escriba **ClienteAtaque** en el cuadro **Nombre** en misma localidad de nombre **AtaqueDoS** y haga clic en **Aceptar**.
31. Guarde el proyecto **ServidorProducto**
32. Haga clic en **Archivo**, apunte a **Agregar Proyecto** y después haga clic en **Proyecto Existente**.
33. En la localidad **AtaqueDoS\ServidorProducto** seleccione el proyecto **ServidorProducto,** que previamente se realizó, y haga clic en **Abrir.**
34. En el **Explorador de soluciones**, haga clic en el botón alterno sobre **Referencias** de **ClienteAtaque**, y después haga clic en Agregar referencia.
35. Seleccione la pestaña **Proyectos** dentro de la ventana **Agregar Referencia**, seleccione el proyecto **ServidorProducto** y por ultimo haga clic en **Aceptar**.
36. De doble clic en la **Form1.vb** del proyecto **ClienteAtaque** y agregue a la forma los siguientes componentes con los nombres siguientes:
    1. Etiqueta **lblLongitudString**,
    2. Caja de Texto **txtLongitudString**,
    3. Etiqueta **lblNumeroRepeticionAtaque**,
    4. Caja de Texto **txtNumeroRepeticionAtaque**,
    5. Etiqueta **lblMemoriaLibre**,
    6. Etiqueta **lblCantidadMemoriaLibre**,
    7. Botón **btnAtaque**,
    8. Botón **btnRestaurarMemoria**.

[26] El ejemplo de código anterior sólo tiene carácter ilustrativo. NO INTENTE EJECUTAR EL CÓDIGO EN LA PC. Si crea y ejecuta esta aplicación, la máquina puede llegar a tener un comportamiento errático.

Y modifique el atributo **Text** de los componentes una vez que fueron distribuidos de la siguiente manera en la **Form1**:



37. Arrastre de la **ToolBox** en **Componentes** el **PerformanceCounter** a la **Form1**.
38. Asigne los siguientes atributos al **PerformanceCounter1**:

    PerformanceCounter1

    NombreCategoria = Memoria
    NombreContador = MBytes disponibles

39. Haga doble clic en el botón **btnAtaque** y escriba las instrucciones necesarias a fin de obtener el siguiente código:

```
    Private Sub btnAtaque_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAtaque.Click

        Dim i As Integer
        Dim StringDemasiadaLarga As String

        'Nota: NumberSyles requiere  Imports System.Globalization
        Dim StringLen As Integer =
Integer.Parse(txtLongitudString.Text,NumberStyles.AllowThousands)
        'Pila de Aparcamiento de Memoria
         'Bucle del tamaño de repeticiones asignadas
        For i = 1 To CInt(txtNumeroRepeticionAtaque.Text)
            'Asignar una cadena de un millón de caracteres por default
            StringDemasiadaLarga = New String("X", StringLen)
            'Declare
             'Private m_products As New ProductServer.Products()
            m_productos.AgregarProducto(StringDemasiadaLarga)
            MostrarMemoriaLibre()
        Next
        MsgBox("Ataque Completado")
    End Sub
```

40. Agregue dentro del código el método MostrarMemoriaLibre()

```
    Private Sub MostrarMemoriaLibre()
        lblCantidadMemoriaLibre.Text = Format(PerformanceCounter1.RawValue(), "#,###,###")
        lblCantidadMemoriaLibre.Refresh()
    End Sub
```

41. Agregue el método MostrarMemoriaLibre() dentro de `Form1_Load` de la siguiente manera:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        MostrarMemoriaLibre()
    End Sub
```

42. Programe el botón **btnRestaurarMemoria** agregando el siguiente código:

```
    Private Sub btnRestaurarMemoria_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRestaurarMemoria.Click
        m_productos = Nothing
        GC.Collect()
        GC.WaitForPendingFinalizers()
        m_productos = New ServidorProducto.Productos
        MostrarMemoriaLibre()
    End Sub
```

Una vez creada la creada la Forma1, la aplicación cliente llama a la función *AgregarProducto* de forma repetida cada vez que se da clic en el botón **btnAtaque**, pasando cadenas extraordinariamente largas. Como el método *AgregarProducto* carece de todo tipo de validación de datos de entrada, y ni siquiera dispone de una comprobación para ver si el *NombresProductos* es único, el atacante forzará a la aplicación servidora a consumir toda la memoria disponible en el servidor. Sin embargo, incluso antes de que se consuma toda la memoria, el servidor se comportará de forma lenta y será incapaz de responder a ninguna nueva petición. Como otro ejemplo, ¿qué pasaría si la función *AgregarProducto* de la biblioteca de clases agregara la cadena a un archivo, tal como se muestra a continuación?:

```
  Public Sud AgregarProducto(ByVal NombresProductos As String)
  Dim hFile As Integer FreeFile()
  ' Agregar sentencias Microsoft.VisualBasic.Compatibility en la parte superior del módulo
de código
  FileOpen(hFile, VB6.GetPath & \NombresProductos.Txt”,_
                                                        OpenMode.Append)
  PrintLine(hFile, NombresProductos)
  FileClose(hFile)
  End Sub
```

Este código atacante producirá un ataque de acaparamiento de recursos o de espacio en disco cuyo resultado final será el mismo que en el caso del ataque de acaparamiento de memoria: el servidor dejará de responder.

La mejor defensa contra los ataques de acaparamiento de memoria y de espacio en disco es limitar el tamaño de la entrada permitida. En el ejemplo anterior, debe comprobarse siempre el tamaño del parámetro de entrada *NombresProductos*. Además, la subrutina *AgregarProducto* debe impedir la introducción de nombres duplicados. Por ejemplo, el

parámetro *NombresProductos* debe estar limitado a un tamaño razonable, tal como 20 caracteres. Para impedir las cadenas duplicadas, antes de agregar una cadena a la colección debe utilizarse el método *Contains* para confirmar que la colección no contiene ya la cadena que está a punto de añadir.

El código mostrado a continuación demuestra que la validación de la entrada (comprobando que la cadena de entrada tiene un tamaño razonable) y el rechazo de los nombres duplicados puede ayudar a impedir un ataque DoS o, al menos, obligar al atacante[27] a trabajar más duro y mejor para poder lanzar un ataque de este tipo.

