

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Software Seguro

en

Visual BASIC.NET

T E S I S

PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

ALEJANDRO FÉLIX REYES



MÉXICO, D. F. **2006**

Software Seguro en Visual BASIC.NET

Tesis para obtener el grado de Ingeniero en Computación

Presenta: Alejandro Félix Reyes

Director: M.C. Ma. Jaquelina López Barrientos

A mi **madre**, *por ser simplemente la mejor.*
A mi **abuela**, *por consentirme y bendecirme.*
A **Valeria**, *sin tu compañía no hubiera llegado hasta aquí*

A **Jaquelina**, *por guiar como nadie a un tesista tan latoso.*
A **Pablo**, *por mostrarme que la ingeniería es más que números.*
A **Orlando**, *por guiar a un estudiante perdido en la computación.*

A **Antonio**, *por encausarme a mi más grande pasión.*
A **Sigrid**, **Perseo** *y* **Rodolfo** *por creer en su cuate.*
A **Jesús**, *por revisar pacientemente este trabajo.*
A todos y cada uno de mis amigos por todo su apoyo.

A todos y cada uno de mis maestros por enseñarme
a ser un ingeniero de calidad.

Al **Laboratorio de Microsoft** *y a la* **Facultad de Ingeniería**
por acogerme y permitirme ser cada día mejor.

Y a la **UNAM** *por permitirme vivir la más hermosa aventura de mi vida.*

¡¡Gracias!!

TABLA DE CONTENIDO

INTRODUCCIÓN 1

1. LA PLATAFORMA .NET 3

1.1. HISTORIA 4

1.1.1. La Necesidad 4

1.1.2. Microsoft .NET 6

1.1.3. Visual Studio.NET 6

1.1.4. .NET en Crecimiento 7

1.1.5. Visual Studio 2003 7

1.2. FRAMEWORK .NET 9

1.2.1. ¿Qué es .NET? 9

a) Framework .NET (Ambiente de Ejecución) 9
b) Active Server Pages .NET (Páginas de Servicio Activo .NET) 9
c) Web Services .NET (Servicios de Internet XML) 9
d) Windows Forms (Formularios de Windows) 10
e) ActiveX Data Object .NET (Objetos de Datos ActiveX .NET) 10

1.2.2. El Ambiente de Ejecución .NET 10

a) La arquitectura del Framework .NET 11
b) Common Language Runtime 12
c) Objetos del Framework .NET 14

1.3. PERSPECTIVAS 20

1.3.1. Problemas legales 20

1.3.2. Implementaciones alternativas 20

1.3.3. Estandarización 21

1.3.4. Visual Studio 2005 21

2. VISUAL BASIC.NET 23

2.1. DE BASIC A BASIC.NET 24

2.1.1. Inicios 24

2.1.2. Expansión 25

2.1.3. Microsoft Quick BASIC 25

2.1.4. Visual BASIC 26

2.2. VISUAL BASIC.NET 29

2.2.1. Historia 29

2.2.2. Todas las Plataformas Nacen Iguales 30

2.2.3. Sintaxis Básica 30

a) Las Variables 31
b) Operadores 31

c) Estructuras de Decisión y de Ciclo 32
d) Arreglos y Colecciones 33
e) Cadenas 34
f) Manejador de errores 35
g) Programación Orientada a Objetos 35
2.3. DIFERENCIAS ENTRE VB6 Y VB.NET 37
a) Cambios en las ventanas y en el diseño 37
b) Cambios en la asignación de teclado 38
c) Cambios en los proyectos 38
d) Cambios en el Desarrollo para Internet 39
e) Cambios en conectividad con Bases de Datos 39
f) Cambios en los Controles 39
g) Cambios en el sistema de coordenadas 41
h) Cambios en los eventos 41
i) Cambios en las fuentes 41
j) Cambios en los eventos de formularios 41
k) Cambios en los gráficos 42
l) Cambios en las matrices de controles 42
m) Cambios en MDI 42
n) Cambios en los cuadros de diálogo 42
o) Cambios de color 43
p) Cambios en la depuración 43
q) Cambios en los espacios de nombre 43
r) Cambios en tiempo de ejecución 43
s) Cambios en los tipos de datos universales 44
t) Cambios en los tipos de datos enteros 44
2.4. PERSPECTIVAS 45
2.4.1. Visual BASIC.NET 2005 45
2.4.2. La Supervivencia de BASIC.NET 45

3. FUNDAMENTOS DE SEGURIDAD INFORMÁTICA 47

3.1. HISTORIA 48
3.1.1. Inicio – La Criptografía 48
3.1.2. Siglo XX 49
3.1.3. La Informática (Virus y Hackers) 49
3.1.4. Nuestros Días 52
3.2. CONCEPTOS BÁSICOS 53
3.2.1. Vulnerabilidades 53
3.2.2. Amenazas 53
3.2.3. Ataques 54
3.2.4. Características de un Sistema Seguro 55
3.2.5. Código Malicioso 55
a) Puertas Traseras 56
b) Bombas Lógicas 57
c) Caballos de Troya 57
d) Zombi 57
e) Virus 57
f) Gusanos 59

3.2.6. Ingeniería Social *60*
3.2.7. Hoax y Spam *61*
3.2.8. El Usuario Común *61*
3.3. ESTÁNDARES INTERNACIONALES 63
3.3.1. La Importancia de los Estándares *63*
3.3.2. TCSEC *64*
3.3.3. ISO 17799 ..... *65*
3.3.4. Criterios Comunes *67*
3.3.5. Otras Normas para Ingeniería de Software *69*
a) CMMI 69
b) ISO/IEC 15504 70
3.4. CRIPTOGRAFÍA 71
3.4.1. Conceptos Básicos *71*
3.4.2. Criptografía *71*
3.4.3. Criptoanálisis *71*
3.4.4. Cifrado Simétrico *72*
3.4.5. Cifrado Asimétrico *74*

4. SOFTWARE SEGURO 79

4.1. INGENIERÍA DE SOFTWARE 80
4.1.1. Historia *80*
4.1.2. Cualidades de un Sistema de Información *81*
4.1.3. Ciclo de Vida de los Sistemas de Información *83*
a) Modelo de Cascada 83
b) Modelo Evolutivo 84
c) Modelo de transformación 85
d) Modelo Espiral 86
e) Modelo Mixto 87
4.1.4. CVSI Seguro *88*
a) Fase de Diseño 89
b) Fase de Desarrollo 93
c) Fase de Pruebas 94
d) Fase de Distribución y Mantenimiento 96
4.2. PROGRAMACIÓN SEGURA 97
4.2.1. Antes de Programar *97*
4.2.2. Programación Eficiente *98*
a) Antiguas Creencias 98
b) Detección del Punto Clave 99
c) Técnicas de Optimización 100
d) Funcionalidad 104
4.2.3. Buenos Hábitos de Programación *104*
a) Nombres 105
b) Option Strict 107
c) Formato de Código 108
d) Comentarios 110

e) Manejador de Errores ... 113
f) Educación y Cultura ... 115
4.3. CÓDIGO SEGURO ... 116
4.3.1. El Desbordamiento de Buffer ... 117
a) Desbordamiento de Pila ... 117
b) Desbordamiento de Heap ... 118
c) Errores de Índice de Arreglos ... 120
d) Errores en Formato de Cadenas ... 121
e) Errores de ANSI y UNICODE ... 122
4.3.2. Inyección SQL ... 123
4.3.3. Errores Criptográficos ... 125
a) Claves ... 125
b) Algoritmos ... 126
c) ¿Qué utilizar? ... 126
4.3.4. Introducción de Datos ... 127
5. SEGURIDAD EN VISUAL BASIC.NET ... 129
5.1. PROGRAMACIÓN SEGURA ... 130
5.1.1. Primera Línea de Defensa ... 130
5.1.2. Desbordamiento de Búfer ... 131
5.1.3. Expresiones Regulares ... 131
5.1.4. Negación de Servicios ... 134
5.1.5. Ataques a Directorios ... 136
5.1.6. Inyección SQL ... 138
a) Validaciones ... 138
b) Parámetros ... 139
5.1.7. Criptografía ... 139
5.1.8. Manejador de Errores ... 145
a) Manejadores Anidados ... 146
b) Generando Errores ... 147
CONCLUSIONES ... 151
GLOSARIO ... 155
BIBLIOGRAFÍA Y MESOGRAFÍA ... 161

Introducción

El Ingeniero en Computación, durante su formación profesional, estudia una gran variedad de materias que le permiten desarrollarse en diversas actividades, entre ellas de manera puntual el desarrollo de sistemas. Por ésto, el software desarrollado debe ser eficiente, es decir, debe cumplir con varios parámetros, entre los cuales podríamos mencionar su buen diseño (propiamente articulado y analizado con base en los requerimientos específicos de cada institución, organismo o usuario), que sea tolerante a fallos, que emita reportes de errores detallados, que posea un tiempo de ejecución corto y un uso de memoria óptimo. Para ello, el software de aplicación debe ser diseñado y desarrollado haciendo uso de métodos que permitan garantizar su eficacia, ésto aunado a un buen y completo periodo de pruebas; dónde se analice no sólo que el software se comporte como esperamos, sino que además cumpla con un mínimo indispensable de pruebas de seguridad.

Debido a diferentes causas (como son: tiempo limitado, descuido, negligencia, conformismo, etc.) generalmente los sistemas no son diseñados con buenos métodos de programación tales como el correcto empleo de estructuras de datos, el buen uso y liberación de memoria o la muy necesaria validación de datos de entrada, lo que conlleva a tener diversas vulnerabilidades en los sistemas desarrollados, por lo que es necesario explorar las metodologías que permitan desarrollar de una manera eficiente software a la par de las diversas necesidades de seguridad que hay actualmente.

Con las metodologías exploradas, los sistemas desarrollados podrán ser revisados por su diseñador y esto permitirá tomar las medidas pertinentes para evitar tener que llegar a la corrección del producto. Es importante no alcanzar la etapa de corrección, y que las vulnerabilidades así como errores, sean detectados lo más temprano en la vida del software, pues esto prevendrá la realización de parches u otros mecanismos de corrección. Todo esto con el objeto de que el software producido cumpla con su labor de una manera eficiente, esto es, evitar vulnerabilidades que puedan ser creadas durante el desarrollo o en la corrección del software y procurando con ello no dar una mala imagen de nuestros productos al mercado.

Para lograr este propósito, he dividido el presente trabajo en 5 capítulos, donde primero se analiza la plataforma .NET, explicando su funcionamiento; luego se analiza al lenguaje de programación Visual BASIC.NET, dando un panorama general de su funcionamiento, historia y perspectivas. En el tercer capítulo, se exploran los conocimientos de seguridad mínimos necesarios; mientras que para el cuarto capítulo se exploran las amenazas y el ciclo de vida de los sistemas de información para generar software seguro. Finalmente en el quinto capítulo se exploran algunas formas de explotar

la plataforma de una manera correcta, probando porqué se ha elegido y cómo nos ayuda a construir software seguro.

Por lo tanto, ante toda esta panorámica el siguiente trabajo de tesis, busca cumplir con los siguientes objetivos:

- Explorar y diseñar las estrategias a seguir para la creación de software seguro desde su diseño hasta su liberación.
- Analizar porqué VB.NET es una herramienta de desarrollo profesional.
- Desarrollar una guía de costumbres para el desarrollo de aplicaciones seguras.

Todas ellas se explorarán en 5 capítulos, donde tendremos la oportunidad de empezar desde cero en nuestro conocimiento de todos los conceptos. Para ello, dedicaré los 3 primeros capítulos para establecer los antecedentes necesarios para el correcto aprovechamiento y valoración de este trabajo. Después de los antecedentes, viene el desarrollo en pleno del trabajo de tesis. En el capítulo 4, exploro absolutamente todos los temas que pienso son de mayor importancia para la creación de un software seguro (sin importar el tipo de proyecto o el lenguaje de programación). Después de esto, en el último capítulo, analizo en particular las estrategias básicas a seguir si la elección es el lenguaje de programación Visual BASIC.NET, por lo que en esta sección encontraremos la mayor cantidad de código.

Espero que una vez explorado este trabajo, el concepto de seguridad y de software seguro quede más claro y más afianzado en quien lo lea.

1. La Plataforma .NET

La plataforma Microsoft .NET es la última respuesta creada por esta empresa para la elaboración de software. Esta herramienta de última generación está pensada para el desarrollo en Internet y para que el programador se olvide de minucias como el uso de la memoria y los apuntadores y se centre más en programar de una manera más eficiente y eficaz.

Como el objetivo de esta tesis es el desarrollo de Software Seguro en Visual BASIC.NET, necesitaremos primero entender dónde nos encontramos parados. Para ello el estudio o breve conocimiento de la plataforma, así como las necesidades que cubre y las soluciones que presenta son de vital importancia para entender porqué se ha escogido este marco de trabajo como base para un proyecto de tesis.

Es así como en este primer capítulo trataré de dar de una manera breve, pero concisa, y sobre todo muy sencilla, el funcionamiento de la plataforma. Recalco el adjetivo breve, ya que un tratado sobre el Framework .NET sería verdaderamente largo y complejo, pues estamos ante más que una máquina virtual, que sólo administra recursos (como Java) sino que nos encontramos ante un sistema operativo montado sobre otro sistema operativo (en este caso Windows). Si tratara de dar a conocer todos y cada uno de los casos, dinámicas y controles del Framework .NET, nos encontraríamos un trabajo extenuante, ajeno al estudio de esta tesis.

Sin embargo, recomiendo a quienes les agrade la plataforma y deseen trabajar sobre ella (no sólo con BASIC.NET), que tomen ésto sólo como una introducción y profundicen sobre los temas tratados y exploren otros que no se incluyen, pues su estudio es fascinante.

1.1. Historia

1.1.1. La Necesidad

La historia de .NET es un tanto dispersa, pues fueron muchos los factores que participaron en su concepción. Después de una profunda investigación, puedo decir que son 3 las principales causas que permitieron el nacimiento de la tecnología .NET de Microsoft:

- La probada funcionalidad de la máquina virtual de Java.
- La necesidad de un lenguaje de programación único.
- El auge de Internet en las comunicaciones internacionales.

Java, junto con C++, a partir de la década de los noventa, ha definido el auge de los lenguajes de programación orientados a objetos. Los métodos de programación estructurada habían llegado a sus límites y se necesitaba una nueva forma de organización de funciones y de información. Con este acercamiento, surge la Programación Orientada a Objetos, que cambia los paradigmas y vuelve del uso común términos como polimorfismo, herencia y clases.

Sin embargo, lo que hace que Java sea mucho más aceptado que C++ es su gran portabilidad. El secreto de esta portabilidad radica en su máquina virtual. Instalando la Máquina Virtual de Java (JVM) en cualquier sistema operativo, aseguramos la correcta ejecución del código. Lo mejor, es que debido a su gran apogeo (sobre todo al lenguaje script JavaScript) la mayoría de las computadoras personales la tienen instalada, por lo que podemos decir que el código de Java es portable al 100%.

Además, la JVM, permitía que su código fuera ejecutado sólo a nivel de la máquina virtual, por lo que ofrecía una seguridad casi absoluta, pues nunca tiene acceso al Sistema Operativo, así que estamos a salvo de cualquier mal funcionamiento de los programas ejecutados en Java.

Sin embargo su principal problema era que si queríamos aprovecharnos de esta portabilidad y seguridad ofrecida, teníamos que saber Java. Esto era todo un problema, pues los programadores de BASIC, Cobol, C, C++, etc. no tenían acceso a estas ventajas, que si bien podrían valer la migración, ésta era frenada por las limitaciones del mismo lenguaje (por ejemplo, la falta de sencillez que los programadores de BASIC tienen como prioridad).

Ésto nos lleva directo al segundo punto, que es la necesidad de un único lenguaje de programación. Esto suena como la panacea. Desde que los programadores programan, se ha buscado ese lenguaje universal, que permita manejar todos los casos, todos los niveles, con la precisión requerida. Se ha buscado, pero no se ha conseguido y es por lo mismo que surgieron diferentes lenguajes de programación que para darle mayor

importancia a ciertos aspectos (BASIC prioritiza la sencillez), dejan otros de lado (BASIC abandona la comunicación con la memoria).

La comunicación entre diferentes lenguajes se lograba sólo mediante "traductores" que permitían escribir en diferentes lenguajes diferentes proyectos dentro de un mismo grupo de proyectos (por ejemplo COM). Sin embargo la programación de estos traductores era muy difícil, pues había casos irreconciliables entre los diferentes lenguajes. Un ejemplo típico es el manejo de cadenas en BASIC (por medio de funciones), C (arreglo de caracteres) y Java (objetos). La mayoría de las veces era mejor aprender el lenguaje donde se programaba la mayor parte del código que programar el traductor.

Finalmente, nos encontramos con el vertiginoso crecimiento de Internet. En los sesenta cuando ARPANET era concebido, no nos imaginábamos que llegaría el día en que tendríamos conectadas en una misma red global más de 25 000 redes locales. Por lo mismo, no es de extrañar que Internet sea en estos días, el terreno que todos debemos dominar.

Aunado al rápido crecimiento de Internet, los equipos para navegarlo y la rapidez de transmisión de datos, no sólo se han hecho mejores, sino que también se han hecho más baratas; consecuentemente el crecimiento parece no tener aun un límite visible. Por lo mismo, las aplicaciones que ahora sólo corren en las computadoras localmente, están perdiendo fuerza, y la ínterconectividad ha ido en aumento, es decir, aquel equipo que no se puede comunicar en la red, está destinado a perecer.

Internet no nació para ser programado, ni mucho menos para ser programado de manera segura. Por lo que la necesidad primordial, es tener un lenguaje de programación que cumpla con todas las expectativas, y que además sea 100% elaborado para Internet.

Java seguía siendo a finales de los noventa la única respuesta confiable para desarrollo en Internet. Microsoft lo sabía, y debido a esto, dependía de él. Pero no tardó mucho en tomar la iniciativa. Algunas tecnologías utilizadas en .NET fueron originalmente desarrolladas por Microsoft como su propia versión de Java. Cuando Microsoft decidió terminar su dependencia de las tecnologías de Java en 1998, su producto existente, denominado Visual J++, fue transformado para ser la base del proyecto .NET

1.1.2. Microsoft .NET

A principios del año 2000, Microsoft anunció la iniciativa .NET (conocida por el nombre clave de "Next Generation Windows Services"). Esta iniciativa consistía en una plataforma de amplia distribución y basada en Internet diseñada para contener nuevas herramientas de desarrollo, servicios en tiempo de ejecución, características de sistema operativo, servidores y protocolos de Internet.

El código utilizado para el CLR fue originalmente hecho por Colusa Software para su OmniVM. Esta empresa fue comprada por Microsoft en 1996.

Los objetivos principales que buscaba .NET en este inicio eran:

1. Reforzar el desarrollo y distribución simplificado de servicios basados en Internet.
2. Permitir la creación de nuevas y más poderosas capacidades de transacción de negocio a negocio y de negocio a cliente; y
3. Enriquecer la experiencia del cliente localmente y a través de la red.

Microsoft definió entonces a su plataforma .NET de la siguiente manera: "un cambio de visión, entre sitios de Internet aislados o componentes conectados a Internet, a constelaciones de computadoras, dispositivos y servicios que trabajan de manera conjunta para entregar más y mejores soluciones".

1.1.3. Visual Studio.NET

Pasaron 2 años en los cuales Microsoft se encargó de desarrollar dos productos que serían pieza angular del desarrollo de esta nueva tecnología.

El primero de ellos era el Framework.NET que es la máquina virtual sobre la cual correrían todas las aplicaciones desarrolladas para .NET; y al igual que Java, permitirían tener esa portabilidad y seguridad que tanto se había buscado y que Microsoft no había podido obtener.

El segundo de ellos, que si bien no era lo más importante, sí era lo que le daría la hegemonía sobre esta poderosa tecnología, el Visual Studio.NET.

El 13 de febrero del 2002, dos años después del anuncio de la nueva tecnología que sería creada, Bill Gates (Arquitecto de Software en Jefe de Microsoft), anunció la liberación del Visual Studio .NET y del Framework .NET que a partir de ese momento se volvía la piedra angular del desarrollo de software para esta compañía.

En las palabras del mismo Bill Gates: "*Visual Studio.NET y el Framework .NET son dos de los más importantes productos que Microsoft haya liberado y demuestran el compromiso de Microsoft con la comunidad a largo plazo. Es el primer ambiente verdaderamente integrado para desarrollo de servicios Web XML y para las próximas generaciones de aplicaciones sobre Internet. Éste permitirá una nueva gran ola de*

oportunidades para los desarrolladores y permitirá que pronto XML se vuelva la base para todo el desarrollo de software." (Figura 1.1)



Fig 1.1 Alcances del Visual Studio .NET

1.1.4. .NET en Crecimiento

El lanzamiento oficial del IDE (Entorno de Desarrollo Integrado) y de la plataforma .NET era el último paso para que Microsoft retomara el liderazgo en cuanto a lenguajes de programación se refiere. Desde el auge de Visual BASIC y Visual C++, se había perdido esta hegemonía. Sin embargo, este nuevo entorno de desarrollo, permite que los números hablen por si solos:

- 250 000 desarrolladores recibieron entrenamiento a través de 200 cursos en línea.
- 190 herramientas se lanzan en conjunto con el Visual Studio.NET
- Más de 200 libros de .NET
- La "MSDN Academic Alliance" incluye 1 500 departamentos miembros de carreras de computación en 815 instituciones, alcanzando cerca de 1 500 facultades y con ello 200 000 estudiantes.

1.1.5. Visual Studio 2003

El 24 de abril del 2003, Microsoft lanza en conjunto con Windows 2003 Server y con SQL Server 2003, la actualización del más reciente y poderoso IDE. Esta actualización, se pone al corriente con el Framework y con las nuevas necesidades que han surgido debido al éxito de ADO.NET.

Lo que más resalta a la vista es que este trío, viene enfocado al desarrollo en .NET pues no sólo esta versión de Windows es la primera en traer ya precargado el Framework .NET, sino que SQL Server viene con la compatibilidad total para ADO.NET

Aunque en sí, .NET no sufre demasiados cambios (el lenguaje que sigue en evolución es BASIC.NET) se tiene un IDE mucho más humano, amigable y con mejores ayudas, lo que lo hacen por mucho el mejor entorno de desarrollo.

1.2. Framework .NET

1.2.1. ¿Qué es .NET?

Microsoft .NET es una infraestructura prefabricada para resolver problemas comunes de aplicaciones de Internet. Así lo define Microsoft, sin embargo es necesario dar una definición más precisa, y ésto sólo se puede lograr definiendo las partes de las cuales está constituido:

a) Framework .NET (Ambiente de Ejecución)

El Framework .NET es el componente para crear y ejecutar la próxima generación de aplicaciones de software y servicios Web XML. Se encarga de la mayor parte de la estructura necesaria para generar software, lo que permite a los programadores centrarse en el código lógico esencial para el sistema. Facilita más que nunca la creación, implementación y administración de aplicaciones seguras, sólidas y de gran rendimiento. Windows .NET Framework se compone de Common Language Runtime y un conjunto unificado de bibliotecas de clases.

b) Active Server Pages .NET (Páginas de Servicio Activo .NET)

ASP.NET es un nuevo ambiente que se ejecuta sobre IIS (Internet Information Services) y está diseñado para que los programadores puedan programar código que construya documentos HTML para que puedan ser vistos con el explorador.

ASP.NET proporciona un lenguaje nuevo e independiente que ayuda a lograr esta función y sobre todo logra manejar con eficiencia las peticiones recibidas por la página Web. Una nueva característica que le da todo el poder es la inclusión de las llamadas WbForms, que son un modelo de programación orientada a eventos. Estas WebForms buscan que la programación para Internet sea muy similar a la programación de aplicaciones para escritorio.

c) Web Services .NET (Servicios de Internet XML)

Se define como una nueva forma para que los servidores de Internet muestren sus funciones a cualquier cliente. Se prevé que en un futuro Internet ya no sea como en nuestros días, sino que los programas clientes (como Ares, Copernic, etc) estén dedicados a buscar y obtener información de Internet, utilizando su propia interfaz para desplegar sus resultados. El Framework .NET provee servicios que permiten al servidor Web mostrar sus resultados sin importar el tipo de sistema operativo que se esté

utilizando. El cliente hace llamadas al servidor utilizando una combinación de componentes XML (por ejemplo HTML) y HTTP; a esta combinación se le llama Servicio Web.

Los Servicios Web son la más innovadora tecnología para los negocios en la Web; tecnologías programables y reutilizables que aprovechan la flexibilidad de Internet. Con ellos es posible tener una infinidad de aplicaciones conectadas en red, ya sea que se ejecuten en diferentes plataformas, proporcionando información a todos sus clientes, socios de negocios y empleados. La mejor manera de desarrollar e implementar Servicios Web XML es a través de software y herramientas de desarrollo Microsoft .NET.