```
'Agregar sentencias Imports System.Collections.Specialized
'en la parte superior del modulo de código
Private m_NombresProductos As New StringCollection
Public Sub AgregarProducto(ByVal NombresProductos As String)
    Const MAXLENGTH_NOMBREPRODUCTO = 20
     If NombresProductos.Length <= MAXLENGTH_NOMBREPRODUCTO AndAlso
           Not_ m_NombresProductos.Contains(NombresProductos) Then
        m_NombresProductos.Add(NombresProductos)
    Else
        Throw New ArgumentException("Nombre de producto invalido")
    End If
End Sub
```

[27] Para impedir que un atacante pueda llamar a un método *Public* contenido en una clase servidor, debe utilizarse también una estrategia profundamente defensiva, tal como utilizar técnicas de seguridad basada en roles para verificar que el usuario que realiza la llamada ha sido autenticado y tiene asignados los permisos basados en roles necesarios para llamar a la función.

PRÁCTICA #5: DEMOSTRACIÓN DE CIFRADO

OBJETIVO

Una vez terminada esta práctica, podrá desarrollar aplicaciones con rutinas de cifrado basadas en llaves públicas y privadas.

INTRODUCCIÓN

Existen dos tipos principales de algoritmos criptográficos: simétrico y asimétrico.

- Simétrico: Los algoritmos criptográficos simétricos tienen una sola clave secreta que se usa tanto para la encriptación como para la desencriptación. El uso de un algoritmo simétrico requiere que sólo el emisor y el receptor conozcan la clave secreta. DES_CSP, RC2_CSP y TripleDES_CSP son implementaciones de algoritmos simétricos.

- Asimétrico: Los algoritmos criptográficos asimétricos, también conocidos como algoritmos de clave pública, requieren que cada entidad mantenga un par de claves relacionadas: una clave privada y una pública. Ambas claves son únicas para cada entidad.

    - La clave pública puede estar disponible a todos y se usa para codificar datos que se enviarán al receptor.

El receptor debe mantener en anonimato la clave privada y se usa para descodificar mensajes codificados utilizando la clave pública del receptor. RSA_CP es una implementación de un algoritmo de clave pública.

¿Qué es el cifrado? Antes de analizar la forma de utilizar el cifrado en Visual Basic .NET, debe tenerse un conocimiento general de lo que se entiende por cifrado. El cifrado está relacionado con el hecho de mantener seguros los secretos al codificar mensajes para hacerlos ilegibles. En términos de cifrado, el mensaje original recibe el nombre de Mensaje en claro o texto claro, el mensaje codificado se denomina Mensaje cifrado o texto cifrado, el proceso de convertir el Mensaje en claro en Mensaje cifrado se denomina cifrado y el proceso de recuperar el Mensaje claro a partir del Mensaje cifrado se denomina descifrado.

El cifrado no se utiliza sólo en el ciberespacio o en misteriosos trabajos gubernamentales. Puede encontrarse ejemplos sobre este tema en casi todas las actividades de nuestra vida diaria. Las computadoras nos permiten cifrar complejos mensajes en tiempo real, para que el cifrado sea eficaz, el emisor y el receptor deben ser las únicas partes que conozcan la forma en que se ha cifrado y descifrado el mensaje. *Microsoft Windows* y *.NET Framework* cuentan con algoritmos robustos para llevar a cabo el cifrado.

Suele ser un error muy frecuente pensar que los algoritmos de cifrado y las funciones ***hash*** deben ser secretos para resultar seguros. Los algoritmos de cifrado y las funciones *hash* (o de resumen) son de dominio público y el código fuente asociado se distribuye libremente por

Internet. Sin embargo, siguen siendo seguros porque son irreversibles (en el caso de las funciones *hash*) o requieren que el usuario proporcione una clave secreta (en el caso de los algoritmos de cifrado). Siempre que sólo las partes autorizadas conozcan la clave secreta, el mensaje cifrado seguirá siendo seguro. El cifrado ayuda a garantizar tres cosas:

- Confidencialidad Sólo el receptor adecuado podrá descifrar el mensaje que haya enviado.
- Autenticación Los mensajes cifrados que reciba habrán sido generados por una fuente de confianza.
- Integridad Cuando envíe o reciba un mensaje no será alterado durante su trayecto.

Tres de los aspectos que ya se han mencionado y estudiado anteriormente, algunos mecanismos criptográficos son de un solo sentido; es decir, producen texto cifrado que no se puede descifrar. Un buen ejemplo de un cifrado unidireccional es el *hash*. Un ***hash*** es un número de gran tamaño, generado matemáticamente a partir de un mensaje en texto claro. Como el hash no contiene información sobre el mensaje original, éste no se puede deducir del hash. "¿Para qué se quiere utilizar el texto cifrado que no se puede descifrar?", puede uno preguntarse. Un hash resulta útil para verificar que alguien conoce un secreto sin tener que almacenar en realidad dicho secreto, así como para aprender a crear y utilizar un *hash* para verificar las contraseñas.