Los Servicios Web tienen como base un conjunto de estándares abiertos, como (XML, SOAP, WSDL y UDDI), los cuales son controlados por el World Wide Web Consortium (W3C). Trabajar con .NET significa usar protocolos abiertos que unen sistemas y aplicaciones existentes, permitiendo aprovechar mejor todos los beneficios que ofrecen. La información de la empresa aparece como una entidad única, integrada y fácil de compartir con otras empresas.

d) Windows Forms (Formularios de Windows)

El regalo de BASIC a la comunidad desarrolladora, es sin lugar a dudas los formularios de Windows. Las interfaces siempre han sido de vital importancia para que un sistema tenga amplia aceptación y .NET nos regala una versión mejorada sustancialmente de las formas utilizadas en Visual BASIC 6, que como ahora está contenida en el framework, está disponible para todos los lenguajes sin perder ninguna de sus propiedades.

e) ActiveX Data Object .NET (Objetos de Datos ActiveX .NET)

ADO permitió tener un gestor de conectividad con cualquier tipo de base de datos soportado por el Visual Studio. Hoy en día, ADO.NET permite tener una conectividad muy superior, pues no sólo tiene la ventaja de trabajar con casi cualquier base de datos utilizada (Oracle 10g, SQL Server, etc) con los comandos básicos del Framework, sino que además está preparada para migar a Internet en cualquier momento, por lo que no estamos hablando de un conector más, sino de una herramienta que es la más poderosa hasta el momento.

1.2.2. El Ambiente de Ejecución .NET

Mejor conocido como Framework .NET, es un modelo de programación de Microsoft para desarrollar, implementar y ejecutar servicios Web XML y todos los tipos de aplicaciones. Microsoft persigue dos objetivos principales:

- Mejorar el desarrollo de Microsoft Windows haciéndolo más orientado a objetos y simplificando el modelo de objetos subyacente.
- Ofrecer un marco de trabajo moderno de tercera generación para crear e implementar Servicios Web XML.

Con el Framework .NET, Microsoft mejora enormemente la productividad del programador, la facilidad de desarrollo y la ejecución de aplicaciones confiables.

a) La arquitectura del Framework .NET

Actualmente, el Framework.NET se diseña para funcionar en los sistemas operativos de Microsoft Win32 y Win64. En el futuro, el .NET Framework será extendido al funcionamiento en otras plataformas, tales como Microsoft Windows CE, Linux, etc.

Al funcionar en Windows, los servicios de aplicaciones, tales como Component Services, Message Queuing, Windows Internet Information Server (IIS), y Windows Management Instrumentation (WMI), están disponibles para el programador. El Framework .NET 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.

El Framework .NET (figura 1.2) 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. El Framework .NET 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, el Framework.NET permite la herencia entre lenguajes, el manejo de errores y la depuración.

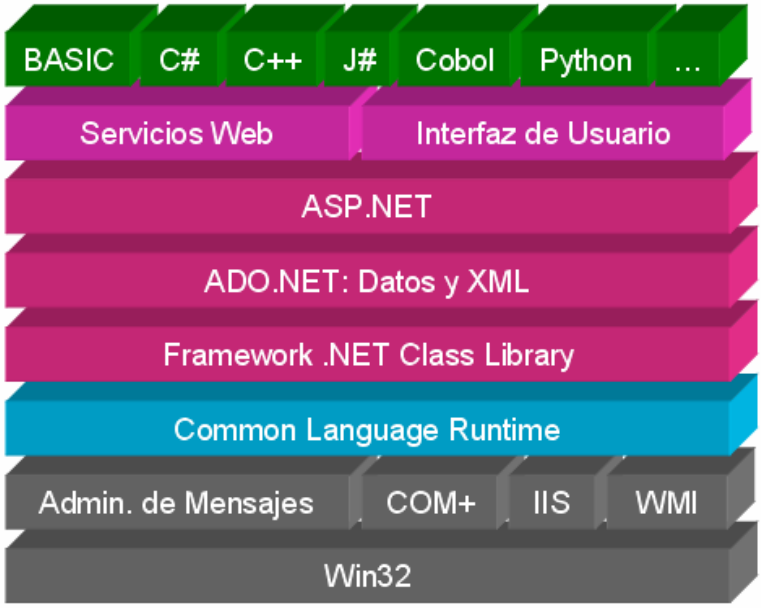


Figura 1.2 Estructura Gráfica de .NET

b) Common Language Runtime

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, los servicios de estos procesos; además de ser el motor de ejecución para las aplicaciones del Framework .NET

Los tiempos de ejecución han acompañado a casi todos los lenguajes de programación de 4ta generación. VB6 y VC++ utilizan el MSVBVM y el MSCVCRT como tiempos de ejecución particulares de estas plataformas.

Puede considerarse como el núcleo del 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 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 CLR desempeña un papel sumamente importante en el desarrollo de un componente durante el tiempo que éste está en 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 las políticas de seguridad y satisfacer cualquier dependencia que el componente pudiera tener con otros componentes.

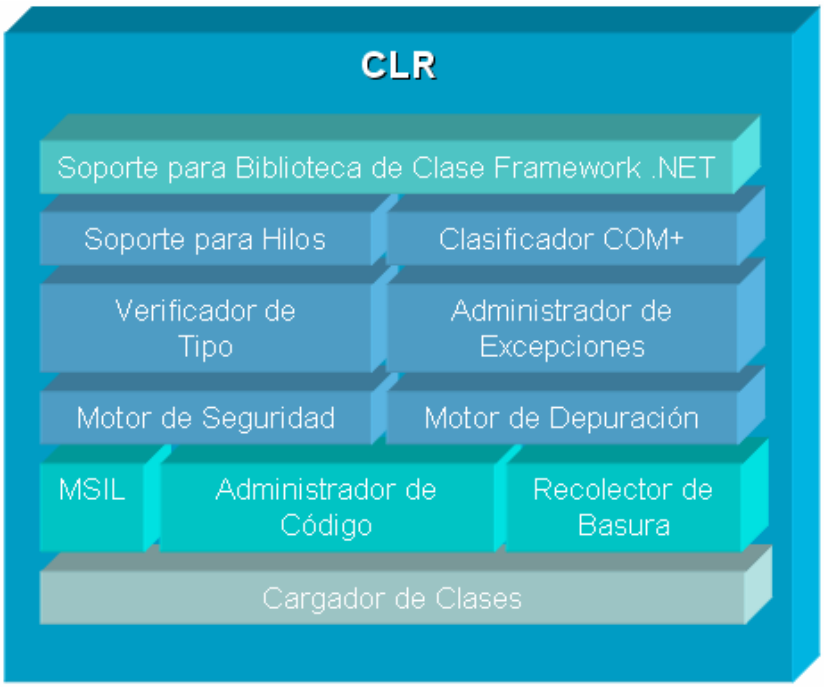


Figura 1.3 Componentes del CLR

En la figura 1.3 vemos los componentes del CLR. Creo importante describirlos todos, sin embargo, los más importantes para el objetivo, así como otras funciones del CLR, serán descritos con mayor amplitud más adelante.

- Cargador de clases.- administra metadatos y la carga y diseño de las clases.
- Administrador de código.- administra la ejecución del código.
- Motor de seguridad.- proporciona seguridad basada en las evidencias, según el origen del código; además de la identidad del código de invocación.
- Motor de depuración.- le permite depurar su aplicación y rastrear la ejecución del código.
- Verificador de tipo.- deshabilita las transmisiones inseguras o variables no inicializadas.
- Administrador de excepciones.- proporciona manejo estructurado de excepciones.
- Soporte para hilos.- proporciona clases e interfaces que permiten la programación multihilada.
- Clasificador COM.- proporciona organización desde y hacia COM.
- Soporte para la Biblioteca de Clase de Framework.NET.- el código con CLR que soporta la Biblioteca de clases de Framework.NET.

c) Objetos del Framework .NET

La extensión del framework .NET impide de definirlo en breve. Por lo mismo, en esta sección hago mención de los elementos más importantes de su estructura, y que nos permitirán desarrollar Software Seguro.

c.1) Compilación y Ejecución

Lo más importante para nuestro objetivo será definir el modo de operación del Framework. Sobre el CLR, correrán nuestros programas que estarán formados por código administrado. El código administrado no es más que código escrito en uno de los 20 lenguajes que están diseñados para explotar las características del Framework.

Lenguajes que ya tienen su versión .NET: Fortran, Pascal, Haskell, Perl, Lenguaje Java, Python, COBOL, RPG, Component, Pascal, Mercury Scheme, Curriculum Mondrian, SmallTalk, Eiffel, Oberon Standard ML, Forth Oz

Una vez que tenemos el código fuente, éste es compilado por un compilador específico por lenguaje (no existe un compilador único). Este compilador de lenguaje, se encarga de generar Código Intermedio de Microsoft (MSIL) que es similar para todos los lenguajes. Es decir que si hacemos un programa que efectúe lo mismo de similar manera en VC# y en VBASIC, ambos al compilar, generarán un código MSIL prácticamente idéntico. Esta característica es cierta para cualquier lenguaje que sea escrito para .NET y se debe a que sólo se usa el lenguaje para dar la estructura lógica, mientras que todas las variables, clases, atributos, etc., son parte de las librerías de clase del Framework.NET.

El MSIL, es el resultado de la compilación y se empaqueta en forma de un archivo EXE por lo que a la vista del usuario es un archivo ejecutable común y corriente. Sin embargo, este EXE lleva empaquetado el código en MSIL y los metadatos correspondientes para su ejecución. Por lo tanto es INDISPENSABLE que el Framework .NET esté presente en el sistema operativo sobre el cual se quiera correr la aplicación.

El siguiente paso es ejecutar nuestro archivo.EXE. Para ello el Framework .NET consta de un compilador "Just In Time" (Jitter o compilador JIT) que como su nombre lo indica, compila el MSIL optimizándolo según sean las necesidades de nuestra plataforma (donde se ejecuta el Framework).

El proceso de Compilación y Ejecución se muestra en la figura 1.4:

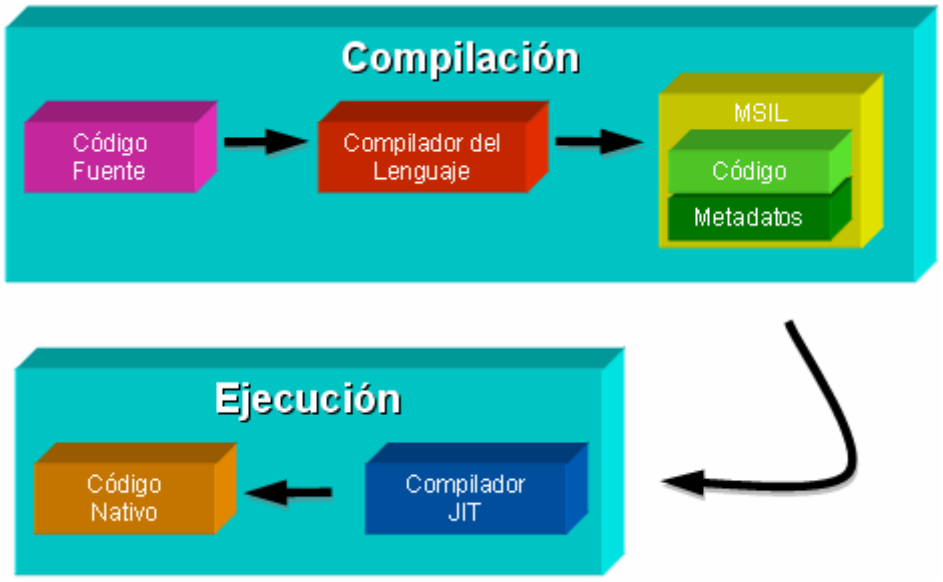


Figura 1.4 Pasos de la Compilación y la Ejecución

c.2) *Librerías de Clase*

Conocidos como "namespaces" son una forma jerárquica de organizar las funciones dentro del Framework .NET. Al referirme a una organización jerárquica quiero decir que cada objeto clase dentro del Framework está contenida en una de mayor generalidad.

Por ejemplo, la función dibujar rectángulo, se puede encontrar en el Namespace System.Drawing (System.Drawing.DrawRectangle) mientras que la función lector de flujo (para archivos) se encuentra en el namespace System.IO (System.IO.StreamReader). Como vemos, esta forma de organización facilita el rastreo de las funciones que queremos encontrar.

c.3) *Ensamblados*

Un ensamblado es un conjunto de uno o más archivos EXE o DLL que contienen información acerca de los recursos de la aplicación así como el código. Cuando creamos un proyecto en Visual Studio .NET se crea un archivo "AssemblyInfo" (figura 1.5) que contiene toda la información acerca de los archivos y versiones necesarias para su ejecución. De esta manera cuando se compila nuestro proyecto, el EXE resultante contiene estos metadatos, que ayudarán en la ejecución del programa.

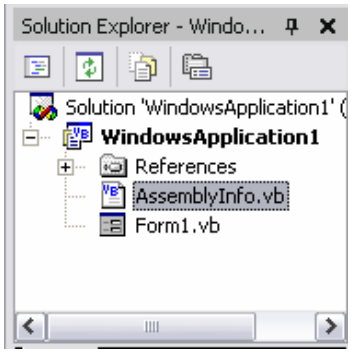


Figura 1.5 Manifiesto

Un DLL (Dynamic Link Library) es un archivo donde el programa ejecutable encontrará todas las herramientas necesarias para su ejecución. Uno de los mayores problemas enfrentados en la ejecución de programas es el llamado "Infierno DLL". La principal causa es la creación de DLL

compartidos, pues en el momento en que alguien necesita modificarlo, lo puede hacer, sin embargo, aquellos otros programas que comparten ese DLL, pueden verse afectados.

Para resolver este problema, .NET establece en sus manifiestos una serie de 4 números llamados "compatibilidad de versión" estos cuatro números son "Versión Mayor.Versión Menor.Revisión.Número de Compilación" (por ejemplo: 7.5.0.0299).

Cuando un cliente ejecuta un programa, el Framework se asegura de encontrar las versiones que la aplicación necesita; si no las encuentra, no se podrá ejecutar el código. Esto en conjunto con la característica que permite tener en el Global Assembly Cache (GAC) diferentes versiones de un mismo ensamblado, asegura, que tendremos siempre las versiones correctas a nuestra disposición (ver figura 1.6).

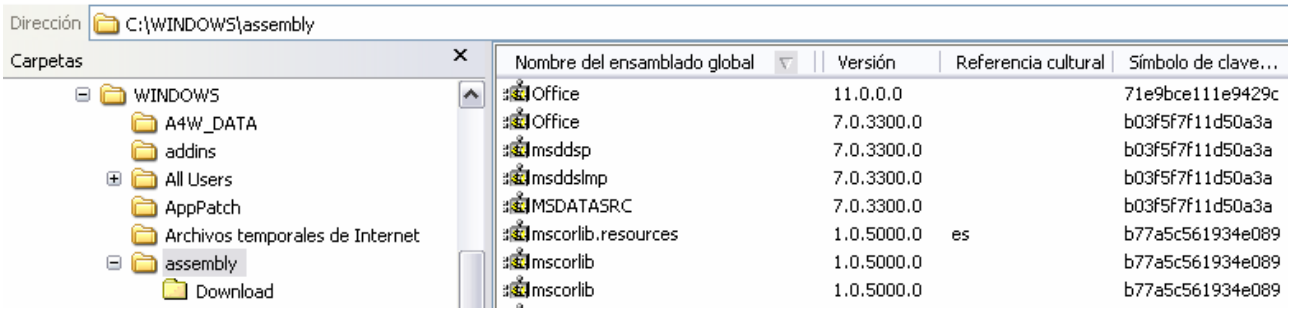


Figura 1.6 En el GAC, el ensamblado Office tiene diferentes versiones

c.4) Programación Orientada a Objetos

Uno de los aspectos más importantes al migrar a .NET es su predeterminada forma de trabajo que es la Programación Orientada a Objetos.

Todos los lenguajes que deseen utilizar las librerías de clase y convertirse así en un lenguaje .NET, deben contar con esta característica. Esto es sumamente sencillo, pues el Framework proporciona funcionalidades para que los programadores puedan utilizarlas en todo su potencial.

Debido a esta característica BASIC.NET fue el lenguaje que más resintió este cambio, pero a la vez, fue el que más se benefició, pues con todas las mejoras que ésto incluye, se volvió un lenguaje que si bien es sencillo, también se vuelve un lenguaje para desarrolladores serios.

El Framework.NET nos proporciona todo lo necesario para establecer una POO completa, dando soporte a características como herencia, constructores, clases base y clases abstractas, polimorfismo, encapsulamiento, etc.

c.5) Recolector de Basura

Uno de los principales problemas en la programación (y en si en la computación) es el correcto uso de la memoria. El hecho de colocar en memoria una variable nos acarrea la responsabilidad de removerla después de su utilización. En lenguajes como C++ es

muy conocido el comando malloc() para generar un espacio en memoria, así como delete() para quitarlo. Ésto acarreaba grandes problemas de manejo de memoria, pues el hecho de que a algún programador se le olvidara quitar de la memoria alguna variable, podía acarrear problemas de saturación de la misma. Aunado a este problema, está el hecho de que muchas veces se perdía más tiempo elaborando un correcto manejador de memoria, que en el mismo problema a resolver.

Por ello, era de vital importancia liberar al programador de esta tarea, devolviéndolo a su principal objetivo que es el de programar. Para ello el "Garbage Collector" fue la solución. El GC se encarga de recoger, limpiar y redireccionar toda la memoria durante la aplicación en ejecución.

Una vez que se crea una variable dentro de la memoria, ésta es alojada ahí, y mientras esté en uso o esté referida en algún lugar del código, será direccionada. En caso de que la variable pierda referencias (es decir, rebasamos su visibilidad) se queda en la memoria, esperando por el colector de basura. Cada determinado tiempo, el colector pasa, y verifica las referencias de cada objeto dentro de la memoria y si encuentra algunos sin referencia los recolecta; además de mover las variables para hacer grupos más grandes de memoria continua.

El recolector de basura no tiene un comportamiento ni un algoritmo de conocimiento público que nos diga a ciencia cierta cada cuando pasa por lo que esta característica lo hace muy poderoso y seguro.

c.6) Manejador de Errores

La misión es simple: evitar que los programas terminen inesperadamente. Tan sencillo es enunciarlo como difícil es resolverlo. Hasta antes de .NET se utilizaban diferentes soluciones, como regresar un valor incorrecto o nulo de una función en C, o el OnError Goto de BASIC. Sin embargo ninguno realmente atrapaba el error, permitiendo que este se propagara a través de nuestro programa. Java y C++ utilizaban un manejador de errores, pero sus limitaciones eran las del mismo lenguaje.

Es así como el "Structured Error Handler" (SEH) nace en el Framework .NET. En el framework, cada vez que ocurre un error en tiempo de ejecución, el CLR lo detecta y arroja una excepción con la información específica del error. Mediante los bloques Try Catch se puede "atrapar" este error y darle el tratamiento adecuado según sea la naturaleza del mismo. Por lo que tenemos un verdadero manejador de errores que atrapa el problema en su fuente y evita que se propague.

El modo de operación del SEH lo podemos apreciar en la Figura 1.7:

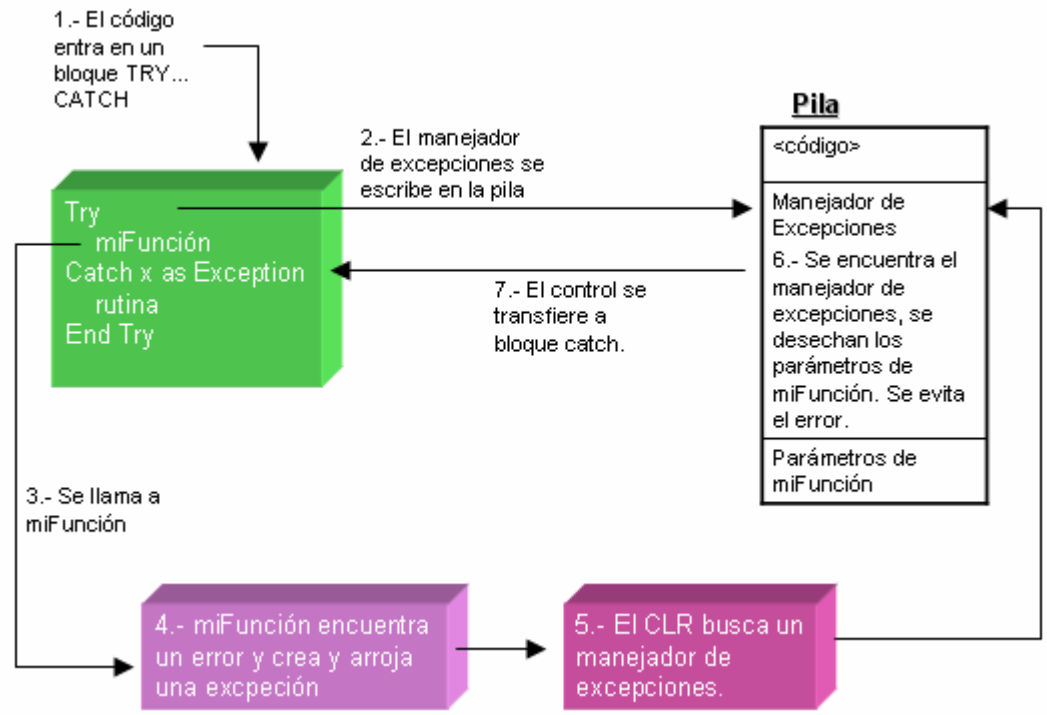


Figura 1.7 Rutina para atrapar errores

Al entrar el programa (1) en un bloque try catch, todo lo que esté ubicado en él será atrapado por un manejador de errores. Para atraparlo, se escribe en la pila un SEH, que será responsable de ese bloque (2). Se continúa con la ejecución (3) y en este caso miFunción detecta un error y genera la excepción (4) que será manejada por un SEH. El CLR, al detectar la excepción (5) busca al manejador; en caso de no encontrarlo produce un error de tiempo de ejecución. En este ejemplo, encuentra al manejador (6) y el manejador se encarga de proteger a la pila, remover todo lo que se había almacenado (incluso a ella misma) y le transfiere el control al bloque catch (7) para que se tomen las medidas pertinentes.

De esta manera tenemos un SEH que es muy capaz y funcional. Con ello ya no hay pretextos para tener errores de tiempo de ejecución, y así al menos podremos avisarle al usuario que ha ocurrido un error y manejar sus datos de manera adecuada (guardarlos, cancelarlos, etc.) y salir del programa debidamente.

c.7) Seguridad de Acceso al Código

La plataforma .NET nos ofrece un modelo de seguridad basado en roles, para definir quién y quién no puede ejecutar cierto código y con que privilegios.

Esto es, dependiendo de los privilegios del código, se irán verificando con cada uno de los roles predefinidos y cuando se llegue al rol que permite la ejecución de este código, entonces tendrá definido sus recursos y sus alcances. Esto lo logra a través del ensamblado, donde podrá guardar a través del manifiesto, los permisos que podrá tener.

Esto asegura que no se ejecute código que pueda afectarnos y queda completamente en manos de los usuarios su ejecución, por lo que si un usuario quiere ejecutar código que excede sus privilegios, éste jamás será ejecutado.

En la figura 1.8 podemos ver como un permiso "P" busca en cada ensamblado los permisos necesarios (G) y hasta que lo encuentra, se detiene (E4).

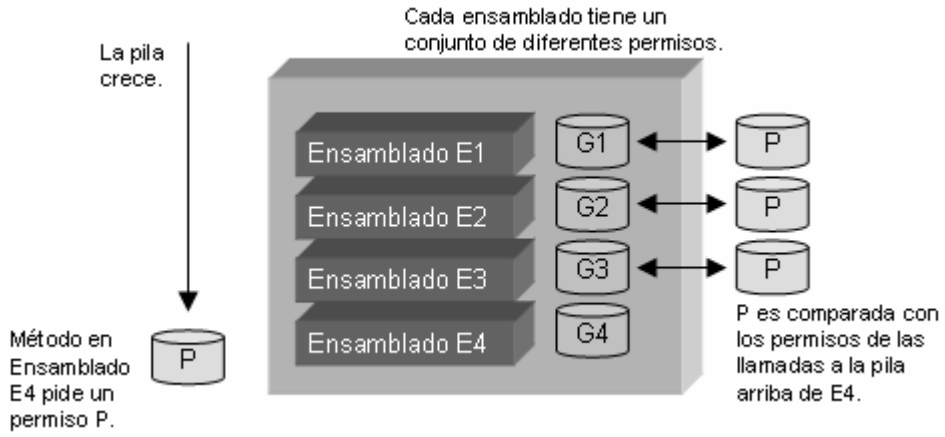


Figura 1.8 Rutina para búsqueda de acceso al código.

c.8) Costo

La segunda ley de David Platt nos dice que "La cantidad de basura permanece constante". Esto es verdad para el software, y esto lo sabemos si queremos mayor precisión, perdemos rapidez y viceversa (esto lo hemos aprendido a través de los diferentes cursos que hemos tomado).

Por lo mismo, si nosotros ganamos toda esta seguridad, manejo de memoria, organización de funciones, etc. ¿Dónde queda toda esa basura que hemos desechado? La respuesta es sencilla, en el hardware y en Sistema Operativo.

De ahora en adelante, los sistemas operativos tendrán que lidiar con más procesos, con más rutinas, con más ciclo y ésto a su vez, se verá reflejado en mayor uso de recursos (procesador, memoria, etc.) por lo mismo, vemos que la segunda ley de Platt es toda una verdad para el software, de hecho, la primera ley de Platt establece que "No importa que tan bueno sea el hardware, los chicos del software siempre lo echarán a perder", aunque suene a chiste, es una verdad absoluta.

1.3. Perspectivas

El ambiente .NET está diseñado para permitir la estandarización y la alta portabilidad de las comunicaciones en Internet y permite que los servidores, computadoras, y diversos dispositivos puedan intercambiar y compartir información fácilmente a través de la red. Se espera que en un futuro, cualquier persona pueda tener acceso a toda su información sin importar en donde se encuentre a través de los dispositivos móviles que hoy están en pleno desarrollo.

Microsoft espera que .NET pueda cumplir con estas expectativas. De hecho ellos dicen que: ".NET permite ir más allá de las ideas concebidas por Internet y los sistemas operativas, haciendo que Internet sea el sistema operativo de todas las computadoras".

Incluso AT&T se había adelantado a esta idea con su "Networking Computing" en 1990, sin embargo hasta el nacimiento de .NET se logró una manera real de cumplir este sueño. Lo único que sí es seguro es que Microsoft apuesta todo a esta plataforma, que de ahora en adelante será parte de todos los sistemas de Microsoft.

1.3.1. Problemas legales

En un aparente intento de evitar nuevos problemas legales, Microsoft dijo que ofrecería tanto versiones en red como estándar del nuevo software. La integración de la tecnología de explorador dentro de Windows es un tema central en la demanda antimonopolio que emprendió el gobierno de EE.UU. en contra de Microsoft.

Sin embargo, el plan de Microsoft .Net dará a los reguladores antimonopolio mucho en qué pensar. Muchas de las funciones computacionales que actualmente son productos que se venden por separado ofrecidos por compañías de software podrían potencialmente ser desplazadas por versiones Microsoft. Esto va desde calendarios personalizados hasta funciones de búsqueda que serán servicios que formarán parte de Microsoft .Net

1.3.2. Implementaciones alternativas

Mientras que el framework .NET es el líder actual en tecnologías, existen otras implementaciones que bien podrían ayudar a que la plataforma sea la más utilizada.

Mono es una implementación en código abierto del Framework .NET que está siendo desarrollada por Ximian (parte de Novell Inc.) y la comunidad del Software Libre. Su maduración es rápida y tiene soporte para ASP.NET y ADO.NET. Incluye un compilador de Visual C# y de Visual BASIC.

El Rotor de Microsoft o el Shared Source Common Language Infrastructure es una implementación del Framework y está diseñado para correr en casi cualquier plataforma como Microsoft Windows XP, FreeBSD, and Mac OS X 10.2. Sin embargo el éxito de la

tecnología .NET no sólo son las cualidades de la plataforma, sino de su amigable ambiente de desarrollo, el Visual Studio, que muy difícilmente podrá ser proporcionado para el código abierto.

1.3.3. Estandarización

Microsoft ha mandado parte de las especificaciones de .NET a ECMA y a ISO para su estandarización. Esta es una excelente forma que ha planeado Microsoft para que se establezca un puente entre el software de Microsoft y el software libre. Por lo mismo, el poder que puede lograr .NET es verdaderamente impactante y muy probablemente se pueda lograr la estandarización que tanto se ha buscado a través de los tiempos.

1.3.4. Visual Studio 2005

Con fecha de lanzamiento de 7 de noviembre del 2005, el Visual Studio .NET 2005 es toda una realidad.

Se ve un acercamiento mayor al usuario, para hacer su estancia y tiempo de programación más productivo y mucho más amigable y disfrutable. Por lo mismo el intelli sense será mejorado y la vista por supuesto tendrá un mejor y más organizado estilo.

En cuanto a los lenguajes, se prevé un mejor desempeño del diseño Web así como un ASP.NET 2.0 mejorado para evitar aún más los desastrosos errores de configuración.

2. Visual BASIC.NET

Desde sus inicios el lenguaje de programación BASIC (**B**eginners **A**ll-purpose **S**tandard **I**nformation **C**ode) fue diseñado para ser el punto de lanzamiento de todos los programadores, pues su simplicidad permitía tener un acercamiento amigable a la computación sin tener que lidiar con problemas fuera de lo que era la programación.

Es así como BASIC surge al mercado, y desde su nacimiento en 1964 a la fecha ha habido diferentes versiones, así como intentos por volverlo un lenguaje que deje de ser para principiantes y llegue a ser un lenguaje 100% efectivo y digno de uso para aplicaciones serias.

De esta manera BASIC lucha por deshacerse de un estigma que lo califica como un lenguaje de "juguete" o "para principiantes" y trata de ganarse el favor de aquellos que ahora le tienen desconfianza.

Desde que Microsoft tomó el liderazgo en el desarrollo de este lenguaje, sobre todo después del lanzamiento de Visual BASIC, el lenguaje ha tomado un nuevo giro, y ha ganado miles de adeptos. Ésto se debe a que el lenguaje se vuelve cada día más poderoso sin perder de vista la sencillez que lo caracteriza.

En últimas fechas, Microsoft por medio de su plataforma .NET, ha puesto en manos de los desarrolladores una herramienta que rebasa todas las expectativas, pues nos permite tener a nuestro alcance un nuevo y reestructurado lenguaje de programación.

Debido a su importancia y trascendencia (y a los cerca de 2 millones de usuarios que la utilizan) el conocimiento del lenguaje es muy importante, pero sobre todo debemos saber dónde nos encontramos parados para hacer una excelente y propia explotación del mismo.

2.1. De BASIC a BASIC.NET

2.1.1. Inicios

El lenguaje BASIC (Código de Instrucciones Simbólicas Multi-Propósito para Principiantes) nació en 1964 en la Universidad Dartmouth en New Hampshire (E. U. A.), dónde fue desarrollado por John G. Kemeney (1926-1993) (fig. 2.1) y Thomas E. Kurtz (1928-actualmente consultor de Microsoft) (fig. 2.1). Kemeney, trabajó primero en el Proyecto Manhattan (1945) y más tarde (1948-1949) como ayudante de Albert Einstein. Así fue como conoció a Kurtz (figura 2.2) en Dartmouth en 1956.



Fig 2.1
John G. Kemeney



Fig2.2
Thomas E. Kurtz

Ambos empezaron a trabajar en un nuevo lenguaje de programación simplificado, y después de las llamadas versiones *Darsimco* y *DOPE*, cambiaron hacia un lenguaje con las especificaciones siguientes:

1. Ser sencillo para principiantes
2. De propósito general
3. Permitir a expertos anexar nuevas instrucciones (sin alterar su simplicidad).
4. Interactivo
5. Proveer mensajes de error amigables y específicos
6. Responder rápido ante programas sencillos.
7. No requerir conocimiento del hardware local
8. Separar al usuario del sistema operativo.

Para lograrlo empezaron por FORTRAN y ALGOL. Kemeney y Kurtz lograron el éxito el 1 de mayo de 1964 a las 4 de la mañana, cuando dos programas BASIC corrieron simultáneamente en el 225 UC de General Electric de la Universidad Darmouth. La versión final se llamó "Dartmouth BASIC" y venía con 14 instrucciones.

El primer BASIC que fue considerado un lenguaje completo, fue el "Tiny BASIC" desarrollado por Li Chen Wang's y que fue implementado para un microprocesador

Kemeney y Kurtz no protegieron su invención con una patente. Esto permitió su crecimiento y también el incremento del número de versiones. Es precisamente en este momento donde inicia su mala fama, pues era idóneo para la programación caótica, no contenía estructuras rígidas como ocurría con Cobol, la falta de estructuras de datos, inexistencia en declaración de variables, un manejo poco común de matrices, en cuanto a su visibilidad (global, pública y privada), era el propio intérprete el encargado de manejarlas. Evidentemente era lo menos parecido a un lenguaje estructurado.

2.1.2. Expansión

Desafortunadamente, la falta de estandarización condujo a versiones diferentes, la mayoría incompatibles entre si, es decir que no nos entregaba la portabilidad que podría ser deseada en un lenguaje que está destinado a liderar el mercado; y ésto se agudiza, pues se podría decir que hay tantas versiones como intérpretes.

En 1974 apareció el CBASIC y muchas otras versiones que condujeron a la ANSI a definir los estándares. La norma que ANSI estableció para el lenguaje no sirvió mas que para crear unas líneas maestras básicas, de tal forma que el parecido entre los diferentes intérpretes no fue más que una excelente idea que no fue plasmada hasta 1978, tiempo en que BASIC ya se había difundido.

Normas ANSI para BASIC r

- **ANSI Standard for Minimal BASIC (X3.60-1978)**
- **ANSI Standard for Full BASIC (X3.113-1987)**
- **ISO Standard for Minimal BASIC (ISO 6373:1984 Data processing - Programming languages - Minimal BASIC)**
- **ISO Standard for Full BASIC (ISO/IEC 10279:1991 Information technology - Programming languages - Full BASIC)**

Según algunos especialistas, no fue sino hasta la llegada de los CP/M (Control Program for Microcomputers) que comenzó una progresión ascendente.

Lo más interesante de esta primera etapa es que aunque apareció más tarde que otros lenguajes en el ámbito computacional, BASIC rápidamente se extendió a muchos sistemas no Unix como un sustituto a los lenguajes de scripts encontrados nativamente en Unix.

Esta es probablemente la razón más importante por la cual este lenguaje es poco usado por la gente de Unix, pues éste tenía un lenguaje de scripts mucho más potente desde su aparición.

2.1.3. Microsoft Quick BASIC

En 1970 cuando la computadora personal del Micro Instrumentation and Telemetry Systems (MITS) Altair estaba siendo creada, Paul Allen convenció a su buen amigo William Gates de ayudarlo a desarrollar un lenguaje BASIC para ella. El MITS mostró interés en este proyecto, y en poco tiempo concesionaron el uso de BASIC a la MITS Altair 8800. Esta versión pesaba un total de 4Kb incluyendo código y memoria en tiempo de ejecución.

Debido al éxito sin precedentes del lenguaje, pronto se requirió que BASIC fuera escrito para diferentes plataformas como Apple, Commodore y Atari. En 1975 Microsoft se funda en Albuquerque, Nuevo México, sin embargo no pasa mucho tiempo antes que Gates y Allen decidan llevar su compañía a Redmon, Seattle. Para finales de los 70, el siguiente producto de Microsoft fue un sistema operativo que fue pensado para el desarrollo de BASIC, por lo que incluía un intérprete para este lenguaje.

La versión IBM-DOS de este intérprete se conocía como BASIC-A que era esencialmente el mismo a Gee-Witz(GW) –BASIC que era en ese tiempo, el BASIC más conocido y aceptado. La IBM era el principal distribuidor de computadoras personales y estas incluían el sistema operativo de MS-DOS (Microsoft Disc Operating System).

MS-DOS contaba con un intérprete de BASIC, sin embargo, se incluía el lenguaje GW-BASIC. Microsoft al percatarse del éxito con el intérprete que ellos habían diseñado, decide distribuir un compilador de BASIC, para que los programas pudieran ejecutarse sin necesidad del intérprete. Quick BASIC (figura 2.3) fue la solución que Microsoft ideó.

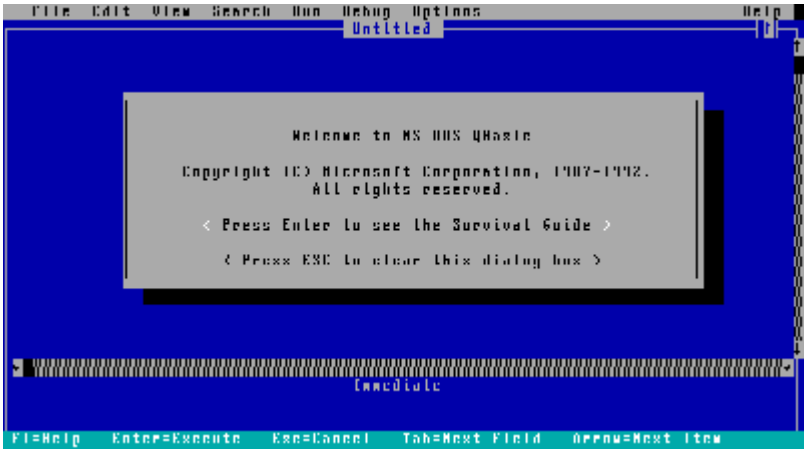


Fig 2.3 Pantalla de QBASIC

Este compilador era distribuido en todos los MS-DOS, y llegó a la versión 4.5. En este tiempo Microsoft decide aumentar las características que BASIC ofrecía al usuario y empieza la distribución de Professional Development System (PDS) BASIC que termina con la versión 7.1 (también conocida como QBASIC Extended); sin embargo esta idea no proliferó.

A pesar del dominio de Microsoft en el mercado, hubo algunas otras aportaciones como el Turbo BASIC de Borland o el Power BASIC de Robert S. Zale que son compiladores muy poderosos, pero que quedan muy por detrás de la siguiente generación de BASIC "Visual BASIC".

2.1.4. Visual BASIC

Con la aparición del entorno gráfico Windows, la primera metodología de programación fueron las llamadas a funciones API (Application Programming Interface) de Windows, esta tarea como su propio nombre lo indica, era una tarea tediosa y difícil de llevar adelante.

BASIC fue el primer producto vendido por Microsoft, También fue el primer caso de piratería masiva. Bill Gates perdió una copia durante un evento de presentación del producto.

El único entorno de desarrollo que existía era por excelencia un compilador de C, pero a medida que Windows fue extendiéndose era necesario la creación una herramienta que

permitiera la explotación sistemática y a fondo del entorno grafico.

En medio de esta necesidad es cuando aparecen los denominados marcos de aplicaciones (Application FrameWorks) tales como MFC (Microsoft Fundation Classes), también aparecen términos como el RAD (Rapid Application Development), pero sin duda el que más rápido fue aceptado, y que fue acuñado por Microsoft en 1991, fue la primera versión de Visual BASIC.

La primera versión de Visual BASIC (la 1.0), vino de la mano de Alan Cooper (figura 2.4). Esta aparición marcó un quiebre en las herramientas de programación visual. Visual BASIC fue la primer herramienta que tenía un entorno de desarrollo visual de cuarta generación, además integraba un editor tradicional, un compilador BASIC y un depurador con un modelo de programación dirigido por eventos.



Fig 2.4 Alan Cooper

De esta manera aparece una nueva forma de programar, basada no en un fragmento de código que debe asociarse a una interfaz grafica, sino centrado en lo que se ve finalmente, o sea el entorno grafico, en el que se programa.

Con la introducción de Visual BASIC 1.0 en mayo de 1991, los programadores podían, por primera vez, implementar aplicaciones de Windows en un ambiente intuitivo y gráfico, simplemente arrastrando controles sobre un formulario.

Visual BASIC 1.0 significó un impacto en la industria informática tan profundo que cambió para siempre el curso del desarrollo del software y creó una explosión en el mercado de las aplicaciones de Windows. Gracias a su puesta en práctica, implementación y capacidades, Visual BASIC 1.0 se propagó a través de la comunidad en cuestión de pocos meses.

Poco después de la fiebre inicial por Visual BASIC, un pequeño pero fuerte grupo de seguidores comenzó a transformar las bibliotecas de código que tenían con sus características, métodos y eventos, y a exponerlos como componentes de Visual BASIC llamados VBXs, o los controles personalizados.

Después de poco tiempo, la producción de estos componentes reutilizables ayudó a Visual BASIC a pasar de ser un logro de software a convertirse en un descubrimiento tecnológico.

A medida que la demanda de Visual BASIC aumentaba, quedaba claro que los desarrolladores requerirían un entorno de desarrollo y un lenguaje mejor y más capacitado. Para tratar a esta necesidad creciente, Microsoft anunció la disponibilidad de Visual BASIC 2.0 en noviembre de 1992. La segunda versión de Visual BASIC proveía a los desarrolladores un funcionamiento perceptiblemente mejorado y mayor capacidad para crear aplicaciones de tamaño mayor y más sofisticadas. Incluía también una ayuda para mejorar la puesta a punto y depuración, proveía de la capacidad de conectarse a bases de datos mediante ODBC, y nuevas y productivas herramientas.

Mientras la adopción de Visual BASIC en las corporaciones se expandía, también lo hacía la necesidad de una herramienta que permitiera aplicaciones dataware robustas. Visual BASIC 3.0, solucionaba esta necesidad combinando el motor de la base de datos de Microsoft Access 1.1 con un conjunto rico de controles dataware. Esto marca un hit, pues por primera vez, los desarrolladores podían conectar fácilmente a las bases de datos en un ambiente cliente/servidor usando un diseñador visual intuitivo.

Posteriormente la industria informática comenzaría a abrazar el movimiento de la programación en 32-bits. La salida al mercado de Microsoft Windows 95 y de Microsoft Windows NT condujeron a esta adopción y destacó la necesidad de herramientas de desarrollo más potentes que podrían soportar la nueva arquitectura. La versión 32-bit fue nombrada Visual BASIC 4.0 (septiembre de 1995).

Las versiones de Visual BASIC 5.0 (marzo de 1997) y 6.0 (junio de 1998), representaron un paso importante pues por primera vez se volvía realidad la programación para Internet; características tales como el compilador del código nativo introdujeron aumentos del funcionamiento de hasta el 2000 por ciento.

Visual BASIC 6 fue introducido al mundo en 1998 y fue incluido como parte de un suit conocido como Visual Studio que también incluía al ya famoso Visual C++. Visual BASIC 6.0 incluía nuevas características en áreas como el acceso a datos, Internet, controles, creación de componentes, etc.

Tal fue su éxito que durante más de 6 años no fue necesaria una actualización que realmente marcara una diferencia total que hiciera surgir una nueva versión. Por lo tanto, durante ese tiempo VB6 logró tener más de 2 millones de programadores en el mundo siendo así el primer lenguaje más usado en el mundo computacional (aún más que C).

Versiones de Visual BASIC

Version 1 (para Windows) – Mayo 20 de 1991

Version 1 (for MS-DOS) – Septiembre de 1992

Version 2 – Noviembre de 1992

Version 3 – 1993

VBA (VB para aplicaciones) 1993

Version 4 – 1995 – soporte para 16- y 32-bits

Version 5 – 1997 – se deja de lado los 16 bits.

Version 6 – 1998 (parte de Visual Studio).

Microsoft durante estos 6 años pasó por diversos y radicales cambios en cuanto a diseño y visión de sistemas. Todos estos cambios se dejan ver en la nueva versión Visual BASIC 7.0 mejor conocido como Visual BASIC.NET.

2.2. Visual BASIC.NET

2.2.1. Historia

Visual BASIC.NET fue concebido como parte de la iniciativa conocida como Microsoft Framework .NET y fue distribuida como parte del Visual Studio .NET. El Framework .NET se produjo con el propósito de producir aplicaciones basadas en XML para el ambiente de Internet de Microsoft.

Además, es un pilar del Framework .NET y es un nuevo paso en la evolución de este lenguaje. Es un lenguaje de programación de alto nivel y además es la entrada más accesible al mundo de .NET.

Visual BASIC.NET fue desarrollado apegándose a estos principios:

1. Este es el descendente de Visual BASIC. Un programador de VB se sentirá familiar con el lenguaje.
2. La sintaxis y la semántica son simples, objetivas y fáciles de entender. El lenguaje desaprueba instrucciones no intuitivas.
3. Le da a los programadores todos los servicios del Framework.NET y lo utiliza sin modificaciones.
4. Es razonablemente parecido a VB6.0.
5. Como el Framework.NET puede utilizar diferentes lenguajes, BASIC.NET trabaja bien en un ambiente multilenguaje.
6. Es compatible con versiones anteriores de VB. Se trató en lo posible de mantener la misma estructura y sintaxis.

El ambiente de desarrollo del Visual Studio así como el Framework impulsó una reestructuración del lenguaje BASIC desde sus cimientos. Microsoft incluyó en esta versión un soporte total para la programación orientada a objetos, así como una integración completa con el CLR.NET (Common Language Runtime). Otro de los cambios más importantes fue la separación del modo en que BASIC desarrollaba las formas de Windows, y la creación de un namespace llamado Windows Forms que podía ser utilizado por todos los lenguajes que utilizaran el Framework.

Para todo aquel programador que migre de BASIC a BASIC.NET resultará conocido el lenguaje, sin embargo el mayor cambio es el que representa la utilización y explotación del Framework, pues el conocimiento de la programación orientada a objetos ahora se ha vuelto determinante. Sin embargo Microsoft realizó un excelente trabajo al mantener la simplicidad del lenguaje.

Los conceptos para programar formas de Windows y el modelo básico de programación orientada a eventos no han cambiado. De hecho sólo ha sido reforzado, lo

que le da mayor estabilidad y control. Sin embargo mucha de la sintaxis y muchos de los nombres y semántica de los objetos, propiedades, métodos y eventos han cambiado. Los tipos de datos han sido ajustados para tener una uniformidad con los lenguajes del Framework. Además de la inclusión del comando STRICT que obliga a declarar correcta y propiamente las variables, así como asegurar su correcta conversión.

Sumado a ésto, BASIC puede utilizar los modelos de última generación referentes a seguridad de software y de instalación y distribución.

Con la adición de la clase Consola para entrada y salida de comandos, Microsoft revive el lenguaje BASIC. Visual BASIC.NET es de hecho un lenguaje orientado a objetos que puede ser programado a nivel consola. Por lo que tenemos un producto que conjunta todas las características del nuevo y viejo mundo, lo que hace que QBASIC ya no sea necesario.

Una segunda versión de VB.NET (VB 7.1 o VB.NET 2003) fue liberada en abril del 2003. Esta versión trae como nuevas características herramientas para programación en dispositivos móviles, mejoras en las características de desarrollo XML, así como soporte para Windows 2003 Server, etc.

2.2.2. Todas las Plataformas Nacen Iguales

Microsoft .NET ofrece una forma orientada a objetos de ver el sistema operativo Windows incluye cientos de clases que encapsulan a las objetos más importantes del kernel de Windows. Por lo tanto casi toda la funcionalidad de los lenguajes está contenida en el Framework.NET y esto hace que para que un lenguaje pueda ser parte de la plataforma, debe explotar sus características. Por ejemplo, si queremos cambiar el tipo de fuente, debemos hacer uso del espacio System.Drawing.Font sin importar el lenguaje que sea.

Microsoft provee varios lenguajes con .NET, incluyendo Visual BASIC, C#, C++, J#, etc. De hecho mucha de la programación que se hacía en Visual C++, ahora se hace en Visual C#.NET, pues el parecido de la sintaxis es similar. Lo más interesante, es que a pesar que muchos denotan a C#.NET como el mejor lenguaje para programación para el Framework.NET, la reestructuración de BASIC ha hecho posible que BASIC.NET sea igual de poderoso que C#.NET. Esto se debe principalmente a que ambos crean más o menos el mismo código en MSIL, que es el lenguaje nativo del Framework.

2.2.3. Sintaxis Básica

Visual BASIC.NET comparte la sintaxis de versiones anteriores de BASIC, por lo que se dará un repaso general de algunas estructuras lógicas. Sin embargo recordemos que esto no es un curso de programación en BASIC.NET

a) Las Variables

Tipo de Dato	Tamaño	Rango
Short	16-bit	-32,768 a 32,767
Integer	32-bit	-2,147,483,648 a 2,147,483,647
Long	64-bit	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
Single	32-bit punto flotante	-3.4028235E38 a 3.4028235E38
Double	64-bit punto flotante	-1.79769313486231E308 a 1.79769313486231E308
Object	32-bit	Para cualquier tipo de dato.
Decimal	128-bit	+/-79,228 1024
Byte	8-bit	0-255
Char	16-bit	Cualquier símbolo Unicode en el rango de 0–65,535
String	16 bits por caracter	De 0 a aproximadamente 2 mil millones de caracteres Unicode
Boolean	16-bit	True o False
Date	64-bit	January 1, 0001, a December 31, 9999

Tabla 2.4 Resumen de variables

La visibilidad de las variables queda especificada por la ubicación de la declaración, así como de la palabra con que es declarada:

- Se utiliza Dim dentro de un procedimiento para declarar una variable local.
- Se utiliza Dim o Private fuera de los bloques de procedimiento para tener acceso a ellas dentro del módulo o la clase donde las declaremos.
- Se utiliza Public dentro de un módulo para generar variables globales.
- Se utiliza Public dentro de un bloque de Clase para declarar un campo público para esa clase.
- Se utiliza Static dentro de un procedimiento para que la variable sea estática y retenga su valor.

La forma para declarar una variable dentro del ambiente BASIC.NET es la siguiente:

[{ Public | Protected | Friend | Protected Friend | Private | Static }] [Shared] [Shadows] [ReadOnly] Dim [WithEvents] *name*[(*boundlist*)] [As [New] *type*] [= *initexpr*]

Ejemplos

Dim estaVariable as String

Static variableEstatica as Short = 0

Private Dim otraVariable as Double

Public a, b, c as Decimal

b) Operadores

Operador	Significado
=	Igual

Operador	Descripción
+	Suma

<>	Diferente
>	Mayor que
<	Menor que
>=	Mayor o igual
<=	Menor o igual

–	Resta
*	Multiplicación
/	División
\	División Entera
Mod	Residuo
^	Exponente
&	Concatención

Tabla 2.5

c) Estructuras de Decisión y de Ciclo

Las estructuras de decisión y de ciclo no cambian en gran medida de las versiones anteriores de BASIC.NET, por lo que sólo se mencionarán los aspectos generales de sintaxis.