Un hash, como ya lo mencione, es un tipo de cifrado unidireccional. Algunas personas se refieren a las técnicas de ***hashing*** como cifrado; otras dicen que no se trata estrictamente de cifrado porque no se puede descifrar un *hash*. Los *hashes* son números de gran tamaño, generados sin más que codificar y condensar las letras de una cadena.

SHA- 1 es la abreviatura de **Secure Hashing Algorithm** (Algoritmo seguro de hashing). El "1' hace referencia a la revisión 1, que fue desarrollada en 1994. SHA-l acepta una cadena como entrada y devuelve un número de 160 bits (20 bytes). Como cualquier cadena se va a condensar en un número de tamaño fijo, el resultado recibe el nombre de resumen *hash* o *hash digest*, donde *digest* (resumen) indica un tamaño menor. Los resúmenes *hash* son cifrados unidireccionales porque es imposible obtener la cadena original a partir del *hash*. Un resumen *hash* es como la huella digital de una persona. La huella digital identifica de forma única a una persona sin revelar ningún detalle más acerca de la misma (no se puede determinar el color de ojos, la altura o el sexo de una persona analizando su huella digital).

Los resúmenes *hash* resultan útiles para verificar que alguien sabe una contraseña, sin tener que almacenar dicha contraseña. Al almacenar contraseñas sin cifrar en una base de datos se abren dos agujeros de seguridad:

- Si un intruso consigue acceder a la base de datos, podrá utilizar dicha información para iniciar una sesión posteriormente en el sistema utilizando el nombre de usuario y la contraseña de algún otro.
- Las personas suelen utilizar las mismas contraseñas en distintos sistemas, por lo que al perder nuestras contraseñas estamos posibilitando que el intruso se conecte a otros sistemas.

Como la contraseña se utiliza exclusivamente para autenticar al usuario, no existe ningún motivo para almacenar la contraseña en la base de datos. En su lugar, se puede almacenar un resumen *hash* de la contraseña. Cuando el usuario se conecte al sistema, se creará un resumen *hash* de la contraseña que haya tecleado y se comparará con el resumen *hash* almacenado en la base de datos. Si algún intruso consigue acceder a la tabla de contraseñas, no será capaz de utilizar el resumen hash para iniciar la sesión en el sistema porque tendría que conocer la contraseña sin cifrar, que no se encuentra almacenada en ninguna parte.

¿Cómo funciona un resumen hash?

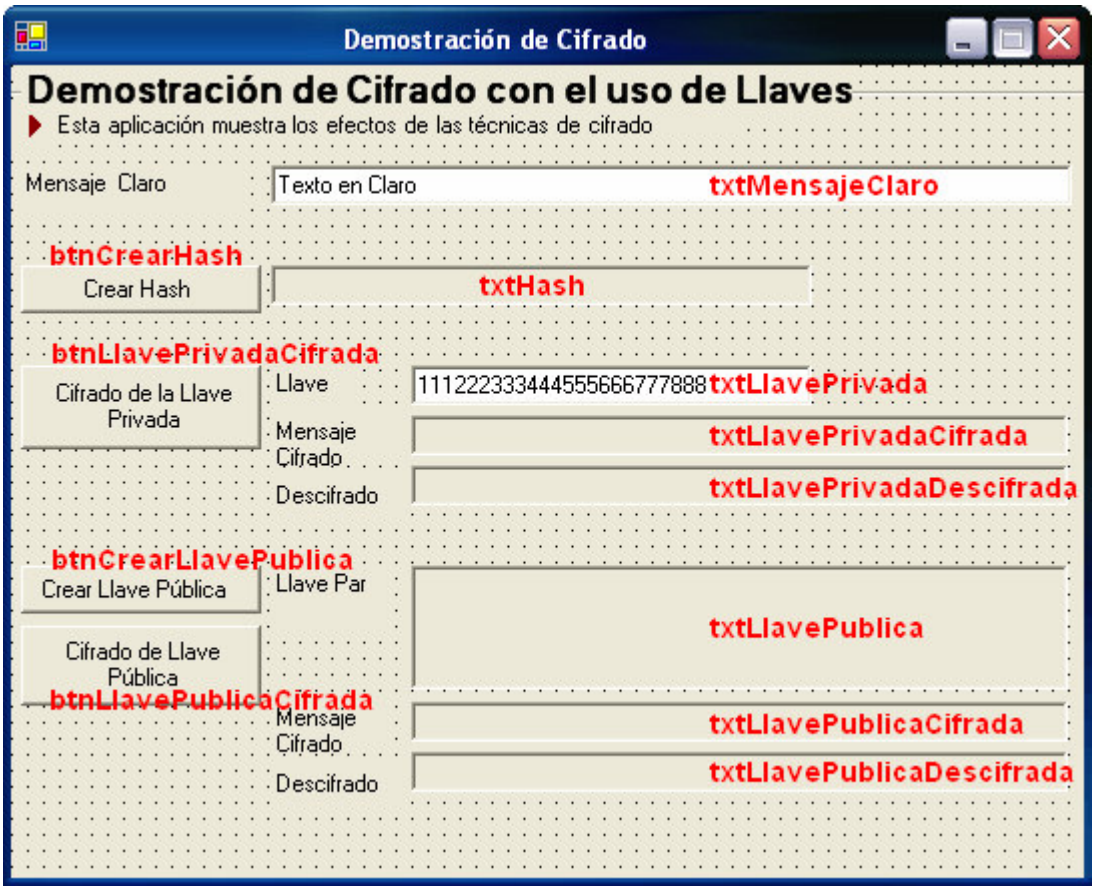
Si cada cadena proporciona un único resumen *hash*, ¿será posible descifrar el resumen hash y obtener la cadena original? Para responder a estas dos preguntas vamos a crear un sencillo algoritmo de *hash*. Es posible comenzar asignando a cualquier letra del abecedario un número, así a la A le corresponde el 1, a B el 2, a la C el 3, y así sucesivamente hasta la que le corresponderá el 26. A continuación utilizaremos estos valores para crear un *hash* sin más que sumar los números correspondientes a cada uno de los caracteres contenidos en la cadena. La cadena Visual Basic genera un *hash* de 24 porque V es la vigésimo segunda letra del abecedario y B es la segunda (22 + 2 = 24).

Conocido el hash con valor 24, ¿uno es capaz de obtener la cadena original? No. Este *hash* no indica la longitud de la cadena, cuál es el carácter inicial , o cualquier otro detalle sobre la cadena Original. Cuando diferentes cadenas producen el mismo valor hash se dice que ha habido una colisión. Un buen algoritmo de *hashing* deberá producir resultados que sean únicos y que estén libres de colisiones. SHA-1 produce resultados libres de colisione y codifica y condensa la cadena original de tal forma que se considera computacionalmente imposible obtener la cadena original.

DESARROLLO

Para crear la forma Cliente

43. Haga clic en **Archivo**, apunte a **Nuevo** y después haga clic en **Proyecto**.
44. En el cuadro de diálogo **Nuevo proyecto**, seleccione **Proyectos de Visual Basic** en el panel **Tipos de proyecto**.
45. Seleccione **Aplicaciones Windows** en el panel **Plantillas**.
46. Escriba **Cifrado** en el cuadro **Nombre** en la localidad de nombre **Cifrar** y haga clic en **Aceptar**.
47. En el **Explorador de soluciones**, haga clic en el botón alterno sobre **Cifrado**, apunte **Agregar**, y después haga clic en **Agregar componente existente**.
48. Busque y agregue la librería **LibreriaSeguridad.vb** (de no contar con el archivo, al final de esta práctica se presenta y explica el código fuente de LibreriaSeguridad.vb)
49. Cambie el nombre de **Form1.vb** a **frmDemoCifrado.vb**
50. De doble clic en la **frmDemoCifrado.vb** del proyecto **Cifrado** y arme frmDemoCifrado.vb con los siguientes componentes, además modifique el atributo **Text,** y **Name** de los componentes anteriores a fin de obtener la siguiente ventana:



51. Haga doble clic en el botón con **btnCrearHash** y escriba las instrucciones necesarias a fin de obtener el siguiente código:

```
Private Sub btnCrearHash_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCrearHash.Click
    txtHash.Text = Hash.CrearHash(txtMensajeClaro.Text)
End Sub
```

El código mostrado crea un hash SHA- 1 de la cadena `txtMensajeClaro.Text` y asigna el resultado a la cadena `txtHash.Text`

52. Haga doble clic en el botón con **btnLlavePrivadaCifrada** y escriba las instrucciones necesarias a fin de obtener el siguiente código:

```
Private Sub btnLlavePrivadaCifrada_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLlavePrivadaCifrada.Click
    txtLlavePrivadaCifrada.Text = LlavePrivada.Cifrar(txtMensajeClaro.Text,
txtLlavePrivada.Text)
    txtLlavePrivadaDescifrada.Text =
LlavePrivada.Descifrar(txtLlavePrivadaCifrada.Text, txtLlavePrivada.Text)
End Sub
```

El código mostrado cifra la cadena `txtMensajeClaro.Text` utilizando la cadena `txtLlavePrivada.Text` de 24 caracteres como clave y asigna la cadena cifrada a la cadena `txtLlavePrivadaCifrada.Text`.

Se descifra la cadena `txtLlavePrivadaCifrada.Text` utilizando la cadena `txtLlavePrivada.Text` como clave y asigna la cadena descifrada a `txtLlavePrivadaDescifrada.Text`. Si la clave es incorrecta o si se ha modificado `txtLlavePrivadaCifrada.Text` desde la operación de cifrado, el cifrado no tendrá éxito y se provocará una excepción.

53. Haga doble clic en el botón con **btnCrearLlavePublica** y escriba las instrucciones necesarias a fin de obtener el siguiente código:

```
Private Sub btnCrearLlavePublica_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCrearLlavePublica.Click
    txtLlavePublica.Text = LlavePublica.CrearParLlaves()
End Sub
```

El código mostrado crea y devuelve un nuevo par de claves pública/privada a `txtLlavePublica.Text`. El par de Llaves pública/privada es una estructura con ocho campos, incluyendo las Llaves pública y privada. Esta estructura es una cadena XML y debe mantenerse de forma confidencial porque la clave privada es la que se utiliza para descifrar los datos.

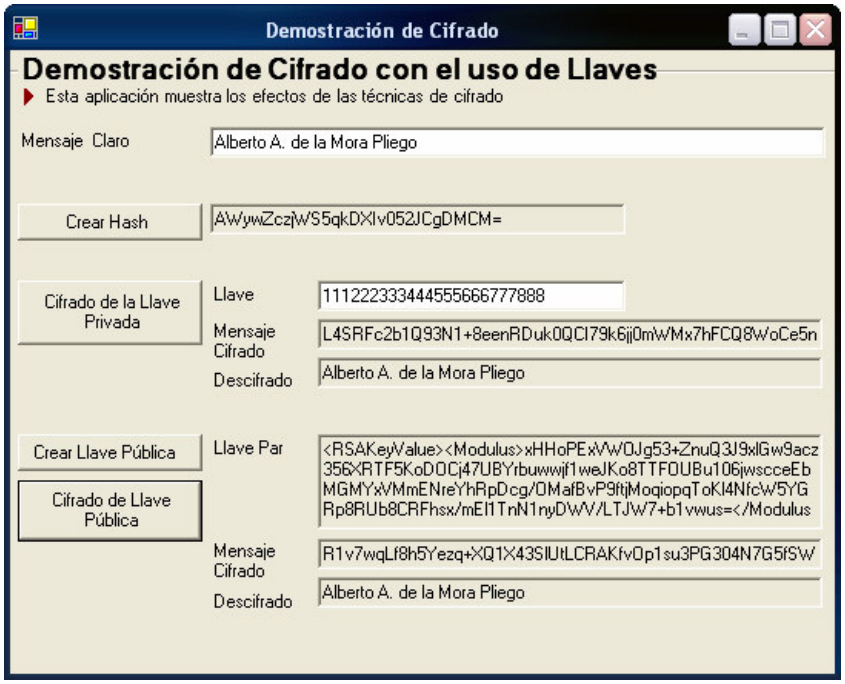
54. Haga doble clic en el botón con **btnLlavePublicaCifrada** y escriba las instrucciones necesarias a fin de obtener el siguiente código:

```
Private Sub btnLlavePublicaCifrada_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLlavePublicaCifrada.Click
    txtLlavePublicaCifrada.Text = LlavePublica.Cifrar(txtMensajeClaro.Text,
txtLlavePublica.Text)
    txtLlavePublicaDescifrada.Text =
LlavePublica.Descifrar(txtLlavePublicaCifrada.Text, txtLlavePublica.Text)
End Sub
```

El código mostrado a continuación cifra la cadena `txtMensajeClaro.Text` utilizando la clave pública `txtLlavePublica.Text` y, posteriormente, asigna la cadena cifrada a `txtLlavePublicaCifrada.Text`. Tanto la clave pública como el par de claves pública/privada se pueden utilizar para cifrar la cadena. Esta función puede cifrar una cadena de 58 caracteres como máximo. Si se intenta cifrar una cadena de más de 58 caracteres el cifrado fallará.

Otra parte del código descifra la cadena `txtLlavePublicaCifrada.Text` utilizando la clave privada `txtLlavePublica.Text` (no se puede utilizar la clave pública para descifrar la cadena). La cadena descifrada se asignará a `txtLlavePublicaDescifrada.Text`. Para que el descifrado se realice con éxito necesitará pasar la cadena original cifrada y la clave privada, asociada. Si se intenta pasar una clave privada errónea o pasar la clave pública, el descifrado fallará provocándose una excepción. De igual manera, si el texto cifrado ha sido modificado desde que fuera cifrado, la operación de descifrado fallará provocándose una excepción.

55. Ejecute la aplicación y observe como se generan, cifran y descifran los mensajes y las llaves.



Realice sus conclusiones y contemple sus aplicaciones en el mundo real.

Código Fuente se la librería **LibreriaSeguridad.vb**, realizada para funciones de seguridad realizadas en esta práctica.

```
Imports System.Security.Cryptography
Imports System.IO
Imports VB = Microsoft.VisualBasic
Imports System.Text
Imports System.Runtime.InteropServices


'Funciones de seguridad...
Namespace Hash
    Module Hash
        Function CrearHash(ByVal strFuente As String) As String
            Dim bytHash As Byte()
            Dim uEncode As New UnicodeEncoding
            'Almacenar la cadena fuente en un arreglo byte
            Dim bytFuente() As Byte = uEncode.GetBytes(strFuente)
            Dim sha1 As New SHA1CryptoServiceProvider
            'Crear el hash
            bytHash = sha1.ComputeHash(bytFuente)
            'regresa una cadena encoded de base64
            Return Convert.ToBase64String(bytHash)
        End Function
    End Module
End Namespace

Namespace LlavePrivada
    Module LlavePrivada
        Function Cifrar(ByVal strMensajeClaro As String, _
                ByVal strKey16 As String) As String
            Dim crp As New TripleDESCryptoServiceProvider
            Dim uEncode As New UnicodeEncoding
            Dim aEncode As New ASCIIEncoding
            'Almacena Mesaje en Claro como un byte array
            Dim bytMensajeClaro() As Byte = uEncode.GetBytes(strMensajeClaro)
            'Crea una stream de memoria  para cargar el  mensaje cifrado
            Dim stmMensajeCifrado As New MemoryStream
            'Llave Privada
            crp.Key = aEncode.GetBytes(Left(strKey16, 16))
            'Se inicializa la semilla del vector de cifrado
            crp.IV = aEncode.GetBytes(VB.Right(strKey16, 8))
            'Crear un escritor de cifrado para cifrar un bytearray
            'dentro de un stream
            Dim csCifrado As New CryptoStream(stmMensajeCifrado, _
                crp.CreateEncryptor(), CryptoStreamMode.Write)
            csCifrado.Write(bytMensajeClaro, 0, bytMensajeClaro.Length)
            csCifrado.FlushFinalBlock()
            'regresa el resultado como una cadena encoded de Base64
            Return Convert.ToBase64String(stmMensajeCifrado.ToArray())
        End Function

        Function Descifrar(ByVal strMensajeCifrado As String, _
                ByVal strKey16 As String) As String
            Dim crp As New TripleDESCryptoServiceProvider
            Dim uEncode As New UnicodeEncoding
            Dim aEncode As New ASCIIEncoding
            'Almacena mensaje cifrado como un arreglo byte
            Dim bytMensajeCifrado() As Byte = _
            Convert.FromBase64String(strMensajeCifrado)
            Dim stmPlainText As New MemoryStream
            Dim stmMensajeCifrado As New MemoryStream(bytMensajeCifrado)
```