La primera estructura de decisión es la sentencia *If*

```
If condición1 Then
        <<declaraciones>>
ElseIf condición2 Then
        <<declaraciones>>
[ElseIf y declaraciones adicionales]
Else
        <<declaraciones>>
End If
```

Ejemplo

```
If variable > 1 then
        variable += 1
ElseIf
        variable -= 1
Else
        variable *= 1
End If
```

Se han añadido los operadores *AndAlso* y *OrElse* para evaluar parcialmente las sentencias If. A estos operadores se les conoce como operadores de cortocircuito. Estas sentencias permiten que se evalúe una condición y luego otra para el caso en que hay múltiples condiciones para una misma sentencia IF.

```
Causa falla
If ValorUno > 0 and ValorDos/ValorUno then
No causa falla
If ValorUno > 0 andAlso ValorDos/ValorUno then
```

```
Causa falla
If ValorUno > 0 or sqrt(ValorUno) then
No causa falla
If ValorUno > 0 orElse sqrt(ValorUno) then
```

La siguiente estructura de decisión es el ya conocido *select case* que es utilizado para realizar estructuras de decisión.

```
Select Case variable
   Case valor 1
      << declaraciones >>
   Case valor2
      << declaraciones >>
   Case valor3
      << declaraciones >>
   ….
   Case Else
```

Ejemplo

```
Select Case variable
   Case 1
      variable += 1
   Case 2
      variable -= 1
   Case Else
```

```
    << declaraciones >>
End Select
```

```
        variable += 1
End Select
```

El ciclo For es la estructura de ciclo más importante de cualquier lenguaje de programación y BASIC presenta una estructura lógica sin cambios con respecto a sus antecesores:

```
For contador [ As datatype ] = inicio To fin [ Step incremento ]
        [ declaraciones ]
        [ EXIT For ]
        [ declaraciones ]
Next [ contador ]
```

Ejemplo

```
For a as integer = 1 to 10 step 2
        arreglo(a) += 1
Next a
```

La última estructura de ciclo, es el ya conocido ciclo Do…Loop que incluye las palabras reservadas, until y while, que permite tener un control total sobre las veces que se realiza el ciclo:

```
Do { While | Until } condición
  [ declaraciones ]
[ Exit Do ]
  [ declaraciones ]
Loop
```

Ejemplos

```
Do While (variable < 0)
  variable -= 1
Loop
```

```
Do
  [ declaraciones ]
[ Exit Do ]
  [ declaraciones ]
Loop { While | Until } condición
```

```
Do
  variable += 1
Loop Until (variable > 1)
```

d) Arreglos y Colecciones

Para declarar un arreglo, se deben considerar la siguiente información:

- Nombre del arreglo.-El nombre que describe el arreglo (Long.Max. De 255 caracteres).
- Tipo de dato.- El tipo de datos que compartirán todos los elementos del arreglo (puede ser object).
- Número de dimensiones.- Los más comunes son de 1(simples), 2(tablas) o 3(cúbicos) dimensiones.
- Número de elementos.- El número de elementos. En VBASIC.NET los arreglos empiezan en 0.
- Valores.- Si se desea se pueden agregar valores entre {} separados por una coma.

Para la creación de arreglos dinámicos el argumento donde se especifica la dimensión del arreglo se deja en blanco para aplicar posteriormente la longitud del arreglo (con la palabra reservada ReDim).

Ejemplo, Declaración de Arreglos

```
Dim miArreglo(9) as long
Dim miArreglo(2,3) as integer
Dim miArreglo(,) as integer={{1,2}, {1,2}}
```

Ejemplo, Declaración de Arreglos Dinámicos

```
Dim miArreglo(,) as Integer
ReDim miArreglo(2,2)
ReDim Preserve miArreglo(2,3)
```

Todos los controles dentro de una forma, forman parte de un arreglo o colección denominado "Colección de Controles". Cada vez que añadimos un objeto a la forma, este objeto se añade a la Colección de Controles (CC).

```
Controls(0).Text = "Hola"
```

Podemos crear colecciones personalizadas con cualquier objeto que tengamos (recordemos que una variable es un objeto).

```
Dim miColeccion As New Collection()
miColeccion.Add(txtTexto)
```

e) Cadenas

En el caso del tratamiento de las cadenas, el .NET Framework y Visual BASIC.NET maneja las cadenas de manera diferente. Mientras que el framework las maneja como objetos, BASIC las maneja como variables; por lo tanto, tenemos métodos para el framework y funciones para BASIC. La única diferencia es el estilo, ya que son equivalentes y nos arrojan los mismos resultados. Sin embargo para unificar se recomienda que se utilice el framework.

```
Dim miCadena as String = "Hola"
```

.NET Framework	BASIC
miCadena.Length -> 4	Len(miCadena) -> 4

Algunas de las funciones y atributos de ambas formas de tratar las cadenas se muestran en la siguiente tabla:

.NET Framework	Visual BASIC	Descripción	.NET Framework	Visual BASIC	Descripción
ToUpper	UCase	Cambia todas a mayúsculas.	IndexOf	InStr	Encuentra la posición de inicio de una cadena en otra cadena.
ToLower	LCase	Cambia todas a minúsculas.	Trim	Trim	Remueve espacios al principio y al final de una cadena.
Length	Len	Determina la cantidad total de caracteres.	Remove	No hay equivalencia	Remueve sub-cadenas de una cadena.
Substring	Mid	Regresa una cadena a partir de un lugar dado. (Nota: El primer elemento de una cadena para .NET es 0.)	Insert	No hay equivalencia	Añade sub-cadenas de una cadena.
			StrComp	No hay equivalencia	Compara dos cadenas (no es sensible a mayus/minus)

Tabla 2.6

f) Manejador de errores

La estructura lógica más importante del Framework.NET es el manejador de errores estructurado. Conocido como bloque Try… Catch, esta estructura sirve para poder reconocer errores no previstos a la hora del diseño.

Al encontrarse VB.NET con un error en tiempo de ejecución, se "arroja" una excepción. Estas excepciones son parte del Objeto Excepción. Se utilizan dos clases dentro este objeto en los programas que son ApplicationException y SystemException.

De hecho la sintaxis es bastante simple, sin embargo, esta es la columna vertebral de cualquier software seguro en Visual BASIC.NET pues toda aquella falla dentro de nuestro sistema será tratada de una manera efectiva por este manejador. Más adelante veremos la forma en como utilizarlo de una manera efectiva.

Se añade la declaración Try justo antes del código que nos pueda preocupar que falle. La declaración Catch sigue inmediatamente de este bloque de código con las instrucciones que queremos que se ejecuten cuando se presente el error. Se finaliza como todo bloque en BASIC, con un End Try.

```
Try

<<declaraciones>>

[ Catch [ exception [ As type ] ]  [ When
expresión ] <<declaraciones>> ]

[ Exit Try ] ...

[ Finally <<declaraciones>> ]

End Try
```

```
Function MiDivision () as Integer

  Dim valorUno as integer = 1
  Dim valorDos as integer = 0

  Try
    MiDivision = valorUno/valorDos
  Catch ex as exception
    Msgbox("Division entre cero")
  End Try

End Function
```

g) Programación Orientada a Objetos

BASIC.NET nos muestra su crecimiento al implementar nuevas características a su poca evolucionada POO. Entre ellas podemos contar a la herencia y constructores. Por lo mismo es importante ver algunas de las sintaxis básicas para generar clases y poder explotar al máximo esta nueva compatibilidad.

g.1) Clases

Las clases se declaran

```
Class <<nombre>>

<<declaraciones>>

End Class
```

```
Class Figuras

Private m_clase as integer = 0

End Class
```

g.2) Propiedades y Métodos

Como se puede ver, las propiedades y los métodos son funciones y subrutinas propias de una clase. Las propiedades regresan y establecen valores, o cualquiera de ambas:

```
Public Property Texto() As String

Public ReadOnly Property Texto() as String

Public WriteOnly Property Texto() as String
```

Su estructura consta de los conocidos Set y Get que se les denomina indexadores y permiten utilizar el encapsulamiento:

```
Public Property Nombre() As String
    Get
        Return <<variable>>
    End Get
    Set(ByVal Value As String)
        <<variable>>
    End Set
End Property
```

De igual manera los métodos son las subrutinas de la clase:

```
Public Sub (ByVal titulo As String)
```

Y realizan operaciones dentro de la clase, para así darle su carácter de estructura completa orientada a objetos.

g.2) Constructores

Los constructores se aplican a la hora de instanciar una clase (crear el objeto). Éstos no son más que métodos sin las cuales un objeto no puede ser creado.

```
Public Sub New(ByVal titulo As String, ByVal texto As String)
```

g.4)Herencia

Para heredar una clase basta con anexar el comando inherits. Con ello se hereda completa la clase base y se pueden utilizar todos sus métodos y propiedades (los constructores no se heredan).

```
Inherits MiClase
```

2.3. Diferencias entre VB6 y VB.NET

Sin lugar a dudas el cambio radical que sufrió Visual BASIC en esta última actualización puede resultar intimidante; sin embargo sólo es la primera impresión, pues la experiencia de haber programado antes en Visual BASIC 6.0 es de gran ayuda ya que se han respetado los principios básicos del lenguaje.

De igual manera para aquel que no haya programado antes en Visual BASIC no le será difícil adentrarse en la plataforma .NET pues BASIC.NET sigue la misma filosofía de ser el lenguaje más accesible para aquel que apenas empieza.

Como se ha mencionado, el lenguaje es el que más ha sufrido cambios para ser parte del Visual Studio, por lo que las diferencias entre ambas versiones deben ser consideradas.

Mencionaré los cambios que a mi parecer son los más importantes, pero sobre todo, mencionaré aquellos que nos servirán para el desarrollo de software seguro. Ésto se debe a que no estamos hablando de un cambio pequeño, sino de una migración total y si quisiéramos tener todos los cambios que se realizaron sería demasiado y eso no es el objetivo de este trabajo.

a) Cambios en las ventanas y en el diseño

En Visual BASIC 6.0, el IDE se establece en diseño MDI de forma predeterminada mientras que en Visual BASIC .NET, el diseño predeterminado del IDE es el nuevo diseño de Organización por fichas.

En Visual BASIC .NET, todas las ventanas son acoplables de forma predeterminada; se puede controlar su comportamiento en el menú Ventana, seleccionando el comando acoplable. (Figura 2.5)

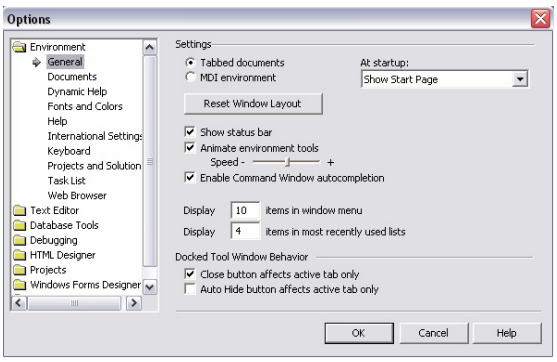


Fig. 2.5 Todas las características del IDE se pueden manejar desde la ventana de opciones.

b) Cambios en la asignación de teclado

Las asignaciones de teclado en Visual Studio .NET están estandarizadas de modo que se pueden utilizar los mismos métodos abreviados de teclado con todos los lenguajes y herramientas. Para tener control sobre las asignaciones de teclado de estas, se pueden establecer desde la ficha Mi Perfil en la Página de inicio de Visual Studio. (Figura 2.6)

En esta ficha también podemos especificar que tipo de desarrollador somos pues podemos ser sólo desarrolladores de un cierto lenguaje, y esto ayudará al IDE a asignar opciones que harán mucho más cómoda nuestra estancia durante el desarrollo.

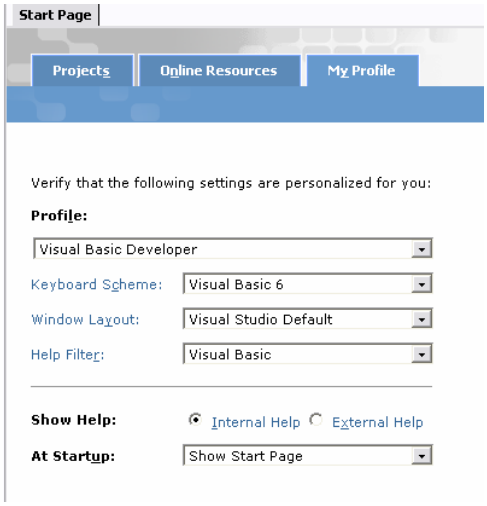


Fig 2.6 La página de inicio nos permite personalizar el IDE de Visual Studio.NET.

c) Cambios en los proyectos

En Visual BASIC 6.0, los proyectos empleaban un modelo basado en referencias (el archivo del proyecto contenía referencias a los elementos del proyecto especificando su ruta de acceso). Visual BASIC.NET utiliza un modelo basado en carpetas. Cuando agrega un archivo de texto, una copia de ese archivo queda colocada en la carpeta del proyecto; cuando se genera el proyecto, el archivo de texto se carga desde esa copia del archivo.

Anteriormente, el comando Quitar quita un elemento de un proyecto, pero el archivo aún permanece guardado en disco. En Visual BASIC .NET, el comando Eliminar, quita y elimina el archivo, reemplaza al comando Quitar. El comando Excluir del proyecto se puede utilizar para excluir un archivo de un proyecto sin eliminarlo.

Antes, los proyectos múltiples se podían agregar en el Explorador de proyectos y se denominaban "Grupos de Proyectos". En Visual BASIC .NET, el Explorador de Soluciones reemplaza al Explorador de proyectos y las Soluciones reemplazan a los Grupos de proyectos. Mientras los Grupos de proyectos sólo podían contener proyectos en Visual BASIC, las soluciones pueden contener proyectos creados en cualquier combinación de lenguajes de Visual Studio .NET.

En Visual BASIC 6.0, los archivos de proyecto (.vbp) y los archivos de grupos de proyectos (.vbg) eran archivos de texto que se podían editar directamente en un editor de texto. En Visual BASIC .NET, los archivos de solución y de proyecto están en formato XML y no se deben editar directamente.

d) Cambios en el Desarrollo para Internet

Visual BASIC 6.0 incluye varias características que admitían la programación en Internet: aplicaciones de Servicios de Internet Information Server (IIS), aplicaciones DHTML, documentos ActiveX y controles ActiveX que se podían descargar a páginas Web.

Visual BASIC .NET (y toda la tecnología .NET) se diseñó desde un principio para admitir la programación para Internet por lo que, las aplicaciones IIS utilizan el modelo de páginas Active Server (ASP) para crear aplicaciones que se ejecutan en los Servicios de Internet Information Server. Por otro lado, las aplicaciones DHTML emplean el modelo de objetos de HTML dinámico y el código de Visual BASIC para crear aplicaciones que pudieran responder a las acciones realizadas por el usuario en un explorador.

Los formularios Web Forms de Visual BASIC.NET se expanden sobre el modelo DHTML, proporcionando capacidades de interfaz de usuario dinámicas más eficaces así como validación del cliente además, la tecnología ASP.NET permite crear aplicaciones empleando las páginas de formularios Web Forms, y crear componentes mediante servicios Web XML.

e) Cambios en conectividad con Bases de Datos

En Visual BASIC 6.0, el acceso a los datos se consigue mediante objetos ADO (ActiveX Data Objects), RDO (Remote Data Objects) y los DAO (Data Access Objects). En Visual BASIC .NET, el acceso a los datos se consigue empleando ADO.NET.

En Visual BASIC .NET, el enlace de datos es un concepto mucho más amplio ya que se puede enlazar cualquier propiedad de un control a cualquier estructura que contenga datos.

f) Cambios en los Controles

Los controles sufrieron cambios muy importantes. Resaltan casos como que ahora todos los elementos manejan la propiedad de "text" en lugar del "caption" que algunos utilizaban. Otro aspecto importante es, que debido a la naturaleza del Framework.NET, todos los controles son objetos de la clase "Control".

También hay cambios importantes dentro de los métodos (características) como podemos apreciar en la tabla 2.1.

Visual BASIC 6.0	Equivalente de Visual BASIC .NET
Alignment 0 - Left Justify 1 - Right Justify	CheckAlign System.Drawing.ContentAlignment.MiddleLeft System.Drawing.ContentAlignment.MiddleRight
Appearance 0 – Flat 1 - 3D	FlatStyle FlatStyle.Flat FlatStyle.Standard

BackColor	BackColor
Caption	Text
CausesValidation	CausesValidation
Container	Parent
Enabled	Enabled
Font, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline	Font
ForeColor	ForeColor.
Height	Height, Size .
HWnd	Handle
Left	Left.
MousePointer	Cursor.
Nombre	Nombre
Parent	Método FindForm
Picture	Image
RightToLeft True False	RightToLeft RightToLeft.Yes RightToLeft.No
Style 0 – Standard 1 - Graphical	Appearance Appearance.Normal Appearance.Button
TabIndex	TabIndex
TabStop	TabStop
Tag	Tag
ToolTipText	Componente ToolTip.
Top	Top .
Value 0 – vbUnchecked 1 – vbChecked 2 - vbGrayed	CheckState CheckState.Unchecked CheckState.Checked CheckState.Indeterminate
Visible	Visible
Width	Width, Size.

Tabla 2.1

Los métodos mostrados en la tabla 2.2 no tienen equivalente en Visual BASIC.NET:

- DataChanged
- DataField
- DataFormat
- DataMember
- DataSource
- DisabledPicture
- DownPicture
- DragIcon
- DragMode
- HelpContextID
- Index
- MaskColor
- MouseIcon
- OLEDropMode
- UseMaskColor
- WhatsThisHelpID

Tabla 2.2

Como podemos apreciar, realmente no hay un cambio sustancial sin embargo ésto es engañoso pues la manera en como se utilizan diversos componentes básicos (color, fuente, etc.) hacen que sea verdaderamente un cambio radical su implementación.

g) Cambios en el sistema de coordenadas

En Visual BASIC 6.0, las coordenadas para formularios y controles se expresan en twips; en Visual BASIC .NET, las coordenadas se expresan en píxeles. Las coordenadas siguen de la misma manera.

h) Cambios en los eventos

En Visual BASIC 6.0, los eventos están ligados a objetos específicos y tienen su propio código de control de eventos. En Visual BASIC .NET, los eventos se ligan a controladores de eventos mediante delegados, permitiendo crear un único controlador de eventos para varios objetos.

i) Cambios en las fuentes

Las fuentes se controlan en Visual BASIC 6.0 de dos formas diferentes: como propiedades de fuente de formularios y controles, o como objetos stdFont. En Visual BASIC .NET, sólo existe un objeto System.Drawing.Font.

j) Cambios en los eventos de formularios

En Visual BASIC 6.0 el evento Initialize se utiliza para ejecutar código antes de cargar un formulario. En Visual BASIC .NET, el código de inicialización debe agregarse al constructor del formulario (Sub New()) después de la llamada a InitializeComponent().

En Visual BASIC 6.0 el evento Terminate se utiliza para ejecutar código después de descargar un formulario. En Visual BASIC .NET, el evento Terminate ya no se admite. El código de finalización debe ejecutarse dentro del método Dispose, antes de la llamada a MyBase.Dispose().

En Visual BASIC 6.0, el evento Unload utilizaba un argumento Cancel; en Visual BASIC .NET, se ha reemplazado por el evento Closed, que no tiene argumento Cancel.

k) Cambios en los gráficos

En Visual BASIC 6.0, se usan diversos métodos y propiedades de gráficos para dibujar en un formulario o control PictureBox. Además los gráficos se basan en las API de la interfaz de dispositivo gráfico (GDI) de Windows.

En Visual BASIC .NET, el espacio de nombres System.Drawing, que encapsula las nuevas API de GDI+, proporciona los gráficos. GDI+ expande las capacidades gráficas de Visual BASIC 6.0, pero no son compatibles.

l) Cambios en las matrices de controles

En Visual BASIC 6.0, se pueden usar matrices de controles para especificar un grupo de controles que compartían un conjunto de eventos. Los controles deben ser del mismo tipo y tienen que tener el mismo nombre.

En Visual BASIC .NET no se admiten las matrices de controles. Los cambios en el modelo de eventos hacen que las matrices de controles sean innecesarias. Lo mismo que las matrices de controles pueden compartir eventos en Visual BASIC 6.0, el modelo de eventos en Visual BASIC .NET permite que cualquier controlador de eventos controle eventos desde múltiples controles. Así se pueden crear grupos de controles de diferentes tipos que comparten los mismos eventos.

m)Cambios en MDI

En Visual BASIC 6.0, las aplicaciones de la interfaz de múltiples documentos (MDI) se crean agregando un formulario MDI a un proyecto y estableciendo la propiedad MDIChild de cualquier formulario secundario. En Visual BASIC .NET, cualquier formulario se puede convertir en un formulario MDI primario estableciendo la propiedad IsMdiContainer en true.

El comportamiento de las aplicaciones MDI también ha cambiado. En Visual BASIC 6.0, una aplicación MDI que contiene un formulario que no fuera un formulario MDI secundario no acababa hasta que ese formulario se cerraba, incluso si el formulario MDI primario estaba cerrado. En Visual BASIC .NET, la aplicación acaba cuando el formulario de inicio se cierre, sin tener en cuenta los formularios de la aplicación que no sean MDI.

n) Cambios en los cuadros de diálogo

En Visual BASIC 6.0 se puede mostrar un formulario como cuadro de diálogo modal llamando al método Show con un parámetro de vbModal. En Visual BASIC .NET, se usa el

método ShowDialog para mostrar un formulario modalmente y el método Show para mostrar no modalmente un formulario.

Además, el control CommonDialog de Visual BASIC 6.0 proporcionaba varios cuadros de diálogo predefinidos. En Visual BASIC .NET, dicho control se ha reemplazado con los controles ColorDialog, FontDialog, OpenFileDialog, PageSetupDialog, PrintDialog, PrintPreviewDialog y SaveFileDialog.

o) Cambios de color

En Visual BASIC 6.0, los colores se representan por medio de un valor de tipo Long; en Visual BASIC .NET los colores son del tipo System.Drawing.Color.

En Visual BASIC 6.0 se proporcionan constantes para los ocho colores estándar, así como para los colores del sistema que podían utilizarse para asignar un color a las preferencias del usuario para el sistema. En Visual BASIC .NET, los colores del sistema son del tipo System.Drawing.SystemColors.

p) Cambios en la depuración

En Visual BASIC 6.0, en la mayoría de los casos se pueden hacer modificaciones en modo de interrupción, y continuar la depuración sin parar y reiniciar. En Visual BASIC .NET, los cambios de código en modo de interrupción requieren que el proyecto se genere de nuevo antes de que los cambios en el código tengan efecto; editar y continuar ya no es compatible.

En Visual BASIC 6.0, los errores sintácticos del código generaban un cuadro de dialogo de mensaje de error que se mostraba durante la depuración. En Visual BASIC .NET, la mayor parte de los errores sintácticos aparecen en la ventana Lista de Tareas en tiempo de diseño; las excepciones de tiempo de ejecución aparecen en la ventana Resultados.

En Visual BASIC 6.0, se pueden evaluar expresiones y establecer valores en la ventana Inmediato. En Visual BASIC .NET, la ventana Comando reemplaza a la ventana Inmediato; cuando se encuentra en modo Inmediato, la funcionalidad es la misma. Visual BASIC .NET admite también varias opciones nuevas para las expresiones.

q) Cambios en los espacios de nombre

Es posible que los espacios de nombres parezcan, al principio, un nuevo concepto en Visual BASIC .NET, pero son parecidos a las bibliotecas de objetos en Visual BASIC 6.0.

En Visual BASIC 6.0 existen diversas bibliotecas que contienen objetos que se usan para generar una aplicación. En Visual BASIC .NET, los objetos se contienen en ensamblados que forman parte de la biblioteca de clases de .NET Framework. Cada ensamblado representa un espacio de nombres.

r) Cambios en tiempo de ejecución

En las aplicaciones creadas con Visual BASIC 6.0, el archivo de tiempo de ejecución de Visual BASIC (MSVBVM60.DLL) se debe instalar en un equipo cliente para que la aplicación se pudiera ejecutar. Normalmente, este archivo se incluye en los paquetes de instalación para que se pudiera instalar junto con la aplicación si fuera necesario.

En Visual BASIC .NET las aplicaciones requieren el Common Language Runtime de .NET Framework, un entorno de tiempo de ejecución que administra la ejecución de código en un equipo cliente. Los archivos del Common Language Runtime están empaquetados en un módulo de combinación que se incluye automáticamente en los instaladores creados con los proyectos de implementación de Visual Studio.

s) Cambios en los tipos de datos universales

Visual BASIC .NET actualiza el tipo de datos universales para interoperatividad con el Common Language Runtime. En Visual BASIC 6.0, el tipo de datos Variant sirve como universal. Esto significa que puede almacenar datos de cualquier tipo en una variable de tipo Variant. En Visual BASIC.NET, el tipo de datos Object es universal. Una variable de tipo Object puede contener datos de cualquier tipo.

t) Cambios en los tipos de datos enteros

Visual BASIC .NET actualiza los tipos de datos enteros para interoperatividad con otros lenguajes de programación y con el Common Language Runtime.

La tabla 2.3 siguiente muestra las correspondencias entre tipos enteros en Visual BASIC 6.0 y Visual BASIC .NET.

Tamaño del entero	Tipo y carácter de tipo identificador de Visual BASIC 6.0	Tipo y carácter de tipo identificador de Visual BASIC .NET	Tipo de Common Language Runtime
8 bits, con signo	(ninguno)	(ninguno)	System.SByte
16 bits, con signo	Integer (%)	Short (ninguno)	System.Int16
32 bits, con signo	Long (&)	Integer (%)	System.Int32
64 bits, con signo	(ninguno)	Long (&)	System.Int64

Tabla 2.3

En sistemas de 32 bits, las operaciones enteras de 32 bits son más rápidas que las enteras de 16 bits o de 64 bits. Esto significa que en Visual BASIC .NET, el tipo numérico Integer es el más eficaz y fundamental. Se puede mejorar el rendimiento de las aplicaciones cambiando las declaraciones Long a Integer cuando migre a Visual BASIC .NET.