```
            'Llave Privada
            crp.Key = aEncode.GetBytes(VB.Left(strKey16, 16))
            'Inicializar el vector
            crp.IV = aEncode.GetBytes(VB.Right(strKey16, 8))
            ' Cree un decodificador de stream del cifrado para descifrar
            ' un stream del mensaje cifrado en un stream del mensaje claro
            Dim csDecrypted As New CryptoStream(stmMensajeCifrado, _
                crp.CreateDecryptor(), CryptoStreamMode.Read)
            Dim sw As New StreamWriter(stmPlainText)
            Dim sr As New StreamReader(csDecrypted)
            sw.Write(sr.ReadToEnd)
            'Limpie después
            sw.Flush()
            csDecrypted.Clear()
            crp.Clear()
            Return uEncode.GetString(stmPlainText.ToArray())
        End Function
    End Module
End Namespace

Namespace LlavePublica
    Module LlavePublica
        Function CrearParLlaves() As String
            'Crear un nuevo par de llaves aleatorias
            Dim rsa As New RSACryptoServiceProvider
            CrearParLlaves = rsa.ToXmlString(True)
            rsa.Clear()
        End Function
        Function ObtenerLlavePublica(ByVal strLlavePrivada As String) As String
            'Obtener la llave publica a partir de
            'par de llaves publica/privada
            Dim rsa As New RSACryptoServiceProvider
            rsa.FromXmlString(strLlavePrivada)
            Return rsa.ToXmlString(False)
        End Function
        Function Cifrar(ByVal strMensajeClaro As String, _
                ByVal strLlavePublica As String) As String
            'cifrar una adena usando la llave privada o publica
            Dim rsa As New RSACryptoServiceProvider
            Dim bytMensajeClaro() As Byte
            Dim bytMensajeCifrado() As Byte
            Dim uEncode As New UnicodeEncoding
            rsa.FromXmlString(strLlavePublica)
            bytMensajeClaro = uEncode.GetBytes(strMensajeClaro)
            bytMensajeCifrado = rsa.Encrypt(bytMensajeClaro, False)
            Cifrar = Convert.ToBase64String(bytMensajeCifrado)
            rsa.Clear()
        End Function
        Function Descifrar(ByVal strMensajeCifrado As String, _
                ByVal strLlavePrivada As String) As String
            'Descifrar una cadena usando la llave privada
            Dim rsa As New RSACryptoServiceProvider
            Dim bytMensajeClaro() As Byte
            Dim bytMensajeCifrado() As Byte
            Dim uEncode As New UnicodeEncoding
            rsa.FromXmlString(strLlavePrivada)
            bytMensajeCifrado = Convert.FromBase64String(strMensajeCifrado)
            bytMensajeClaro = rsa.Decrypt(bytMensajeCifrado, False)
            Descifrar = uEncode.GetString(bytMensajeClaro)
            rsa.Clear()
        End Function
    End Module
End Namespace
```

```
 Namespace Settings
     Module Settings
         <StructLayout(LayoutKind.Sequential)> Private Structure DATA_BLOB
             Dim cbDato As Integer
             Dim pbDato As IntPtr
         End Structure
         <StructLayout(LayoutKind.Sequential)> Private Structure
PROYECTOCIFRAR_STRUCTURAPROMPT
             Dim cbTamanio As Integer
             Dim dwPromptFlags As Integer
             Dim hwndApp As IntPtr
             Dim szPrompt As String
         End Structure
         Private Declare Function DatosProtegCifrado Lib "Crypt32.dll"(_
             ByRef pDataIn As DATA_BLOB, _
             ByVal szDataDescr As String, _
             ByRef pOptionalEntropy As DATA_BLOB, _
             ByVal pvReserved As IntPtr, _
             ByRef pPromptStruct As PROYECTOCIFRAR_STRUCTURAPROMPT, _
             ByVal dwFlags As Integer, _
             ByRef pDataOut As DATA_BLOB) As Boolean
         Private Declare Function DatosDesprotegCifrado Lib "Crypt32.dll" ( _
             ByRef pDataIn As DATA_BLOB, _
             ByVal szDataDescr As String, _
             ByRef pOptionalEntropy As DATA_BLOB, _
             ByVal pvReserved As IntPtr, _
             ByRef pPromptStruct As PROYECTOCIFRAR_STRUCTURAPROMPT, _
             ByVal dwFlags As Integer, _
             ByRef pDataOut As DATA_BLOB) As Boolean
         Function SalvarEncriptado(ByVal strNombreSetting As String, ByVal strValor As
String) As Boolean
             Dim bytMensajeClaro() As Byte
             Dim bytMensajeCifrado() As Byte
             Dim strMensajeCifrado As String
             Dim strNombreArchivo As String
             Dim uEncode As New UnicodeEncoding
             Dim strExitoso As Boolean = False
             Dim bbMensajeClaro, bbMensajeCifrado, bbEntropy As DATA_BLOB
             Dim pmt As PROYECTOCIFRAR_STRUCTURAPROMPT
             Dim intNumeroArchivo As Integer
             'Inicialice la estructura del pmt
             pmt.cbTamanio = Marshal.SizeOf(pmt)
             pmt.hwndApp = IntPtr.Zero
             pmt.szPrompt = vbNullString
             ' Convierte el MensajeClaro en un arreglo byte, y copia
             ' a la memoria global
             bytMensajeClaro = uEncode.GetBytes(strValor)
             bbMensajeClaro.pbDato = Marshal.AllocHGlobal(bytMensajeClaro.Length)
             If bbMensajeClaro.pbDato.ToInt32 = 0 Then
                 MsgBox("Global Alloc Falló", , "ProtectString")
                 Return False
             End If
             bbMensajeClaro.cbDato = bytMensajeClaro.Length
             Marshal.Copy(bytMensajeClaro, 0, bbMensajeClaro.pbDato,
bytMensajeClaro.Length)
             'Llamada a windows Crypto API DatosProtegCifrado a cifrado el Mensaje Claro
             strExitoso = DatosProtegCifrado(bbMensajeClaro, "", bbEntropy, IntPtr.Zero,
pmt, 1, bbMensajeCifrado)
             If strExitoso = False Then
                 MsgBox("CryptProtect Falló", , "ProtectString")
                 Return False
             End If
```