2.4. Perspectivas

2.4.1. Visual BASIC.NET 2005

Los cambios más importantes del lenguaje Visual Basic incluyen My (Elementos que proporcionan accesos directos bien organizados a las clases de Framework o conjuntos de objetos dinámicos creados al agregar formularios, configuraciones, recursos y servicios Web a los proyectos.), comentarios XML y componentes genéricos. Algunas nuevas instrucciones del idioma llenan huecos lógicos, entre las que se incluyen Using, Continue, TryCast y la palabra reservada Global. La sobrecarga de operadores y los operadores de conversión son novedades, junto con el operador IsNot. Otros cambios útiles incluyen los tipos sin signo, las instancias predeterminadas de formularios, las advertencias del compilador y los límites explícitos de matrices.

El lenguaje Visual Basic 2005 incorpora algunas características importantes y otras mejoras más pequeñas que, en su conjunto, aumentan considerablemente la facilidad de uso y la productividad de los desarrolladores. El lenguaje se ha vuelto más completo y resulta más parecido a C# en características importantes como los comentarios XML y la sobrecarga de operadores.

Con un gran número de mejoras del IDE, Visual Basic 2005 está destinado a ser la mejor versión de Visual Basic que jamás se ha lanzado.

2.4.2. La Supervivencia de BASIC.NET

Tal y como ya he comentado el lenguaje Visual Basic .Net es un verdadero lenguaje orientado a objetos. Esta conversión a un lenguaje 100% orientado a objetos fue inevitable y aunque es muy provechoso, no deja de tener sus inconvenientes. Muchos programadores profesionales consideraban a VB como un lenguaje menor, era (y es) el lenguaje más utilizado del planeta debido sobretodo a su sencillez. Con VB .Net, se precisan unos conocimientos sólidos de POO para apreciar en profundidad las ventajas del mismo y realizar programas que aprovechen todas sus posibilidades. Un programador que posea todos estos conocimientos optará probablemente por trabajar en C# ó Java, que tienen una sintaxis más estándar.

Se podría decir que quizás el suprimir la sencillez de Visual Basic en aras de hacerlo más moderno acabe con su principal encanto.

3. Fundamentos de Seguridad Informática

Desde la prehistoria el hombre ha catalogado a su información como lo más importante y sobre todo, lo más valioso que puede poseer. Esto es claro, ya que todos los seres vivos de este planeta, usan la información que van acumulando (o que viene de nacimiento) para poder sobrevivir.

Entre los animales, la comunicación es la más importante, pues con ella se puede difundir la información obtenida, para el bien de la comunidad (imagínense que la cebra que vislumbra al león, no le informe a las demás que ya lo vio).

Entre los seres humanos, esta comunicación se traduce en cuestiones mucho más poderosas, pues no sólo nos encontramos ante la necesidad de comunicarnos, sino de sólo comunicar lo que queremos. Por lo mismo, el concepto de seguridad aparece como un elemento clave en nuestra formación como "seres pensantes".

Es así como vemos que la seguridad no es un concepto nuevo y que estamos acostumbrados desde pequeños a valorarla. La información es lo más valioso que existe (sino pregúntenle a Hitler y el día D) y por lo tanto hemos tratado desde el principio de los tiempos de protegerla.

De esta manera nacen 2 divisiones muy interesantes de la seguridad, lo que es la seguridad interna y la seguridad externa. Por seguridad interna me refiero a la seguridad en nuestro lugar de trabajo; ya sea, nosotros mismos, nuestra propia casa, una base militar o el Pentágono, siempre será necesario que donde guardemos la información, sea un lugar seguro, disponible sólo a los que deben verla, lejos de peligros, de pérdidas, etc. Con seguridad en externa quiero englobar a todas aquellas técnicas que nos sirven para que la información llegue a su destino, sin ser modificada (recuerdan el juego: "teléfono descompuesto"), reemplazada, interceptada, etc. Dentro de esta Seguridad Externa, podemos ver que hay dos opciones, que son el proteger o a la información o al canal que la lleva. De hecho, la mejor opción es protegerlas a ambas.

La seguridad no es ajena a nosotros de ninguna manera y la utilizamos a diario. Por lo que no será de extrañar que los conceptos que se mencionarán, sean ya ampliamente conocidos.

3.1. Historia

Del antiguo Egipto a la era digital, los mensajes cifrados se han utilizado para proteger la información. En especial, todos los mensajes cifrados y el concepto de seguridad de la información está relacionado en sus primeras etapas (hasta hace poco de hecho) con las estrategias militares, con la protección de información militar, por lo que no es de extrañar que todos los antecedentes sean de tipo militar.

3.1.1 Inicio – La Criptografía

La criptografía (del griego kryptos, "escondido", y graphein, "escribir"), el arte de enmascarar los mensajes con signos convencionales, que sólo cobran sentido a la luz de una clave secreta, nació con la escritura. Su rastro se encuentra en las tablas cuneiformes, y los papiros demuestran que los primeros egipcios, hebreos, babilonios y asirios conocieron y aplicaron sus técnicas, que alcanzan hoy su máxima expresión gracias al desarrollo de los sistemas informáticos y de las redes mundiales de comunicación.

El primer criptosistema que se conoce fue documentado por el historiador griego Polybios, se trataba de un sistema de sustitución basado en la posición de las letras en una tabla. También los romanos utilizaban sistemas de sustitución, como el del César, ya que se tienen bases para creer que Julio César lo utilizó en sus campañas. Otro de los métodos criptográficos utilizados por los griegos fue la escitala espartana, un método de transposición basado en un cilindro que servía como clave en el que se enrollaba el mensaje para poder cifrar y descifrar.

En 1465 el italiano León Batista Alberti inventó un nuevo sistema de sustitución polialfabética que supuso un gran avance de la época. Otro de los criptógrafos más importantes del siglo XVI fue el francés Blaise de Vigenere que escribió un importante tratado sobre "la escritura secreta" y que diseñó una cifra que ha llegado a nuestros días asociada a su nombre.

Durante los siglos XVII, XVIII y XIX, el interés de los monarcas por la criptografía fue bastante importante, ya que se lograron avances sumamente significativos. Las huestes de Felipe II utilizaron durante mucho tiempo una cifra con un alfabeto de más de 500 símbolos que los matemáticos del rey consideraban inexpugnable. Cuando el matemático francés Francois Viete consiguió criptoanalizar aquel sistema para el rey de Francia, a la sazón Enrique IV, el conocimiento mostrado por el rey francés impulsó una queja de la corte española ante del papa Pío V acusando a Enrique IV de utilizar magia negra para vencer a sus ejércitos. Por su parte, la reina María Estuardo, reina de los Escoceses, fue ejecutada por su prima Isabel I de Inglaterra al descubrirse un complot de aquella tras un criptoanálisis exitoso por parte de los matemáticos de Isabel.

3.1.2. Siglo XX

Hasta este momento, sólo se había tenido esta preocupación de proteger la información. Sin embargo con la llegada de las computadoras a la estrategia militar, empieza a surgir un interés genuino en mayor seguridad, pues no sólo era necesario proteger la información, sino también a quienes se dedican a protegerla y a decodificarla.

Desde el siglo XIX hasta la Segunda Guerra Mundial, las figuras más importantes fueron la del holandés Auguste Kerckhoffs y del prusiano Friedrich Kasiski. Pero es en el siglo XX cuando la historia de la criptografía vuelve a presentar importantes avances. En especial durante las dos guerras mundiales, que fueron sin lugar a dudas las impulsoras de la tecnología moderna.

A partir del siglo XX, la criptografía usa una nueva herramienta que permitirá conseguir mejores y más seguras cifras: las máquinas de cálculo. La más conocida de las máquinas de cifrado, posiblemente sea la máquina alemana Enigma. Enigma era una máquina de rotores que automatizaba los cálculos que se tenían que realizar para las operaciones de cifrado y descifrado de mensajes. Para vencer a la inteligencia alemana, fue necesario el concurso de los mejores matemáticos de la época y un gran esfuerzo computacional.

Tras la conclusión de la Segunda Guerra Mundial, la criptografía tiene un desarrollo teórico importante; siendo Claude Shannon y sus investigaciones sobre teoría de la información esenciales hitos en dicho desarrollo.

3.1.3. La Informática (Virus y Hackers)

Los avances en computación automática suponen tanto una amenaza para los sistemas existentes como una oportunidad para el desarrollo de nuevos sistemas.



Fig. 3.1 John Von Neumann

En 1949 se da el primer indicio de definición de virus. John Von Neumann, expone su "Teoría y organización de un autómata complicado". Este artículo, resultó ser el artículo que definió el futuro de la arquitectura de todas las computadoras que se diseñarían de ese momento en adelante.

En 1959: En los laboratorios AT&T Bell, se inventa el juego "Guerra Nuclear" (Core Wars) o guerra de núcleos de ferrita. Consistía en una batalla entre los códigos de dos programadores, en la que cada jugador desarrollaba un programa cuya misión era la de acaparar la máxima memoria posible mediante la reproducción de si mismo.

Para la década de los sesenta empieza la entrada de las computadoras a los servicios públicos, esto lo vemos en el primer año de la década, cuando las llamadas telefónicas se procesan por primera vez por una computadora.

Para 1967, la agencia de los proyectos de investigación avanzada (ARPA) comienza a trabajar junto con especialistas en computadoras de ESTADOS UNIDOS, para formar una gran red de computadoras. Las computadoras actuarían como pasarelas a las supercomputadoras de una gran variedad de instituciones de los Estados Unidos y proporcionaría una parte importante de lo qué se convertiría en Internet en los años venideros. Dos años después, ARPA crea ARPANET, un servicio diseñado para proporcionar maneras eficientes de comunicación entre la comunidad científica.

Los primeros hackers de computadoras emergen en el MIT en 1969. Toman prestado su nombre de un término usado por los miembros de un grupo de maquetas de tren en la escuela que "hackean" (alteran) los trenes eléctricos, las pistas, y los interruptores para hacerles que se hagan más rápidos y diferentes.

Ya en la década de los setenta, empieza la difusión de virus con el virus Creeper que es difundido por la red ARPANET en 1970. El virus mostraba el mensaje "¡SOY CREEPER... ATRAPAME SI PUEDES!". Otro virus famoso de la época fue el virus Rabbit, hacía una copia de si mismo y lo situaba dos veces en la cola de ejecución del ASP de IBM lo que causaba un bloqueo del sistema.

A mediados de los años 70 el Departamento de Normas y Estándares norteamericano publica el primer diseño lógico de un cifrador que estaría llamado a ser el principal sistema criptográfico de finales de siglo: el Estándar de Cifrado de Datos o DES. En esas mismas fechas ya se empezaba a gestar lo que sería la, hasta ahora, última revolución de la criptografía teórica y práctica: los sistemas asimétricos. Estos sistemas supusieron un salto cualitativo importante ya que permitieron introducir la criptografía en otros campos que hoy día son esenciales como el de la firma digital.

Para la década de los ochenta, ARPANET había crecido de una manera inimaginable, por lo que un sitio que no se había planeado para ser seguro, empieza a sufrir por sus deficiencias. Es en esta década, cuando el público empieza a tener acceso y con las máquinas unidas en esta gran red, empiezan a ser atacadas.

Y así, inicia la década con la infección de un "gusano" que fue originada por Robert Tappan Morris, un joven estudiante de informática de 23 años, aunque según él fue un accidente. Es mismo año, la pandilla de Roscoe, incluyendo a Kevin Mitnick, invade el sistema informático de los E.E.U.U sobre arrendamientos.



Fig. 3.2 Kevin Mitnick “el hacker” más famoso de la historia.

Ante la incursión de los delitos informáticos a la realidad cotidiana, se empieza a rastrear a aquellos que explotaban

las vulnerabilidades de los sistemas. En 1981 Ian Murphy ('Capitan Zap') es el primer hacker que se juzga y condena como criminal. Murphy hackeaba las computadoras de AT&T y cambiaba los relojes internos que median las tarifas de facturación. La gente conseguía tipos de descuento nocturnos cuando llamaban al mediodía. Ese mismo año Kevin Mitnick, de 17 años, es arrestado por robar los manuales de las computadoras del centro de intercambio de datos de la Pacific Bell en Los Angeles, California. Lo procesan como menor y es condenado a la libertad condicional.

En 1983 se forma Internet cuando ARPANET está partida en secciones militares y civiles. Es decir, nace Internet y con ella empieza la gran aventura de la seguridad informática.

El 12 de octubre de 1985, hubo una publicación del New York Times que hablaba de un virus que se distribuyo para optimizar los sistemas IBM basados en tarjeta gráfica EGA, pero al ejecutarlo salía la presentación pero al mismo tiempo borraba todos los archivos del disco duro, con un mensaje al finalizar que decía "Caíste".

El primer contagio masivo de computadoras se da en 1987. El Virus MacMag también llamado Peace Virus sobre computadoras Macintosh. Este virus fue creado por Richard Brandow y Drew Davison y lo incluyeron en un disco de juegos que repartieron en una reunión de un club de usuarios. Uno de los asistentes, Marc Canter, consultor de Aldus Corporation, se llevó el disco a Chicago y contaminó la computadora en el que realizaba pruebas con el nuevo software Aldus Freehand. El virus contaminó el disco maestro que fue enviado a la empresa fabricante que comercializó su producto infectado por el virus.

El virus de Navidad (1987) consiste en una tarjeta navideña digital enviada por correo electrónico, atasco las instalaciones en los EE.UU por 90 minutos. Cuando se ejecutaba el virus este tomaba los datos de la lista de contactos del usuario y se retransmitía automáticamente, además que luego se colgaba de la computadora anfitrión. Esto causo un desbordamiento de datos en la red.

El 16 de diciembre 1988, Kevin Mitnick, fue detenido sin fianza en los cargos que incluyen robar 1 millón de dólares en software a DEC (Digital Equipment Corporation), incluyendo el código fuente del sistema operativo VMS, y causar a la firma 4 millones de dólares en daños.

A finales de los ochenta, empiezan a surgir ya no sólo ataques sencillos, sino toda una ciencia dedicada al robo, alteración y destrucción de la información.

Un "Caballo de Troya" se distribuyo en 10 000 copias de un paquete con información sobre el SIDA. El programa, de una empresa panameña llamada PC Cybort, encriptaba el contenido del disco duro y pedía al usuario que pagara por la licencia de uso para obtener la clave de desencriptado.

3.1.4. Nuestros Días

Llegada la década de los noventa, empieza lo más importante y lo que da origen a una tesis de este tipo, lo que es la concienciación del usuario, del programador y de los administradores para generar políticas, software y protocolos seguros.

El virus más famoso de la historia, fue el virus Michelangelo, ya que había infectado de fábrica a más de 500 computadoras IBM. Este virus lo que hacía era borrar el contenido del disco duro cada 6 de marzo, que era el cumpleaños de este famoso artista.

El virus Natas inició una "epidemia" por todo México en 1994, éste seguramente fue escrito y distribuido por mexicanos.

Y la historia continúa, pero ahora nos enfrentamos no sólo a ataques y robos de información, sino además a toda una nueva gama de ataques muy de nuestros tiempos, como el hoax, el phishing, etc. todos ellos igual o más peligrosos que a los que se enfrentaba la ingeniería hace tiempo.

3.2. Conceptos Básicos

En esta sección estableceremos algunos de los conceptos imprescindibles para continuar con nuestro camino hacia un software seguro. Tal vez el conocimiento establecido aquí sea muy sencillo y para algunos trivial, de hecho eso sería lo mejor, que todos tuviéramos esta concienciación de seguridad y que estos temas fueran del uso común.

3.2.1. Vulnerabilidades

Podría decirse que una vulnerabilidad es una debilidad y por ello es cualquier cosa que le dé a un atacante la oportunidad de explotar una falla en el sistema. Para comprometer un sistema, el atacante desea explotar todas las vulnerabilidades del sistema. Una vulnerabilidad puede incluir:

- Un error de configuración de un servicio en ejecución.
- Una falla en el sistema operativo o una aplicación en el sistema.
- Una debilidad en un protocolo usado por un servicio dentro del sistema.

Cuando nuevas vulnerabilidades son encontradas en diversas plataformas o productos de uso general, ésta es reportada a los administradores y portales especializados para que así se puedan cubrir y tomar las medidas pertinentes.

3.2.2. Amenazas

Se refiere a cualquier método que intente o trate de vulnerar la seguridad de una red o sistema.

Su clasificación se basa en determinar cuál es el origen o motivo de ellas, dentro de las diferentes causas podemos contar las siguientes:

i) Naturales.- tales como terremotos, incendios, inundaciones, etc.

ii) Recursos Humanos.- éstas son las más importantes por la gran variedad que hay de ellas así como por todos los daños que pueden llegar a causar, y los diversos factores que pueden motivar a alguien para llevarlas a cabo, entre las que se encuentran:

(1) Venganza.- las más comunes, generalmente hechas por un antiguo (o actual) empleado, que por alguna razón no está conforme. Al conocer el sistema, puede atacarlo de una manera directa y eficaz.

(2) Ignorancia.- se puede dar acceso a alguien no autorizado que haga uso de la buena fe, falta de cultura informática o falta de capacitación de los

usuarios del sistema; así como de la falta de un sistema amigable y que facilite la labor de lo usuarios.

(3) Descuido.- también muy común, al dejar la contraseña en un lugar visible o descuidar aspectos importantes sobre la estructura del sistema.

(4) Diversión.- es muy común, sobre todo con los "script kiddies" que echan a andar código "para ver qué pasa", esto puede desencadenar en un gran ataque si el sistema no tiene seguridad mínima.

(5) Ocio.- el hacker puede no tener nada mejor que hacer (bien dicen que es la madre de todos los vicios), ésto, aunado a aspectos de operación segura no considerados durante el desarrollo del sistema, lo hacen vulnerable.

(6) Beneficio Personal.- como robos, fraudes, evasión de pagos, etc.

3.2.3. Ataques

Se define como la realización de una amenaza. Su clasificación no está definida y se pueden hacer de diferentes maneras. Veamos primero las que están clasificadas por su origen:

- Externo estructurado.- ataques por personas u organizaciones maliciosas.
- Externo no estructurado.- ataques por agentes inexpertos como los "script kiddies".
- Internas.- ataques por personas de la misma compañía u organización (personal con conocimiento).

En el caso de impacto al sistema tenemos las siguientes clasificaciones:

- Ataques activos.- incluyen a quienes desean modificar o destruir la información mientras esté siendo transmitida o mientras esté en el sistema objetivo (Puertas traseras y troyanos, modificación o borrado de archivos del sistema).
- Ataques pasivos.- incluyen a quienes no desean modificar o destruir la información, sino simplemente interceptarla con el fin de monitorearla para obtener información confidencial, como contraseñas, números de tarjeta de crédito, etc.

También los podemos clasificar por el tipo de ataque:

- Ataques de acceso.- el intruso trata de ganar acceso a los recursos del sistema explotando fallas en el software, como "sobrecarga de pila" y "goteo de información" y con ello ganar más privilegios de los debidos dentro del sistema.
- Ataques de negación del servicio (DoS).- el intruso trata de denegar el acceso a los recursos del sistema a los usuarios autorizados.

- Ataques de reconocimiento.- el intruso trata de hacer un mapa del los servicios del sistema para explotar las vulnerabilidades que se encuentren.

Por supuesto que existen más amenazas, motivos y clasificaciones, sin embargo, no es el objetivo de este trabajo hacer un análisis detallado de ellos.

3.2.4. Características de un Sistema Seguro

Podemos decir que un sistema es seguro cuando cumple con las siguientes características que se reciben el nombre de "Servicios de Seguridad":

- Disponibilidad.- la información debe estar disponible para que todos los usuarios puedan acceder a ella en el momento que lo necesiten.
- No Repudio.- se debe tener la certeza de que el destinatario recibió la información y que éste no pueda negar haberla recibido.
- Confidencialidad.- la información sólo debe ser accedida por quienes tienen derecho de accederla y por ello, cuando está en tránsito, debemos protegerla de ataques de intercepción de información.
- Integridad.- se refiere a que la información debe estar como fue generada, es decir, no debió haber sido cambiada ni durante la transmisión, ni durante su almacenamiento.
- Autenticidad.- es el proceso que verifica que quien accede a la información es quien dice ser.
- Control de Acceso.- se requiere que el acceso a la información sea controlado, es decir que se sepa quién, cuándo y dónde tuvo acceso a la información, a pesar de ser un usuario con autorización.

Sin lugar a dudas, estas características a pesar de ser tan básicas y algunos podrían decir que son "obvias" son muy difíciles de alcanzar, tanto así que la seguridad se encarga de estudiar los métodos que nos permitan garantizar estas características.

3.2.5. Código Malicioso

Seguramente los más sofisticados tipos de amenazas para las computadoras son los programas hechos para explotar las vulnerabilidades de los sistemas de cómputo. De hecho, no sólo los compiladores y sistemas operativos son afectados, sino que también las aplicaciones sufren los ataques más comunes.

En la figura 3.3 podemos apreciar la clasificación de los códigos maliciosos (también conocidos como malware):



Figura 3.3 Clasificación de Código Malicioso

Como vemos, este tipo de programas se puede dividir principalmente en dos clases, los que necesitan un cliente (un programa residente) y los que son independientes y pueden correr directamente sobre el sistema operativo. Además existe otra característica que es el hecho de que los virus, los gusanos y los zombis, se pueden reproducir y pueden ser activadas en ese mismo equipo o en otro. Esta clasificación no es estricta, por ejemplo, hay caballos de Troya que pueden ser parte de un gusano o de un virus, sin embargo esta organización, nos permitirá tratarlos por sus rasgos más significativos.

a) Puertas Traseras

Una puerta trasera es una entrada secreta a un programa que permite que se pueda tener acceso a la aplicación, sin tener que pasar por los procedimientos de seguridad usuales. Las puertas traseras han sido ampliamente usadas por los programadores para depurar y probar los programas. Los casos más usuales son: que el programador quiera evitarse los pasos de autenticación; o que quiera tener un acceso de "emergencia" por si los métodos de seguridad fallan.

Las puertas traseras es código que reconoce una serie especial de eventos poco comunes, o una identidad especial, o una serie de eventos especiales. Sin embargo, éstas se vuelven un problema cuando los hackers se enteran de ellas por diversos métodos (ingeniería social, caballos de Troya, etc) o cuando el programador es deshonesto y utiliza esta información a su favor.

Este tipo de vulnerabilidades se controlan ya sea en la etapa de desarrollo o por medio de parches de seguridad.

b) Bombas Lógicas

Ésta es una de las formas más antiguas de vulnerar una aplicación. Son parecidos a los caballos de Troya. Dentro de un programa legítimo se instala el código que fue hecho para "explotar" en un cierto día, o cuando ciertos archivos falten o algún evento se cumpla (hasta una orden directa).

c) Caballos de Troya

Un caballo de Troya es un programa útil (o por lo menos eso pensamos) que contiene código oculto que cuando es convocado, realiza acciones dañinas o no deseadas.

En general se usan para labores de espionaje, espionaje que no puede ser realizado directamente por un usuario no autorizado. Por ejemplo, puede guardar todas las claves del sistema y ponerlas en formato legible para cualquier usuario, o también puede ejecutarse en segundo plano, mientras se usa otra aplicación, el caballo de Troya puede destruir o alterar la información.

d) Zombi

Un zombi es un programa que secretamente toma el control de una computadora conectada a Internet, para lanzar desde ese equipo ataques que son muy difíciles de rastrear hasta su origen. Generalmente son usados para realizar ataques de negación de servicio, generalmente contra sitios Web. Esto es posible debido a que el zombi se puede colocar en miles de equipos, que pueden realizar el ataque de forma simultánea.

e) Virus

Un virus es un programa que puede "infectar" otros programas modificándolos; la modificación incluye la copia de los programas por el virus, que una vez infectados pueden contagiar otros programas.

Como su contraparte biológica, los virus se transmiten, tomando el control de una célula viva y reemplazan el ARN por su propio código para que estas puedan reproducirlo. En el caso de los virus, esto también aplica, ya que toman el control de un programa, se copian a él y con él se ayudan a reproducir. Esto es muy peligroso, ya que se pueden esconder en cualquier tipo de archivo y su propagación puede ser por cualquier dispositivo de almacenamiento, por Internet o por cualquier medio de compartición de archivos.

Los virus son muy peligrosos debido a que pueden realizar cualquier actividad que el usuario pueda realizar una vez ya instalados en la máquina contagiada.

3.2.5.1.1. Ciclo de Vida de los Virus

- Creación.- el programador establece las características y capacidades del virus: sus módulos de reproducción, de ataque y de defensa.
- Gestación.- el virus después de haber infectado el sistema se aloja, en la memoria del computador para esperar una orden o instrucción para poder causar los daño.
- Reproducción y propagación.- el virus ejecuta la misión principal para la que fue creado y se reproduce para continuar con su labor de infección.
- Activación.- el virus comienza a funcionar en un momento concreto, cuando se cumplen unas condiciones específicas. el virus ejecuta su módulo de ataque y produce el daño en el sistema para los que ha sido programado.
- Detección y Destrucción del virus.- la respuesta eficaz a los nuevos virus puede ser desarrollada por las casas de antivirus en cuestión de horas. Estas empresas reciben una muestra del nuevo ejemplar, analizan su código y lo incluyen en sus bases de datos lo más rápidamente posible.

3.2.5.1.2. Clasificación de los virus

A pesar de que existen más tipos de virus, esta clasificación engloba a la mayoría de ellos:

- Virus parasíticos.- los tradicionales y más comunes. Este tipo, se anexa a un archivo ejecutable y se replica buscando otros programas que infectar cuando el huésped es ejecutado,
- Virus residentes en memoria.- se cargan en memoria como parte del un programa residente. Desde esa posición, puede infectar cualquier otro archivo que se ejecute.
- Virus de sector de arranque.- infecta el registro maestro de arranque o los registros de arranque y se esparce cuando el sistema es arrancado desde el disco que contiene el virus.
- Virus furtivos.- una forma de virus específicamente diseñados para esconderse de la detección del software anti-virus.
- Virus polimorfo.- un virus que muta con cada infección, haciendo que las detecciones sean muy difíciles.
- Virus de correo electrónico.- estos virus se caracterizan por mandarse por mail a todos los usuarios de la lista de contactos.

f) Gusanos

Los gusanos no persiguen la infección de otros archivos, sino que su objetivo es propagarse lo más rápidamente posible haciendo copias de ellos mismos. Para lograrlo, utilizan las vías de comunicación entre computadoras: Internet, correo electrónico, disquetes etc. Se parecen a los virus de correo electrónico, sin embargo éstos últimos requieren ser activados, mientras que los gusanos están específicamente diseñados para hacerlo solos.

Pueden venir integrados en archivos adjuntos al correo, y una vez ejecutado el adjunto producen la infección, reenviándose a los contactos de la libreta de direcciones, con lo que su propagación se incrementa exponencialmente. Los gusanos suelen efectuar modificaciones en el registro de Windows.

Los gusanos pueden ser de varios tipos:

- Gusanos de correo electrónico.- son gusanos que se propagan a través de mensajes de correo electrónico, mediante la utilización de programas clientes de correo. Para ello se reenvían automáticamente a los contactos de la libreta de direcciones.
- Gusanos de IRC (gusanos de mIRC y de Pirch). Son gusanos que se propagan a través de canales de IRC (Chat). Los programas de IRC que emplean habitualmente para ello, son mIRC y Pirch.
- Gusanos de Windows32.- son gusanos que se propagan a través de las API de Windows (las funciones pertenecientes a un determinado protocolo de Internet).

En la figura 3.4 podemos apreciar como ha cambiado la manera en que los virus se han diseminado en las redes. Como vemos para el último caso de estudio (1999) los archivos adjuntos por mail son la principal causa, mientras que en 1996 lo eran los dispositivos de almacenamiento. Todo esto nos deja claro que la principal fuente de infección es la falta de conocimiento de los usuarios comunes, por lo mismo, el difundir una cultura de seguridad entre los usuarios es un imperativo para que disminuyan los ataques.

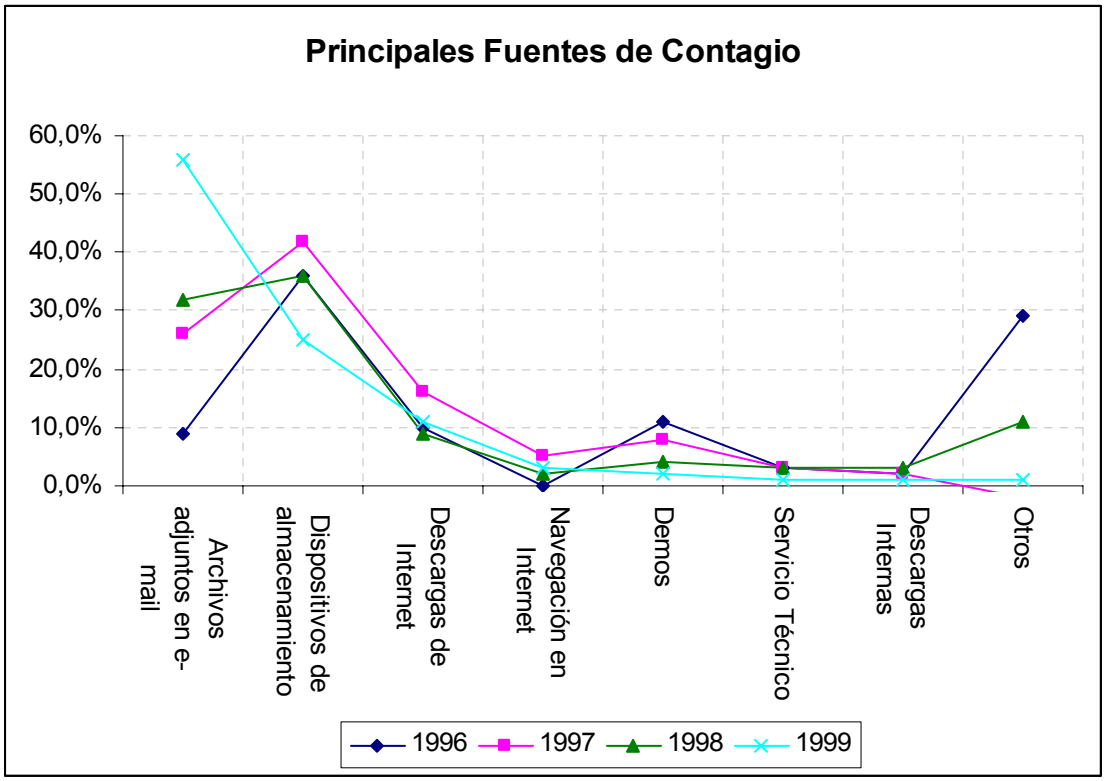


Figura 3.4 Principales fuente de infección

3.2.6. Ingeniería Social

La ingeniería social consiste en la manipulación de las personas para que voluntariamente realicen actos que normalmente no harían; en algunos casos no es perjudicial (por ejemplo un vendedor puede aplicar ingeniería social para conocer las necesidades de un cliente y ofrecer así mejor sus productos), si las intenciones de quién la pone en práctica no son buenas se convierte quizás el método de ataque más sencillo, menos peligroso para el atacante y por desgracia en uno de los más efectivos. Ese atacante puede aprovechar el desconocimiento de unas mínimas medidas de seguridad por parte de personas relacionadas de una u otra forma con el sistema para poder engañarlas en beneficio propio.

Consideremos el siguiente correo electrónico falso:

```
From: Super-User <root@sistema.com>
To: Usuario <user@sistema.com>
Subject: Cambio de clave

Hola,
Para realizar una serie de pruebas orientadas a conseguir un óptimo funcionamiento
de nuestro sistema, es necesario que cambie su clave mediante
la orden 'passwd'. Hasta que reciba un nuevo aviso (aproximadamente en una
```

```
semana), por favor, asigne a su contraseña el valor de su login (ejemplo, si su
login es "Dulce", su password será "DULCE", recuerde dejarlo en mayúsculas).
Rogamos disculpe las molestias. Saludos,
                                Administrador
```

Un usuario sin conocimientos mínimos de seguridad caería sin lugar a dudas en la trampa facilitándole la entrada al atacante que de alguna manera logró mandar este mail ilícito. Como vemos, el explotar la buena fe y voluntad de las personas se transmite a la red de manera inmediata.

3.2.7. Hoax y Spam

Un hoax (del inglés: engaño, bulo) es un mensaje de correo electrónico con contenido falso o engañoso y normalmente distribuido en cadena. Algunos informan sobre virus desastrosos, otros apelan a la solidaridad con un niño enfermo o cualquier otra noble causa, otros contienen fórmulas para hacerse millonario o crean cadenas de la suerte como las que existen por correo postal. Los objetivos que persigue quien inicia un hoax son: alimentar su ego, captar direcciones de correo y saturar la red o los servidores de correo.

En Diciembre de 1994, y casi al mismo tiempo de creación del spam, se envía el primer hoax masivo, que tenía como asunto Good Times. El contenido de este correo es: "¡Cuidado! Si llega un mensaje titulado 'Good Times', simplemente leyendolo, el virus malicioso actúa y puede borrar todos los contenidos del disco duro".

El hoax hace perder tiempo, crea confusión de los distintos virus o problemas que puedan surgir en la red, además de incrementar el tráfico en la red.

Por otra parte, el spam son mensajes electrónicos (habitualmente de tipo comercial) no solicitados y en cantidades masivas. Aunque se puede hacer por distintas vías, la más utilizada entre el público en general es la basada en el correo electrónico.

El spam mediante el servicio de correo electrónico nació a partir del día 5 de Marzo de 1994. Este día una firma de abogados de Canter and Siegel, publicó en Usenet un mensaje de anuncio de su firma legal, el cual en el primer día después de la publicación, facturó cerca de 10 000 dólares por casos de sus amigos y lectores de la red. Desde ese entonces, el marketing mediante correos electrónicos ha crecido a niveles impensados desde su creación

3.2.8. El Usuario Común

Sin lugar a dudas una de las partes más importantes para generar una cultura de seguridad, es la concienciación. La ignorancia juega un papel clave para que los delincuentes informáticos puedan poner en práctica sus técnicas, por lo mismo, todo el personal que se encuentre en un lugar debe estar conciente de las políticas de seguridad, además de dominar con los conceptos básicos.

Dentro de los nuevos usuarios de Internet o de los usuarios que no han tenido contacto con esquemas de seguridad, es normal caer en los 7 mitos de la seguridad en Internet:

1. Tengo un software antivirus; eso es todo lo que necesito.- este es el mito de Internet más común. El software antivirus sólo ofrece un tipo de seguridad (evitar que los virus infecten nuestro sistema) cuando estemos en línea. A pesar de ello, software antivirus no puede desviar a un hacker decidido.
2. No hay nada en mi computadora que pudiera querer un hacker.- la mayoría de nosotros creemos que esto es verdad. No obstante, un hacker podría querer la información personal que guardamos en el sistema (número de la cuenta bancaria, del seguro social, etc.) que pueden utilizar para hacer compras fraudulentas nuestro nombre.
3. Sólo las grandes compañías son objetivo de los hackers, no los usuarios particulares del hogar.- los hackers generalmente buscan presas fáciles y nuestra computadora es más fácil de violentar que una gran red corporativa. Además, las grandes compañías han invertido mucho en los departamentos de tecnología de la información, por lo que, son más difíciles de atacar.
4. Se necesita muchos conocimientos tecnológicos para ser un hacker.- ¡¡NO!!! Contrario a la creencia popular, no se necesita ser genio para hackear una computadora. Para hackear una computadora realmente se necesita muy poco conocimiento, porque cualquier motor de búsqueda listará las "herramientas para la piratería informática", las cuales están disponibles y se pueden descargar en pocos minutos; incluso traen las instrucciones.
5. Mi proveedor de Internet me brinda protección (antivirus y/o firewall) cuando estoy en línea.- rara vez los proveedores de Internet brindan protección total, por lo tanto, debemos verificar con nuestro proveedor de Internet sobre el nivel de seguridad que tiene contra virus y hackers.
6. Utilizo la conexión de acceso telefónico, por lo tanto no debo preocuparme por los hackers.- es verdad que los usuarios de banda ancha son más vulnerables a ser atacados. Con una conexión de acceso telefónico mucho más lenta, la dirección IP es dinámica, pero no nos protege de que un hacker instale un caballo de Troya de acceso furtivo para vernos cada vez que se conecte. Cuando se contagia con un caballo de troya, no importa si su conexión es de banda ancha o de acceso telefónico.
7. Tengo una computadora Macintosh.- con frecuencia los usuarios de Mac se sienten seguros porque la mayoría de virus están diseñados para las plataformas basadas en Windows. Sin embargo, eso no importa para un hacker; una computadora es una computadora, y no les interesa qué plataforma se utilice, sólo buscan puertos abiertos.

3.3. Estándares Internacionales

Para todas las ramas de la ingeniería se ha aceptado desde hace mucho tiempo la necesidad de estándares, para gobernar la calidad de los procesos físicos, eléctricos, etc. Sin ellos, el mundo sería un caos, pues cada quien haría las cosas como mejor le parecieran, dificultando así cualquier comunicación entre los diferentes participantes de un mismo proyecto.

En el caso de la seguridad, tenemos numerosos estándares que hablan sobre como certificar la seguridad de un sistema o de un proyecto. Este tipo de especificaciones han sido seguidas sobre todo en las comunicaciones y transmisión de datos; de modo que ya son ampliamente conocidas y uno puede estar certificado en ello.

Sin embargo, en el caso de desarrollo de software no tenemos el mismo nivel de aceptación. De hecho, éste es un gran problema ya que ni siquiera se acepta por toda la comunidad ingenieril que la "Ingeniería de Software" sea una realidad.

En esta sección, veremos algunos de los estándares internacionales que ayudan para este desarrollo.

3.3.1. La Importancia de los Estándares

Si bien, antes cada quién podía hacer las cosas como mejor le pareciera, pues la distribución no era mucha. Sin embargo, con el comercio internacional, se ha necesitado que todos nos apeguemos a una forma de realizar las cosas que sea similar y esté regulada. En el campo de las computadoras, ésto es aún más preocupante, pues debemos estar concientes que nuestros equipos se deberán comunicar con otros, y esta comunicación se debe dar de manera clara y sin errores. Por ello, los estándares están en todos lados, sobre todo en lo que nos concierne que es la computación.

Como todo siempre tenemos ventajas y desventajas. Las principales ventajas son:

- Un estándar asegura que habrá un gran mercado para un producto en específico. Esto hace que haya una producción mayor, y se puedan desarrollar tecnologías con costos más bajos.
- Un estándar permite a diferentes productores comunicarse, dándole al consumidor flexibilidad en la elección de equipo.

Las principales desventajas son:

- Un estándar tiende a congelar la tecnología. Ya que en lo que se publica la tecnología lo ha rebasado.
- Existen muchos estándares sobre el mismo tema. Esto puede traer como consecuencia que haya confusión entre los diferentes métodos de hacer las cosas.

3.3.2. TCSEC

El Criterio de Evaluación de Sistemas de Cómputo Confiable (TCSEC) es una colección de criterios que fue previamente utilizada para evaluar a los sistemas de información. Actualmente ya no es usada, sin embargo, mucho de los sistemas que existen, fueron evaluados utilizando este estándar y por lo mismo es importante todavía su estudio. Algunas veces el TCSEC es referido como "El Libro Naranja" debido a su pasta naranja.

La "Serie Arcoiris" es el nombre que se le dio a la colección de documentos de interpretación y documentos guía publicados por el Centro de Seguridad de Cómputo Nacional. Cada documento tiene un diferente color de pasta, de aquí su nombre, y están diseñados para expandir, clarificar los requerimientos de la TCSEC.

Un producto que ha sido evaluado por la TCSEC es un producto que sigue las normas establecidas y por lo tanto tiene un nivel de confianza dictaminado por la clase a la que pertenece; sin embargo esto no quiere decir que sea impenetrable. Es posible que se haya pasado por alto algo o que el auditor haya hecho una mala interpretación del estándar.

Por características de seguridad entendemos una función que se puede implementar y que responde a alguna política del sistema de seguridad. Como ejemplos podemos mencionar al control de acceso, al acceso permitido y a la auditoria. En el TCSEC no evalúa características aisladas, sino que especifica que tipo de características de seguridad debe tener para lograr una cierta clase. Las clases están divididas como se muestra en la Tabla 3.1:

Clasificación	Tipo	Descripción	Ejemplos
D Protección Mínima *(Sistemas que no cumplen con ninguna otra clasificación)*			
C Protección Discreta *(Sistemas que usan protección de archivos, directorios o elementos.)*	C1 Protección de Seguridad Discrecional		Algunas plataformas UNIX
	C2 Protección de Control de Acceso	Basado en C1, pero se agregan las Listas de Control de acceso.	La mayoría de los sistemas operativos comerciales caen en esta clasificación como por ejemplo Windows, NetWare, Oracle 7, IBM OS/400.
B Protección Obligatoria *(Sistemas que utilizan la "Mandatory Access Control" (MAC))*	B1 Protección de Seguridad Etiquetada	Lo mismo que C2, con la cualidad de que se etiquetan los archivos, directorios y elementos.	Trusted IRIX, HP-UX BLS
	B2 Protección Estructurada	Se basa en B1, con la separación de elementos críticos y no críticos y protección contra intrusos.	Multics y Trusted XENIX
	B3 Dominios Seguros	Basada en B2 con monitoreo a todos los accesos a objetos y procedimientos de recuperación del sistema.	La única plataforma es Betronics/Wang Federal XTS-300
A Protección Verificada *(El más alto nivel de seguridad para sistemas de cómputo.)*	A1 Protección Verificada.	Basada en B3 con la adición de la una prueba formal de integridad y diseño verificado.	Las únicas plataformas A1 son Boeing MLS LAN, Gemini Trusted Network Procesor y HoneyWell SCOMP.
	A2 No ha sido definida.		

Tabla 3.1 Clases según la TCSEC

Cada una de las clasificaciones, cuenta con un número de características que la deben o no cumplir, esto ya sea parcial, total o nulamente. En la tabla 3.2 podemos observar cuales son las características que evalúa la norma para las diferentes clases que ya mencioné con anterioridad, en el caso de que deban cumplirla se marcó con una ✓ mientras que si no deben cumplirla con un × y si su cumplimiento es parcial con un ☐.

		A1	B3	B2	B1	C2	C1
Políticas de Seguridad	Control de Acceso Discrecional	✓	☐	✓	✓	☐	☐
	Reutilización de Objetos	✓	✓	✓	✓	☐	×
	Etiquetas	✓	✓	☐	☐	×	×
	Integridad de Etiquetas	✓	✓	✓	☐	×	×
	Exportación de Información de Etiquetas	✓	✓	✓	☐	×	×
	Exportación de Dispositivos Multinivel	✓	✓	✓	☐	×	×
	Exportación de Dispositivos Mononivel	✓	✓	✓	☐	×	×
	Etiquetación Entendible de Salida	✓	✓	✓	☐	×	×
	Control de Acceso Obligatorio	✓	✓	☐	☐	×	×
	Etiquetas Sujetas a Sensibilidad	✓	✓	☐	×	×	×
	Etiquetación de Dispositivos	✓	✓	☐	×	×	×
Contabilidad	Identificación y Autenticación	✓	✓	✓	☐	☐	☐
	Auditoría	✓	☐	☐	☐	☐	×
	Ruta Confiable	✓	☐	☐	×	×	×
Aseguramiento	Arquitectura del Sistema	✓	☐	☐	☐	☐	☐
	Integridad del Sistema	✓	✓	✓	✓	☐	☐
	Pruebas de Seguridad	☐	☐	☐	☐	☐	☐
	Verificación y Especificaciones del Diseño	☐	☐	☐	☐	×	×
	Análisis del Canal Seguro	☐	☐	☐	×	×	×
	Administración de Instalaciones Confiables	✓	☐	☐	×	×	×
	Configuración de Administración	☐	✓	☐	×	×	×
	Recuperación Confiable	✓	☐	×	×	×	×
	Distribución Confiable	☐	×	×	×	×	×
Documentación	Guía de Usuario a Características Seguras	✓	✓	✓	✓	✓	☐
	Manual de Instalaciones Confiables	✓	☐	☐	☐	☐	☐
	Documentación de Pruebas	☐	✓	☐	✓	✓	☐
	Documentación de Diseño	☐	☐	☐	☐	✓	☐

3.3.3. ISO 17799

La norma ISO 17799 empezó como el "código de práctica" de Seguridad Informática del departamento de Industria y Comercio del Reino Unido. Fue publicado a principios de los noventa, pero no fue sino hasta 1995 cuando el Instituto de Estándares Ingleses lo aceptó y tomó el nombre de BS7799-1.

Este documento tenía sus seguidores, pero no era aceptado del todo. Para 1999 el estándar fue revisado minuciosamente su uso se extendió, tanto así que para el 2000, la ISO lo adoptó como parte de las normas que la componían y le dio el nombre de ISO 17799.

Como todos los estándares, sólo son guías y sirven para tener una certificación que nos avale como un producto de calidad, sin embargo, no son obligatorias, aunque deberían de tomarse como tal.

En el caso de la norma ISO 17799 que es para desarrollo e implementación de sistemas seguros (ya sean de software o hardware), tenemos que se cubren todos los aspectos de un sistema computacional avanzado. Debido a esto, es la norma más aceptada a nivel mundial y casi todas las compañías de software cumplen y se guían con sus indicaciones. De hecho Microsoft tomó en consideración varias de las especificaciones sugeridas para la realización de la plataforma .NET.

En la tabla 3.3 tenemos a la norma desglosada, con sus principales objetivos por capítulo.

Capítulo	Sección	Metas
1	Visión	
2	Términos y Definiciones	
3	Políticas de Seguridad	• Definición de seguridad informática. • Declaración de intención administrativa para dar soporte a los principios de seguridad informática. • Explicación de las especificaciones aplicables y principios generales, estándares y requerimientos a cumplir. • Explicación del proceso para informar acerca de incidentes de seguridad. • Definición del proceso de revisión para mantener las políticas. • Medios para lograr la efectividad de la política utilizada y de los cambios tecnológicos. • Nombramiento del responsable de la política de seguridad.
4	Seguridad Organizacional	• Manejo de la información dentro de la organización. • Mantener la seguridad de la información organizacional relativas al acceso de información por terceros. • Mantener la seguridad de la información organizacional cuando se ha dado su manejo en “outsourcing”.

5	Clasificación y Control de prioridades	• Mantener el inventario y la protección de las prioridades organizacionales, incluyendo prioridades de información, software y físicos. • Garantizar que las prioridades reciban la protección pertinente según su clasificación
6	Seguridad de Personal	• Reducir el riesgo de errores humanos, fraude, robo o uso equivocado de las instalaciones. • Asegurar que los empleados estén concientes tanto de las políticas de seguridad, así como de los conceptos básicos de seguridad. • Minimizar el daño de los incidentes de seguridad y de los fallos, estableciendo una rutina para aprender de ellos.
7	Seguridad Física y Ambiental	• Prevenir el acceso ilegal, daños e interferencias a la información clasificada. • Prevenir daños, pérdidas o compromiso de las prioridades ante posibles interrupciones.
8	Administración de Comunicaciones y Operaciones	• Asegurar el correcto y seguro modo de operación de las instalaciones. • Minimizar el riesgo de las fallas del sistema. • Proteger la integridad del software y los datos. • Mantener la integridad y la disponibilidad de la información, de su proceso y comunicación. • Prevenir el uso no autorizado ni permitido de la información.
9	Control de Acceso	• Controlar el acceso a la información. • Prevenir acceso no autorizado a los sistemas de información. • Asegurar la protección de los servicios de red. • Prevenir el acceso a equipo no autorizado. • Detectar actividades no autorizadas. • Asegurar la seguridad de la información cuando se utilizan dispositivos móviles y redes inalámbricas.
10	Desarrollo y Mantenimiento de Sistemas	• Tener la certeza de que la seguridad está planeada en el Sistema Operativo. • Prevenir la modificación, alteración o pérdida de datos en los sistemas de información. • Utilizar el mejor método criptográfico para la protección de los datos organizacionales. • Verificar que los sistemas de información y las actividades de soporte técnica cumplan con las políticas de seguridad. • Mantenimiento del software seguro para aplicaciones y datos.
11	Administración Continua de Negocios	• Planes para contrarrestar interrupciones en las actividades del la organización y procesos críticos de la empresa en caso de errores mayores, o accidentes.
12	Cumplimiento	• Eliminar cualquier omisión de leyes civiles, estatutos, regulaciones u obligaciones contractuales que puedan intervenir con la correcta aplicación de las políticas de seguridad. • Asegurar que el sistema cumpla con las normas y estándares establecidos. • Maximizar la efectividad del sistema de auditoría.

Tabla 3.3 Metas por capítulo de la ISO 17799

3.3.4. Criterios Comunes

A pesar que la ISO 17799 se encarga de dar los parámetros para evaluar la seguridad, necesitamos una herramienta que nos dé la pauta para saber como evaluar el sistema y saber de esta manera si se cumplen los lineamientos de la norma ISO 17799. Para ello existen los llamados criterios comunes que nos dan herramientas y métodos para dicha evaluación.

Los Criterios Comunes representan el resultado de los esfuerzos por desarrollar criterios para la evaluación de la seguridad en las tecnologías de información. Se basó principalmente en los modelos europeos (ITSEC), estadounidense (TCSEC) y canadiense (CTCPEC). Los criterios comunes resuelven las diferencias técnicas y de concepto entre todos estos criterios.

La estructura de los Criterios Comunes está dividida en tres partes:

- Parte I - Introducción y Modelo General.- se definen los conceptos generales y los principios de la evaluación de seguridad informática, su finalidad es expresar los objetivos de seguridad para la selección y definición de los requerimientos de seguridad y para escribir especificaciones de alto nivel para productos y servicios.
- Parte II - Requerimientos funcionales de Seguridad.- establece un conjunto de requerimientos funcionales como una vía estándar de corrección de los requerimientos funcionales para las TOE. Catálogo de funciones y clases posibles.
- Parte III - Requerimientos de Garantía de Seguridad.- establece un conjunto de componentes como un estándar de expresión de los requerimientos de garantía. Los criterios de evaluación para los programas y sistemas. Presenta los niveles de garantía de evaluación.