```
            'el resultado es almacenado en un bloque de memoria. Convierte este bloqu a
una cadena
            ReDim bytMensajeCifrado(bbMensajeCifrado.cbDato)
            Marshal.Copy(bbMensajeCifrado.pbDato, bytMensajeCifrado, 0,
bbMensajeCifrado.cbDato)
            Marshal.FreeHGlobal(bbMensajeClaro.pbDato)
            strMensajeCifrado = Convert.ToBase64String(bytMensajeCifrado)
            'Salva el  setting sifrado a un archivo en el folder aplicación del usuario
            intNumeroArchivo = FreeFile()
            strNombreArchivo =
System.Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) & "\" &
strNombreSetting & ".txt"
            FileOpen(intNumeroArchivo, strNombreArchivo, OpenMode.Output,
OpenAccess.Write)
            PrintLine(intNumeroArchivo, strMensajeCifrado)
            FileClose(intNumeroArchivo)
            Return True
        End Function
        Function CargarCifrado(ByVal StrSettingName As String) As String
            Dim strMensajeCifrado As String
            Dim uEncode As New UnicodeEncoding
            Dim bytMensajeCifrado() As Byte
            Dim blnExitoso As Boolean = False
            Dim bbMensajeClaro, bbCifrado, bbEntropy As DATA_BLOB
            Dim pmt As PROYECTOCIFRAR_STRUCTURAPROMPT
            Dim strNombreArchivo As String
            Dim intNumeroArchivo As Integer
            'Cargar la sentencia cifrada del archivo dentro del
            ' folder aplicaciones del usuario
            intNumeroArchivo = FreeFile()
            strNombreArchivo = System.Environment.GetFolderPath( _
              Environment.SpecialFolder.ApplicationData) & _
              "\" & StrSettingName & ".txt"
            FileOpen(intNumeroArchivo, strNombreArchivo, _
              OpenMode.Input, OpenAccess.Read)
            strMensajeCifrado = LineInput(intNumeroArchivo)
            FileClose(intNumeroArchivo)
            'inicializar la estructura pmt
            pmt.cbTamanio = Marshal.SizeOf(pmt)
            pmt.hwndApp = IntPtr.Zero
            pmt.szPrompt = vbNullString
            'copiar el mensaje ifrado dentro un arreglo byte y almacenarlo
            'en memoria global
            bytMensajeCifrado = Convert.FromBase64String(strMensajeCifrado)
            bbCifrado.pbDato = Marshal.AllocHGlobal(bytMensajeCifrado.Length)
            If bbCifrado.pbDato.ToInt32 = 0 Then
                MsgBox("Global Alloc failed", , "UnprotectString")
                Return ""
            End If
            bbCifrado.cbDato = bytMensajeCifrado.Length
            Marshal.Copy(bytMensajeCifrado, 0, bbCifrado.pbDato, bbCifrado.cbDato)
            'llamada al API Windows de DatosDesprotegCifrado
            blnExitoso = DatosDesprotegCifrado(bbCifrado, vbNullString, bbEntropy,
IntPtr.Zero, pmt, 1, bbMensajeClaro)
            If blnExitoso = False Then
                MsgBox("CryptUnprotect failed", , "UnprotectString")
                Return ("")
            End If
            'el usuario almacena en un bloque de memoria. Convierte a una
            'cadena y la regresa al usuario
            Marshal.FreeHGlobal(bbCifrado.pbDato)
            Dim plainText(bbMensajeClaro.cbDato) As Byte
            Marshal.Copy(bbMensajeClaro.pbDato, plainText, 0, bbMensajeClaro.cbDato)
```

```
            Return uEncode.GetString(plainText)

MsgBox(System.Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData))
        End Function
    End Module

End Namespace
```