Cada una de las partes puede ser utilizada de manera distinta por consumidores, desarrolladores y evaluadores como podemos apreciar en la tabla 3.4

	Consumidores	**Desarrolladores**	**Evaluadores**
Parte I	Para información de conocimientos previos y de referencia.	Para información previa y diferencia para el desarrollo de requerimientos y formulación de especificaciones de seguridad para los TOEs (Target of Evaluation).)	Para información previa y para propósitos de consulta.
Parte II	Para guía y consulta cuando se formulan enunciados de requerimientos para funciones de seguridad.	Para referencia en la interpretación de enunciados de requerimientos y formulación de especificaciones funcionales del TOE.	Enunciados obligatorios de criterios de evaluación para determinar si el TOE cumple con las funciones de seguridad anunciadas.

Parte III	Para guía para determinar los niveles requeridos de garantía.	Para referencia para interpretar enunciados de requerimientos de garantía y para determinar las aproximaciones de garantía al TOE.	Enunciado obligatorio del criterio de evaluación para determinar la garantía del TOE.

Tabla 3.4 Funcionalidad de los Criterios Comunes por tipo de Usuario

Los siete Niveles de Garantía de Evaluación (EAL) que los criterios comunes dan a los sistemas evaluados son 7 y en la tabla, podemos ver algunos detalles al respecto.

	Descripción	**Equivalencia con TCSEC**
EAL 1	Funcionalmente probado	D
EAL 2	Probado estructuralmente	C1
EAL 3	Probado y verificado metódicamente	C2
EAL 4	Diseñado, revisado y probado metódicamente	B1
EAL 5	Semiformalmente diseñado y probado	B2
EAL 6	Semiformalmente verificado, diseñado y probado	B3
EAL 7	Formalmente verificado, diseñado y probado	A1

Tabla 3.5 Niveles de competencia según CC

3.3.5. Otras Normas para Ingeniería de Software

Si hay un aspecto en la computación a la cual se le ha dado todo un seguimiento en cuanto a normas internacionales lo más seguro es que sea el área de redes. Sin embargo, el software no ha sido tratado con esta profundidad, debido a que su existencia como una ingeniería es relativamente corta.

No es sino hasta los años noventa que empezaron a aparecer estándares para la elaboración de software de calidad. En general, existen dos estándares que intentan establecer normas para el desarrollo de software. Sin embargo, la seguridad, no es tomada en cuenta como parte integral, sino como un apartado, por lo que quedan atrás de la ISO 17799.

a) CMMI

El proyecto llamado "Modelo Integral de Madurez Capaz", ha comprendido a muchas personas de diferentes organizaciones en todo el mundo. En estas organizaciones se utilizaba el CMM de diversas maneras y todas ellas optaron por crear un marco de especificaciones que respondiera al mundo actual.

El proyecto fue patrocinado por el departamento de la defensa de EUA, especialmente por la Oficina de defensa, adquisición, tecnología y logística y gracias a este apoyo y a las diversas empresas que lo utilizaron, se formó el Marco CMMI, que es un conjunto de modelos integrados del CMMI, así como el método CMMI y soporte de productos.

Sin embargo, su uso ha empezado a caer en desuso, debido a que no contó con la realidad del momento ni con el futuro, pues el modelo sólo da estándares y métodos para un desarrollo de software organizacional y olvida parte esenciales como la misma seguridad de la información como se la da la norme ISO 17799.

b) ISO/IEC 15504

SPICE son las siglas para "Determinación de Capacidad de Mejora de Procesos de Software", y fue establecida en 1993 para darle soporte a la WG10 (Working Group 10) para el desarrollo de software. El WG10 es una entidad organizacional perteneciente a la ISO y a la IEC.

Este documento, se enfocó en el desarrollo, prueba y promoción de un estándar internacional para el desarrollo de software. Tomó varias de sus ideas del trabajo del CMMI, sin embargo no son iguales, ya que este es un documento mucho más avanzado que establece normas y no es un simple aglomerado de métodos como lo es la CMMI.

3.4. Criptografía

En el último tema de este capítulo, trataremos algunos conceptos básicos sobre cifrado. No es la intención dar un curso de criptografía, ni tampoco exponer los métodos, algoritmos y métodos de encriptación, sólo veremos a grandes rasgos cuales son los que se utilizan y el porqué de su importancia.

3.4.1. Conceptos Básicos

Realmente sólo es entender algunas palabras, y estas las podemos definir rápidamente en el siguiente párrafo.

Un mensaje original es conocido como mensaje en claro, mientras que al mensaje en código se le llama criptograma. El proceso de convertir el mensaje en claro a criptograma es conocido como encriptamiento o encriptación y el proceso inverso es conocido como descifrado o desciframiento. Las técnicas utilizadas para encriptar constituyen un área conocida como criptografía. Al proceso de utilizar las técnicas sobre mensaje se le denomina sistema criptográfico o cifrado. Las técnicas utilizadas para descifrar un mensaje sin conocimiento de los detalles de su encriptación se le conoce como criptoanálisis. Las áreas de criptografía y criptoanálisis se conocen como criptología.

En este caso, no nos interesa ahondar en el criptoanálisis, ya que nuestra labor es dar esa seguridad a la información, sin vulnerarlos.

3.4.2. Criptografía

Un sistema criptográfico se caracteriza por los siguientes elementos:

- El tipo de operación usado para transformar el mensaje en claro a criptograma. Todos los algoritmos de encriptación están basados en dos principios generales: sustitución (cambiar un carácter por otro) y transposición (reordenar los caracteres). El requerimiento principal es que no se pierda información.
- El número de claves usadas. Si el remitente y el destinatario utilizan la misma clave, se le denomina sistema simétrico, si ambos utilizan una clave diferente, se le denomina sistema asimétrico.
- El modo en que el mensaje en claro es procesado. Se puede dar por bloques, esto es dividiendo el mensaje en claro en partes o por elemento, obteniendo el resultado uno por uno.

3.4.3. Criptoanálisis

Hay dos maneras generales de atacar un sistema convencional de encriptación:

- Criptoanálisis.- este tipo de ataques recae en la naturaleza del algoritmo, además del conocimiento de algunas características del mensaje en claro. Este tipo de ataque explota las vulnerabilidades de los algoritmos para obtener la clave.
- Ataque por fuerza bruta.- el atacante trata cada posible clave en el criptograma, hasta que se obtenga un resultado. En promedio se deben tratar el 50% de las claves, antes de que se encuentre la correcta.

3.4.4. Cifrado Simétrico

Indudablemente es el modelo más conocido y difundido. Su modelo es muy sencillo y era el único antes de la existencia de las claves públicas.

El esquema consta de 5 elementos fundamentales:

- Mensaje en claro.- es el mensaje original que se desea transmitir. Éste se alimenta al sistema de encriptación.
- Algoritmo de encriptación.- el algoritmo realiza varias sustituciones y transformaciones al mensaje.
- Clave Secreta.- la clave secreta se alimenta de igual manera al algoritmo. El algoritmo genera un resultado diferente para diferentes claves en un mismo mensaje en claro.
- Criptograma.- este es el producto del algoritmo de encriptación. Depende del mensaje en claro y de la clave secreta. A la vista parece texto sin sentido y es virtualmente inviolable sin la clave (en teoría).
- Algoritmo de descifrado.- generalmente es el algoritmo de cifrado pero hecho de manera inversa. Si es exitoso, dará el mensaje en claro original.

En la figura 3.5 podemos ver como es que interactúan todos estos cinco elementos:

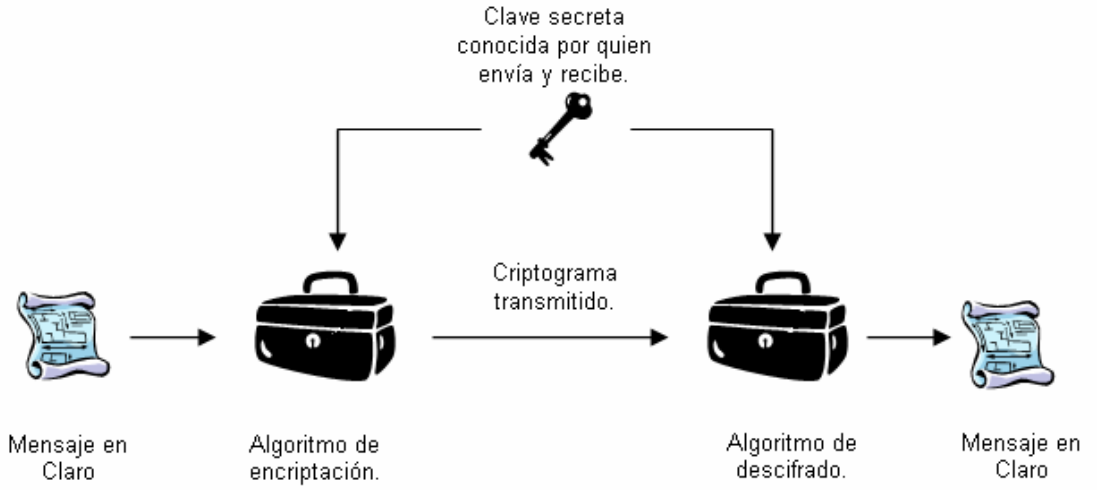


Figura 3.5 Sistema Simétrico de Encriptación

Existen dos requerimientos para el uso correcto de la encriptación tradicional:

- Necesitamos un algoritmo de encriptación robusto. Como mínimo, necesitamos que quien tenga acceso al criptograma y al algoritmo de encriptación, no pueda deducir rápidamente la clave con la que se cifró. Este requerimiento debe de ser cumplido de tal manera que si el atacante tiene varios mensajes en claro con sus respectivos criptogramas, no sea capaz de deducir la clave.
- El destinatario y el remitente deben de tener copias de la clave en un lugar seguro. Esta es la parte principal de la seguridad, si la clave es encontrada, la encriptación pierde su efectividad.

Al asumir que es prácticamente imposible descifrar el mensaje sin la clave adecuada, nosotros podemos hacer público el algoritmo de encriptación. Esto es lo que facilita que este sea el método más utilizado, pues todos pueden tener acceso a él y utilizar el mismo, sin preocuparse por criptoanalistas que traten de descifrarla, ya que se presume es invulnerable sin el conocimiento de la clave.

¿Cómo se produciría un ataque? La pregunta es de lo más natural. En el caso de un ataque generalmente tenemos la intercepción de un criptograma por un criptoanalista, y suponemos que no tiene acceso a la clave, pero sí tiene acceso a los métodos de encriptamiento y de descifrado (Figura 3.6). Su tarea es recuperar el mensaje en claro. Sin embargo esto no puede ser todo, sino que podría también tener acceso a futura información que se realice con la misma clave. Para ello podría analizar el sistema de tal manera que pueda tener acceso a la clave, y que esta pueda ser utilizada posteriormente.

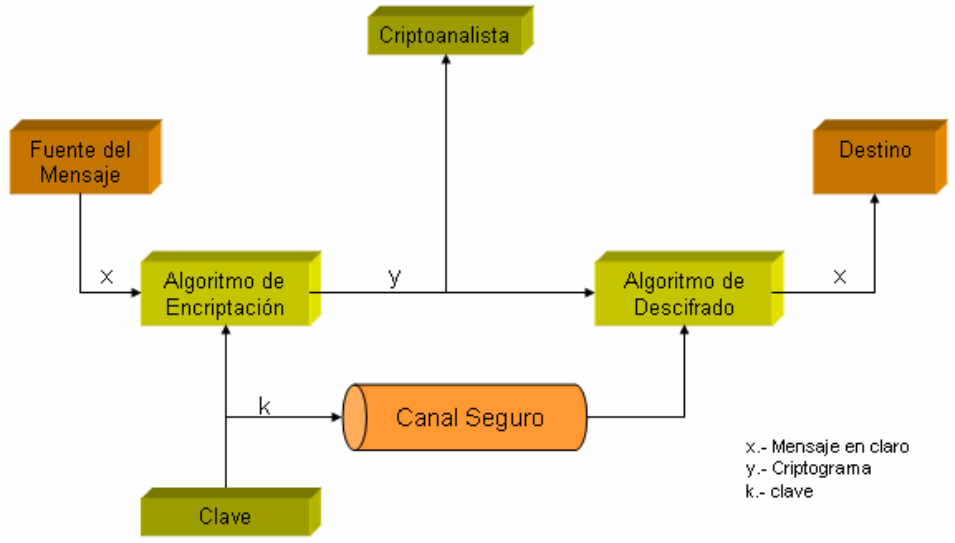


Figura 3.6 Sistema Típico de Encriptación Simétrica

En la tabla 3.6 listo los algoritmos de encriptación más famosos y más utilizados. Se recomienda consultar dichos algoritmos en libros de criptografía, para comprender a detalle su utilización, sobre todo porque en el capítulo siguiente se implementarán

algunos en .NET para tratar de explotarlos al máximo, y es recomendable tener conocimientos básicos del modo de operación de estos algoritmos para su mejor aprovechamiento.

Nombre	Características	Descripción
Data Encryption Standard (DES)	Clave de 56 bits. El estándar del gobierno de EUA hasta 1998, sin embargo perdió fuerza al ser vulnerado. Es relativamente lento.	DES utiliza una clave de 64 bits, de las cuales sólo 56 bits son utilizadas para la encriptación. El algoritmo transforma 64 bits de mensaje, y realiza sustituciones y transposiciones con un total de 16 iteraciones.
3DES	Realiza tres veces la operación del DES, por lo que se tiene un equivalente a una llave de 168 bits. Mucho más seguro que el DES, altamente usado, algo lento.	Muy rápido y es el estándar más aceptado y utilizado para comunicaciones.
Advanced Encryption Standard (AES)	Diferentes longitudes de claves, el último y más robusto estándar utilizado por el gobierno de de EUA.	AES maneja claves de 128, 197 y 256 bits, proveyendo alrededor de 10^{77} posibilidades de clave, por lo que es invulnerable a los ataques de fuerza bruta. Se basa en métodos de topología numérica y transposición vectorial.
Internacional Data Encryption Algorithm (IDEA)	Clave de 128, requiere licencia para uso comercial.	Es un algoritmo que utiliza claves de 64 bits, y divide el mensaje en bloques de igual tamaño. Es rápido y efectivo y se utiliza en algunos algoritmos de encriptación comerciales como Pretty Good Privacy.
Blowfish	Clave de diferente longitud, algoritmo de uso libre, muy rápido y efectivo.	Es un algoritmo que surgió como opción a DES e IDEA. Utiliza una clave de 32 a 448 bits, muy aceptado debido a su rapidez. Se utiliza como parte de 150 productos, entre los que se encuentra OpenBSD y el kernel de Linux.
RC4	Clave de diferentes longitudes, cifrado continuo, efectivo en dominio público.	RC4 es un algoritmo que utiliza diferentes longitudes de claves. Realiza la operación XOR con la clave y el texto original para generar criptogramas. Es de amplia aceptación, aunque está quedando en desuso.

Tabla 3.6 Los algoritmos simétricos más utilizados.

3.4.5. Cifrado Asimétrico

También conocido como cifrado público, se puede considerar como el más grande y más importante avance en la historia de la criptografía. Dejando de lado las técnicas de

permutación y transposición, cuando llegó el avance de la máquina de encriptación de rotor, se buscaron técnicas mucho más sofisticadas.

Para ello, se buscó un sustituto y se encontró en las matemáticas puras. Esto derivó en el concepto más importante de la criptografía moderna: las claves privadas y públicas.

El principal problema que se resuelve con respecto al cifrado simétrico es la comunicación de la clave. Esto se debe a que si se quiere distribuir la clave de manera masiva, se tiene que hacer pública y si se hace pública, esto rompe con toda la seguridad dada por el encriptamiento simétrico.

El esquema del cifrado asimétrico tiene 6 elementos fundamentales:

- Mensaje en claro.- es el mensaje original que se desea transmitir. Éste se alimenta al sistema de encriptación.
- Algoritmo de encriptación.- el algoritmo realiza varias sustituciones y transformaciones al mensaje.
- Clave Pública y Privada.- este es el conjunto de llaves que han sido seleccionadas para que si una ha sido utilizada para la encriptación, la otra se utilice para el desciframiento.
- Criptograma.- este es el producto del algoritmo de encriptación. Depende del mensaje en claro y de la clave secreta. A la vista parece texto sin sentido y es virtualmente inviolable sin la clave (en teoría).
- Algoritmo de descifrado.- generalmente es el algoritmo de cifrado pero hecho de manera inversa. Si es exitoso, dará el mensaje en claro original.

Los pasos esenciales son:

1. Cada usuario genera un par de llaves para ser usadas en el proceso de encriptación y descifrado de los mensajes
2. Cada usuario coloca una de las dos claves en un registro público. Esta es la clave pública, la que queda resguardada es la clave privada. Generalmente se guarda una un "llavero" de todas las claves que se han obtenido de los diferentes usuarios.
3. Si Alex, quiere mandarle un mensaje a León, Alex encripta el mensaje utilizando la clave pública de León.
4. Cuando León recibe el mensaje, ella lo descifra utilizando su clave privada. Ninguna otra clave funcionará, ni siquiera la de Alex.

Todos estos pasos los podemos apreciar en la figura 3.7 podemos apreciar estos pasos de manera gráfica:

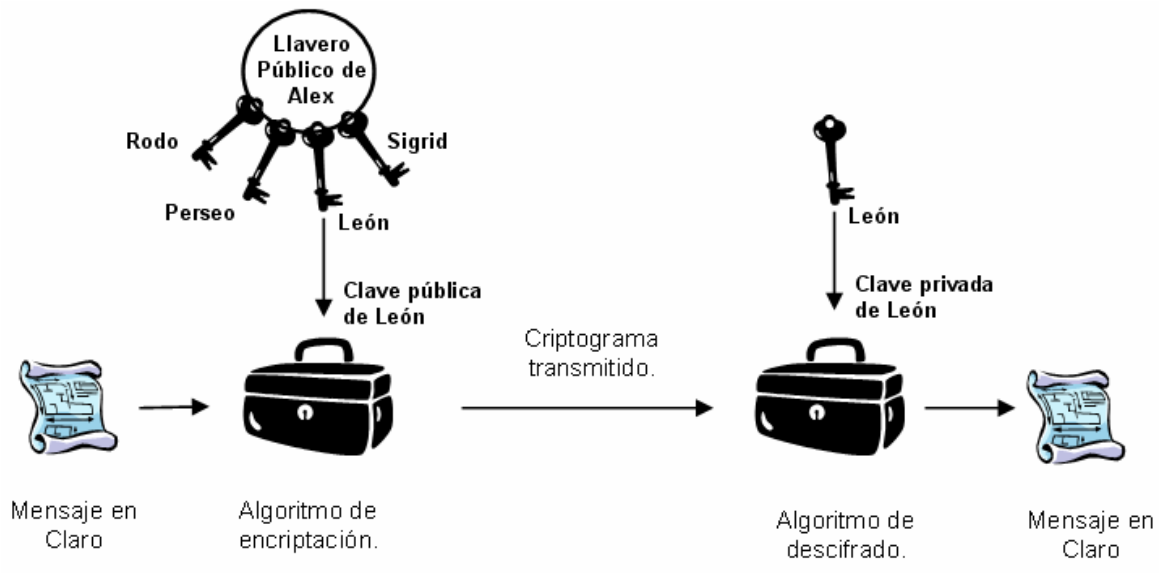


Figura 3.7 Pasos para un encriptado asimétrico

¿Cómo se produciría un ataque? Ya vimos que en cuento al sistema de encriptación simétrica se refiere, se tiene que proteger la clave. En el caso de la encriptación asimétrica, podemos contar con que no se necesita un canal seguro. Esto se refleja en que el atacante, podrá obtener la clave pública del destinatario y el criptograma resultado de ésta. Sin embargo, no podrá descifrarlo y su labor es encontrar la clave privada del destinatario.

Este sistema lo podemos observar con detalle en la figura 3.8.

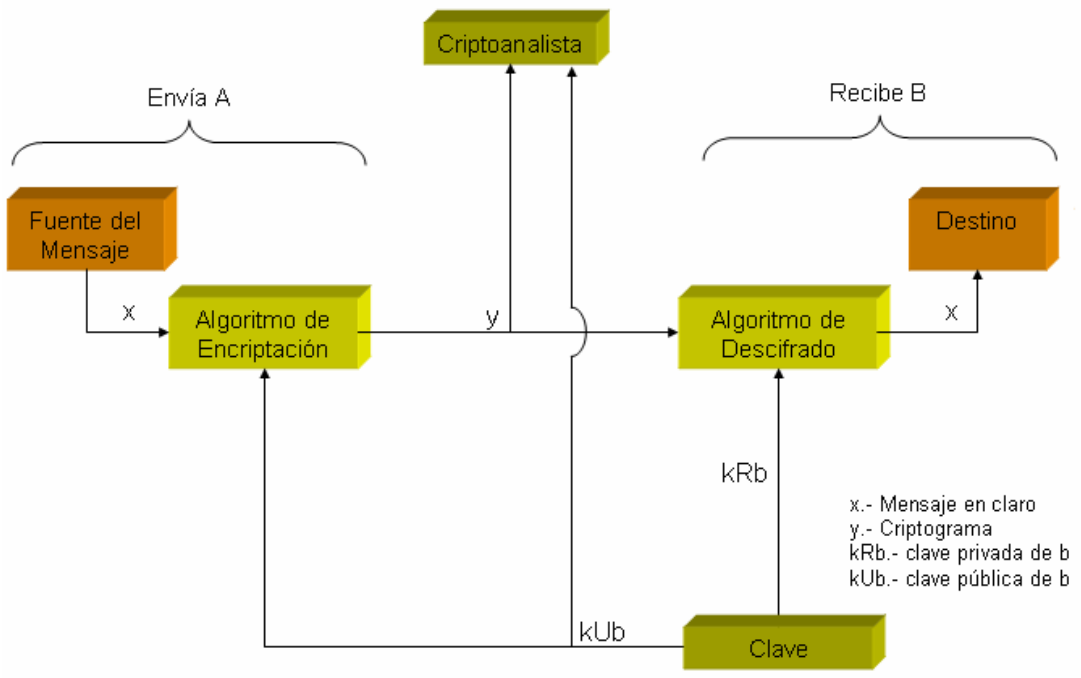


Figura 3.8 Sistema Típico de Encriptación Asimétrica

Es importante mencionar que tenemos la seguridad de que sólo el destinatario puede tener acceso a la información, ya que sólo él tendrá la clave correcta para descifrar el mensaje. Por lo mismo, este es el principio básico que provee a este tipo de encriptación de su principal virtud, que es la autenticación del destinatario. Esta característica es muy explotada por las firmas digitales, que se usan actualmente para autenticar de quién vienen los mensajes.

En la tabla 3.7 podemos apreciar los más usados y famosos sistemas criptográficos:

Nombre	Características	Descripción
Rivest-Shamir-Adleman (RSA)	Clave de diferente longitud, el estándar para encriptación de clave publica.	RSA es un algoritmo asimétrico que se basa en la teoría de residuos. No existe criptoanálisis conocido más que la aproximación por fuerza bruta. Se recomienda que se utilicen claves de al menos 768 bits ya que se ha visto que claves menores son vulnerables. Es muy lento y se utiliza para el intercambio de claves para IDEA y DES.
Diffie-Hellman	Clave de diferente longitud, usada para establecer conexiones seguras.	Este fue el primer algoritmo implementado para el encriptamiento asimétrico. Se utiliza para intercambio de claves en aplicaciones como SSL, IPSec, etc.
Curvas Elípticas	Clave de diferente longitud, muy lenta para su implementación amplia.	Las curvas elípticas se basan en las ecuaciones de la forma $y^2=Ax^3+Bx^2+Cx+D$. Bajo ciertas circunstancias estas curvas tienen un par de enteros que formas un grupo Abeliano finito. Su robustez es importante y su contribución es que se pueden tener claves mucho más cortas que son mucho más seguras, sin embargo su lentitud y gran cantidad de cálculos ha frenado su expansión.

Tabla 3.7 Los algoritmos asimétricos más utilizados

4. Software Seguro

Una vez estudiados todos los conceptos básicos que necesitaremos para comprender lo que es la elaboración de software seguro, estamos en tiempo de empezar a desarrollar la parte central de este trabajo.

La elaboración de Software Seguro no se refiere solamente a métodos de programación, ni tampoco a análisis de amenazas. Como ingenieros tenemos la responsabilidad de coordinar proyectos y actuar como los propietarios de la idea. La programación entonces queda como parte de todo un proceso de elaboración de software, por lo que ya no podemos solamente enfocarnos en esta parte para poder argumentar que hemos creado una aplicación segura.

Para ello, primero exploraremos la ingeniería de software, para rediseñar el ciclo de vida a fin de que éste nos dé como resultado un producto seguro. Es muy importante resaltar esta parte pues tenemos bajo nuestra responsabilidad el diseño de todo un proyecto que cumpla con los requerimientos del cliente de una manera eficiente. Por lo mismo, la seguridad parte desde cómo planeamos nuestro proyecto, y cómo nosotros debemos ser los primeros concienzados de la importancia de no sólo seguir una metodología, sino de también lograr permear a nuestro grupo de trabajo.

El siguiente paso, será analizar los buenos hábitos de programación, ya que un código que no se comprende, es un código que no es seguro, pues no puede ser modificado, ni analizado, ni mantenido, etc. Con ello entraremos de lleno a un breve vistazo a la programación eficiente, que es parte del software seguro debido a que un buen proyecto de software estará siempre basado en que sea eficiente y no sólo haga las cosas bien, sino de una manera eficaz.

Avanzaremos con esto a un análisis de las amenazas más importantes que nos ocupan en nuestros días, como son los desbordamientos de búfer, las inyecciones SQL, entradas maliciosas, etc., para tener un panorama general de cuáles son los problemas que debemos atacar.

Finalmente, analizaremos algunas técnicas para detener estas amenazas, para preparar el camino para que .NET nos ayude con sus nuevas características de seguridad, basándonos en el lenguaje de programación más utilizado del mundo, Visual BASIC.

4.1. Ingeniería de Software

El software seguro es parte de la Ingeniería de Software desde el momento en que los responsables de que éste sea programado somos los ingenieros. Al estar a cargo de los proyectos, cae dentro de nuestra responsabilidad que el software producido tenga todas las características que se marcan y además debe ser seguro y no depender solamente de la seguridad que nos pueda proporcionar la red.

Analicemos, cómo la ingeniería de software debe sufrir cambios para no tratar a la seguridad como un tema más, sino como parte fundamental y básica para su desarrollo.

4.1.1. Historia

Cuando las computadoras empezaron a existir, la programación era completamente empírica. Los programadores eran los mismos que creaban los equipos de cómputo, y con ello satisfacían sus necesidades. Una vez que la computadora empezó a expandirse para el uso científico, estas personas tenían la responsabilidad de generar sus propios programas, por lo que creaban sus propios algoritmos, y se encargaban de utilizarlos correctamente.

Sin embargo, los componentes se empezaron a volver más baratos y más pequeños, y con la entrada del circuito integrado, las computadoras empezaron a emigrar de los laboratorios a las empresas. Es así como por primera vez, el software creado por algún programador, no sería usado por él mismo, o por alguien que estuviera empapado de los conocimientos en computación suficientes para operarlos correctamente. Sin embargo, no era de tanta importancia, pues ya que eran tan pocas se podía tener gente capacitada que estuviera en las empresas verificando su correcto funcionamiento y uso.

Así llegamos a la mitad de los años setenta, donde viene el auge de las computadoras con la invención de nuevas tecnologías, de la mano de la casa Intel, que al introducir su familia de procesadores 80xx logra que los precios bajen de manera estrepitosa, por lo que ya todas las empresas se empiezan a equipar de computadoras.

Ante la gran cantidad de equipos era imposible seguir con especialistas que manejaran todas las máquinas, por lo que el usuario común empezó a llegar a ellas. Por lo mismo el software que se implementara para esas computadoras, ya no podía ser como era antes, ahora había nuevas necesidades. El software debía ser lo suficientemente amigable para el usuario sin perder su eficiencia, además de cumplir con nuevas características que eran necesarias para cualquier sistema ingenieril.

De hecho, el término "Ingeniería de Software" fue acuñado hasta finales de los años 50, cuando los programadores se empezaron a topar con que se necesitaban programas mucho más grandes, para los cuales todas las aproximaciones y soluciones propuestas por los físicos y matemáticos (como estructuras de datos, algoritmos, etc.) ya no eran

suficientes para generar un gran sistema de información. Se necesitaba una solución del tipo ingenieril, donde primero se tiene un acercamiento al problema y luego se utilizan o implementan herramientas y técnicas estándar para su solución.

Otro problema que forzó a que la Ingeniería de Software cobrara mayor importancia, era que los proyectos de software, ya no eran sólo programados por una persona, sino por varios, ya que se necesitaba que diversos especialistas aportaran sus conocimientos. Esto motivaba que se llevara una metodología para que los módulos programados fueran fácilmente ensamblados, y es cuando la parte ingenieril tomaba más importancia que la parte de programación, es decir, el diseño del software se volvió altamente importante.

Además, los proyectos de software ya no eran cuestión de unas cuantas semanas, sino que se volvieron proyectos de meses (inclusive años), y en estos meses, el personal podía abandonar el proyecto por diversas razones, por lo que si un nuevo miembro entraba al equipo, no se contaba con la documentación correcta, y lo peor del asunto, era que muy probablemente, sólo quien estaba a cargo de ese módulo, sabía que estaba haciendo, por lo que algunas veces se tenía que rediseñar y empezar de nuevo. Esto definitivamente no era lo mejor para el proyecto.

Es ante toda esta problemática, que la ingeniería de software nos introduce sus métodos para el desarrollo de software como una rama más de la ingeniería. Sin embargo, cuando nació Internet, las cosas dieron un giro muy brusco, pues ya no teníamos el ambiente seguro del usuario que no tenía ninguna intención de destruir nuestro sistema, sino que ahora con la incursión de los hackers, tenemos que cuidar el software que desarrollemos, para que no dependamos sólo de la seguridad que nos otorga la red. Por lo mismo, los métodos de ingeniería de software sufren un cambio sustancial para la generación de software seguro.

4.1.2. Cualidades de un Sistema de Información

Todos los sistemas de información deben presentar las siguientes cualidades para considerarse un producto de calidad.

- Correcto.- un programa es funcionalmente correcto si se comporta como debe de comportarse cuando se utiliza como se debe. La cualidad es absoluta, si tiene una falla que no cumpla con las especificaciones, se considera que no es correcto.
- Confiable.- es confiable si el usuario puede depender de él. Esto es característico de todas las ramas de la ingeniería, así como el conductor que utiliza un puente sin temor para cruzar un barranco, así debe ser nuestro software.
- Robusto.- se considera que es robusto si se comporta de una manera aceptable cuando no se utiliza como se debe. Con esto nos referimos a como

reacciona cuando es utilizado de manera incorrecta, se tienen entradas erróneas, o se sufre algún ataque.

- Eficiente.- cualquier producto ingenieril se espera que cumpla con un cierto desempeño. Un producto de software se considera que tiene un buen desempeño si utiliza los recursos de manera económica.
- Amigable.- se considera que es amigable un producto de software cuando el usuario común puede utilizarlo de manera correcta sin necesidad de capacitación previa. Esto definitivamente no está peleado con el buen desempeño del sistema.
- Verificable.- si las propiedades de un sistema son fácilmente verificables entonces se dice que el sistema es verificable. Por lo mismo es necesario mantener un diseño modular, un uso estricto de estándares prácticos y además una elección de lenguaje correcto.
- Mantenible.- se utiliza este término para referirnos a qué tan sencillo es actualizar la aplicación, no sólo para resolver defectos, sino para añadir características que no estaban planeadas en el software original. Existen grandes rasgos 3 tipos de mantenimiento:
    - Mantenimiento Adaptativo.-se realiza cuando el software debe ser actualizado para ser compatible con otras versiones de software u otros productos que no estaban planeados desde el principio.
    - Mantenimiento Correctivo.- se presenta cuando el sistema tiene errores o fallas. Se conocen generalmente como parches.
    - Mantenimiento Perfectivo.- este tipo sirve para añadir características nuevas al software, así como nuevos diseños y propiedades.
- Reparable.- se considera que el software es reparable si la corrección del mismo se puede realizar con poco esfuerzo. En muchas áreas de la ingeniería, el costo de la producción decrece tanto, que a veces es mejor comprar de nuevo el producto que repararlo, este es el estado que se preferiría para el software.
- Evolutivo.- el hecho de que un software evolucione, tiene que ver con que se pueda utilizar el código como base para generar nuevas versiones del producto. Esta es una característica del producto y del proceso, por lo que también debe de ser sencilla su evolución.
- Reutilizable.- se considera que el software es reutilizable cuando sus módulos, funciones y procesos pueden utilizarse en otros productos. En la evolución utilizamos gran parte del producto, mientras que en la reutilización, utilizamos partes del sistema para generar uno completamente nuevo.

- Portabilidad.- esta es una característica que ya he analizado. Como lo señalé en el Framework .NET, la portabilidad es la capacidad de que el sistema se pueda ejecutar sin importar la plataforma que estemos utilizando.
- Ser Entendible.- es característica interna referente a las especificaciones del sistema y como está diseñado. Es importante que sea entendible, pues esto propiciará que sea de fácil evolución y reutilizable.
- Interoperabilidad.- el poder comunicarse con diferentes productos es una característica muy deseable, aunque muchas veces no es posible. Un ejemplo claro es Microsoft Office que trabaja de manera aceptable entre sus mismas aplicaciones, pero es muy difícil verla operar correctamente con otras aplicaciones aunque también sean productos de Microsoft.
- Productividad.- otra de las características internas más importantes, ya que con este concepto nos referimos a que tanto se produce y que tan bien se produce el sistema, en términos de tiempo, eficiencia y calidad. Ya que la productividad es resultado de las productividades del personal que labora, es importante disponer de diferentes técnicas y herramientas que permitan elevarla (como el IDE de .NET).
- Visibilidad.- un proceso de desarrollo de software es visible si todos los pasos y su estado actual está documentado de manera clara y concisa. Esto no sólo sirve para que el líder de proyecto se dé cuenta del estado del proyecto, sino para generar informes a los clientes del estado de su producto.
- Entrega a Tiempo.- el software es un producto profesional y tiene que ser entregado con un 100% de funcionalidad en la fecha en que se especificó que se entregaría, por lo mismo, todo el Ciclo de Vida, debe ser diseñado con el profesionalismo de cualquier producto de ingeniería.

Es decir, los proyectos de software ya no son el producto de la inspiración de algún genio, sino ahora son producto de un sistema ingenieril, que como tal, debe de cumplir con todas las expectativas.

4.1.3. Ciclo de Vida de los Sistemas de Información

Existen métodos que sugieren pasos y etapas para la creación de software y que son los más aceptados. Después de repasarlos tomaremos el último método y lo adaptaremos a nuestras necesidades para elaborar software seguro.

a) Modelo de Cascada

Como su nombre lo indica el modelo es una serie de pasos "en cascada" uno tras otro que deriva en uno más. Este fue el modelo más empleado ante el surgimiento de la

ingeniería de software allá por los años cincuenta y tuvo una vigencia hasta finales de los setenta, donde el tamaño de los productos de software lo rebasó.

Las variantes que se utilizan son muchas, sin embargo, las etapas y el modelo en sí es el mismo, por lo que podemos darnos una idea muy clara si vemos con atención la figura 4.1. La figura nos muestra como cada etapa está seguida de otra, cabe resaltar que este modelo no toma en cuenta que los requerimientos pueden cambiar, y que es sumamente inflexible.

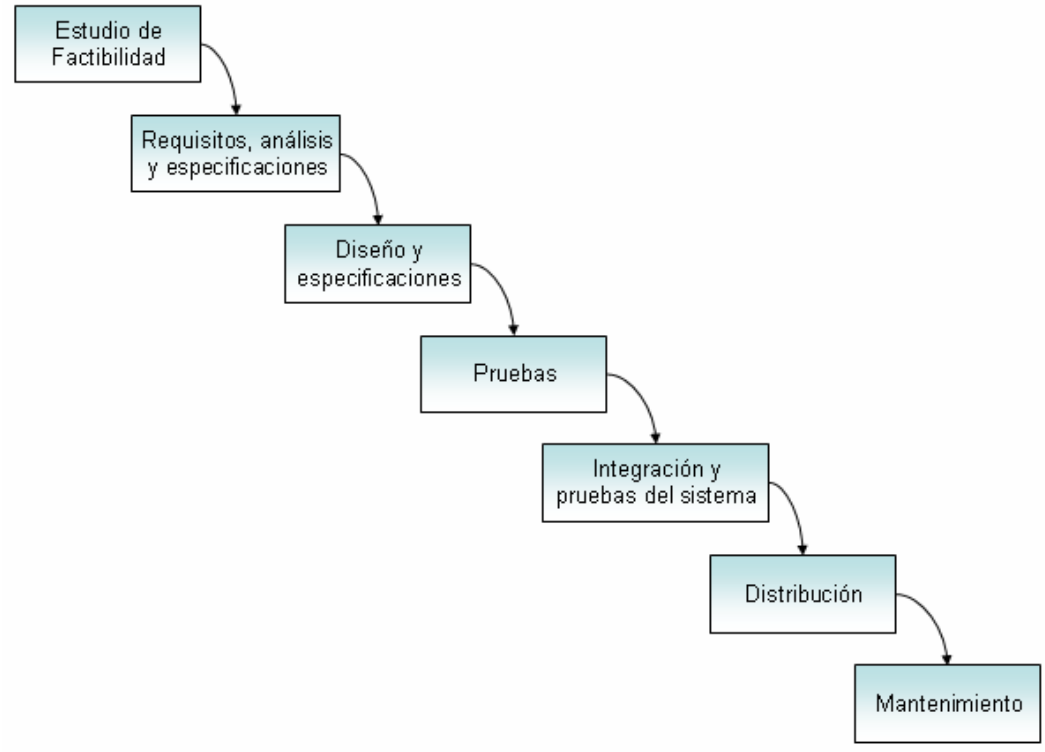


Figura 4.1 El modelo en cascada.

b) Modelo Evolutivo

El modelo evolutivo surgió poco después del modelo en cascada. Se basa en él, pero con la particularidad de que el evolutivo se basa en un principio dicho por Brooks (1975), que dice: "hazlo dos veces". Esto lo podemos traducir a un beta, que es el método empleado actualmente. Se libera una versión preliminar y se retroalimenta, con ello, se regresa a todo el modelo, corrigiendo cada una de las partes que salieron mal y que deben ser corregidas (Figura 4.2)

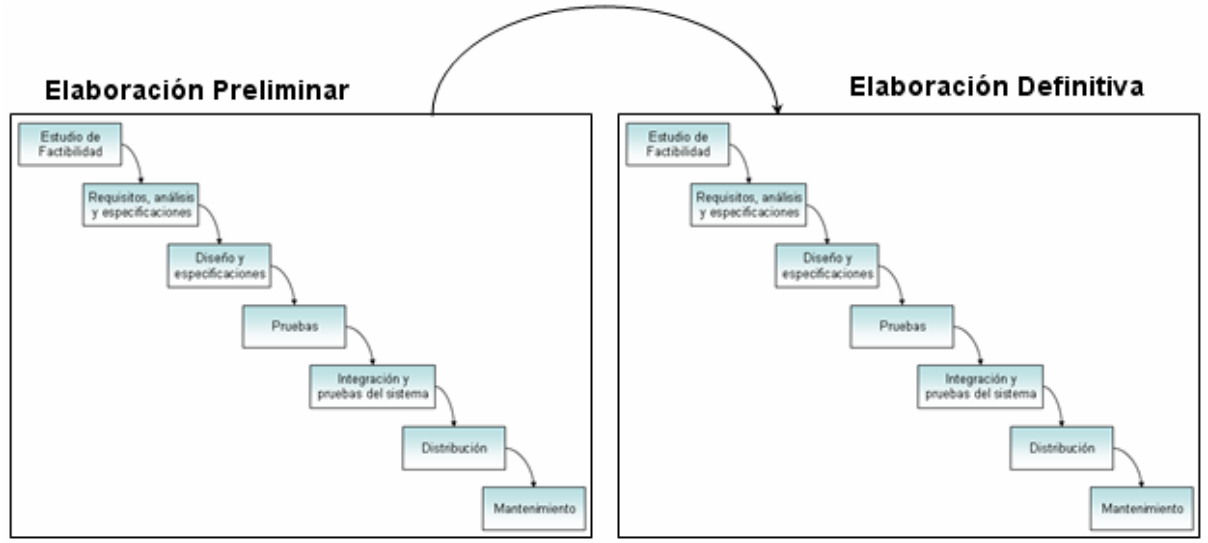


Figura 4.2 El modelo evolutivo.

c) Modelo de transformación

La idea general en este modelo es que se puede ver al proceso de creación de software como una secuencia de pasos que gradualmente transforman una especificación en una implementación. Esto significa que primero partimos de requerimientos informales, y los vamos concretando, generando un diseño que cumpla con los requerimientos formales (figura 4.3), así podemos pasar a la optimización en donde tenemos un proceso similar al anterior, donde vamos generando las implementaciones y comparándolas con las especificaciones finas, hasta que se cumpla el total de las especificaciones del cliente. En esta parte, es importante llevar la documentación del proyecto, así como tener contacto constante con nuestro equipo de trabajo y el cliente para hacer las modificaciones más pertinentes.

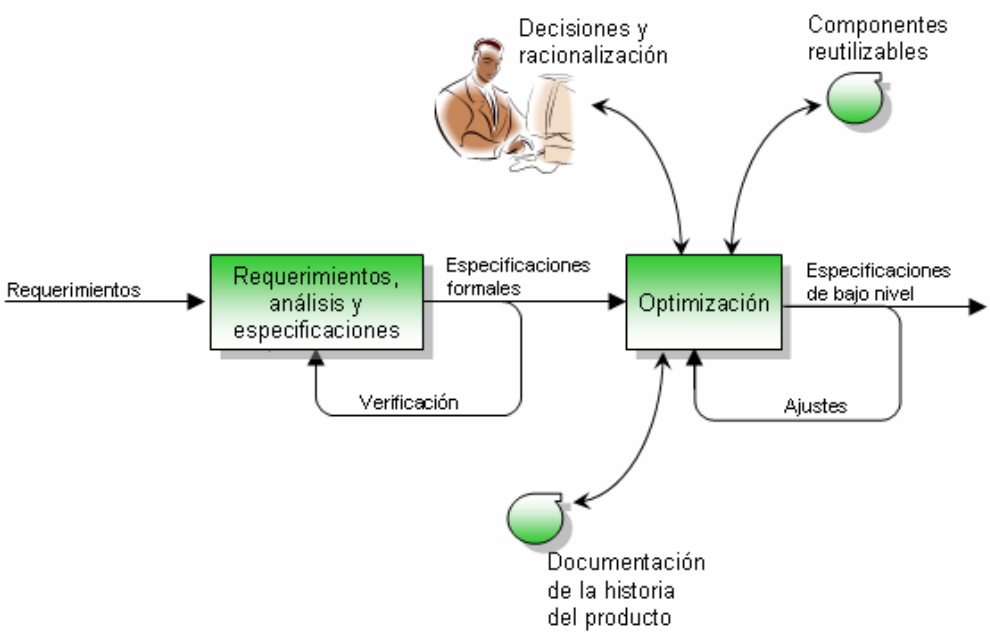


Figura 4.3 Modelo de Transformación

d) Modelo Espiral

Uno de los modelos más vanguardistas, es el modelo en espiral, donde no se practica una serie de pasos secuenciales, sino más bien un ciclo, donde todas las partes se visitan de manera continua sin excepción. De hecho, su éxito radica en que puede cobijar a cualquier modelo, esto es, que en una espiral podemos tomar el modelo en cascada y luego seguir con uno de transformación.

Uno de los conceptos nuevos que maneja, es el riesgo. Como estudio riesgos se comprende el análisis de todo posible incidente que se pueda convertir en amenaza y pueda atentar contra el buen desempeño de nuestro sistema. Es así como por primera vez, la seguridad tomaba un papel más importante en la elaboración de software, ya que podía caber dentro de esta etapa.

Como podemos apreciar en la figura 4.4 el modelo en espiral consta de 4 etapas (en 4 cuadrantes) que son análisis de riesgos, planeación, desarrollo y determinación de objetivos. Cada uno nos da un prototipo que nos servirá como base para comenzar una nueva espiral.

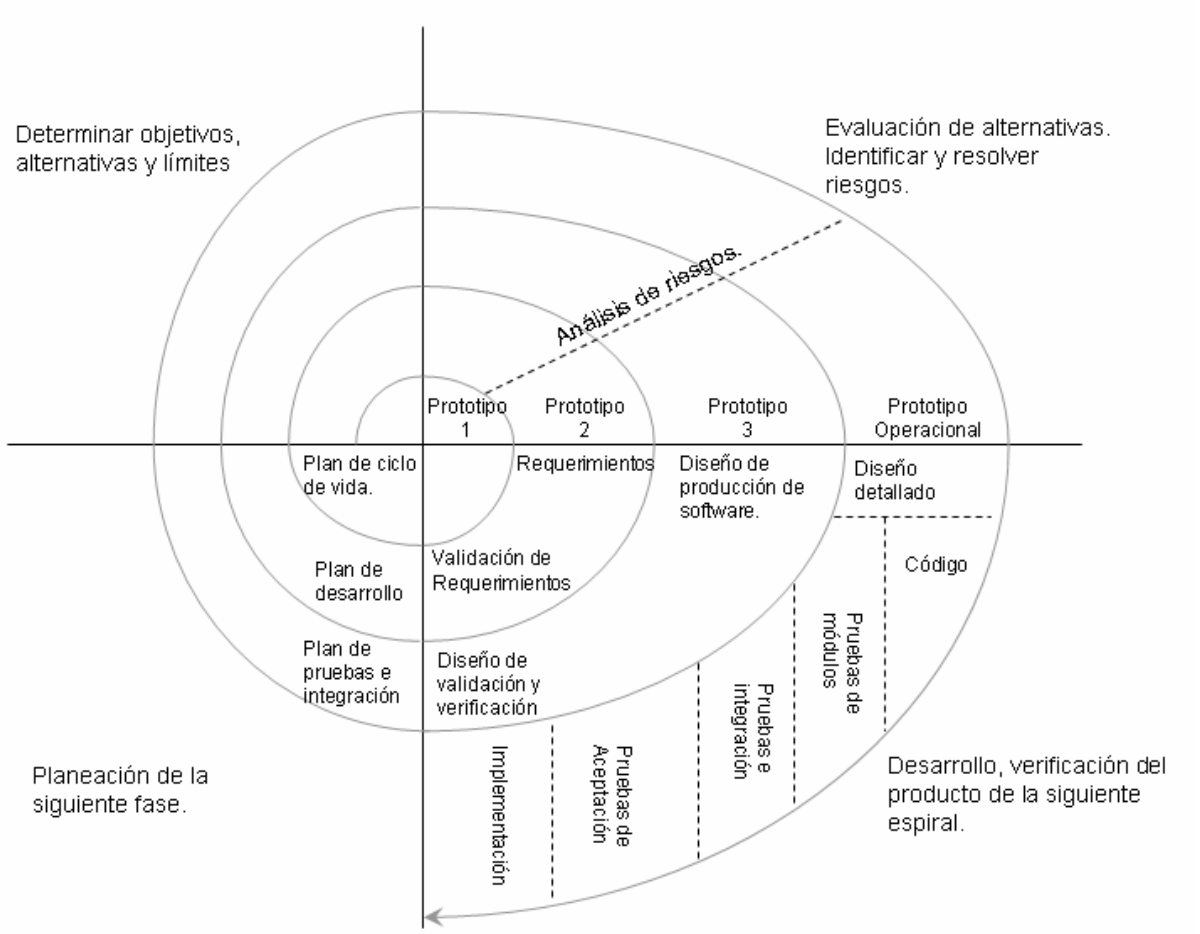


Figura4.4 Modelo Espiral

e) Modelo Mixto

En la actualidad muchas de las compañías que desarrollan software, manejan un modelo que reúne las mejores características de todos los modelos que hemos visto, en general manejan 4 etapas. Estas etapas contienen todos los pasos que se deben llevar a cabo para completarla. Lo más importante de este modelo, es que las etapas no son una tras otra, sino que se realizan de manera traslapada, dando una retroalimentación continua proporcionada por los diferentes equipos que van terminando en tiempos diferentes y ayudan en las diferentes etapas del proyecto (figura 4.5).

Este tipo de modelo es empleado en empresas con proyectos de software cuya duración puede llegar a ser de años, por lo que diferentes equipos participan en la elaboración del sistema. De hecho, cuando la labor de análisis queda terminada, se empieza con el análisis del próximo proyecto, que se traslapa de igual manera con el mantenimiento del anterior, por lo que podemos tomarlo como un modelo en espiral mejorado.

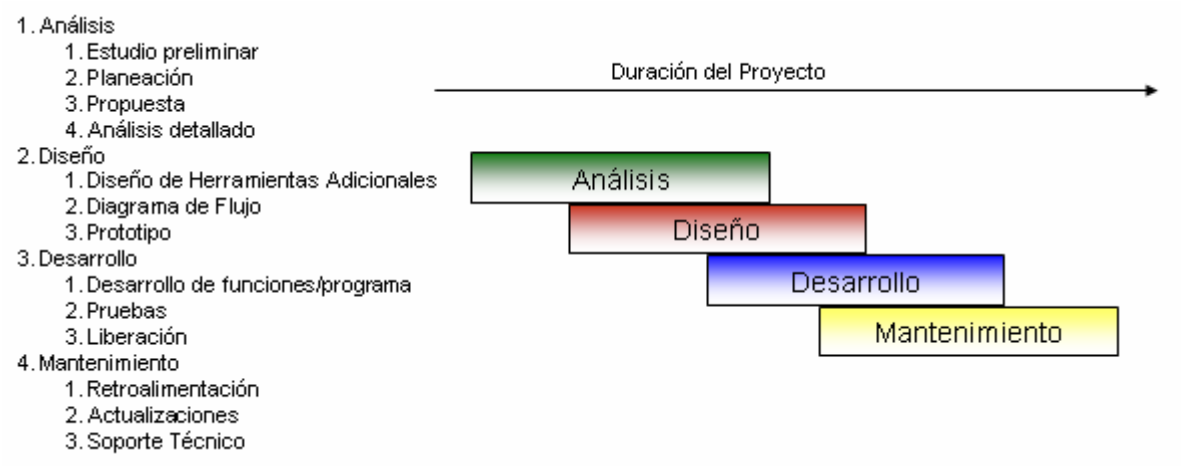


Figura 4.5 Modelo de Traslape Clásico

4.1.4. CVSI Seguro

El ciclo de vida de los sistemas de información ahora toma otro giro, pero no de forma, sino de fondo, esto es, que se respetarán los modelos que se empleen, pero serán modificados para albergar los nuevos conceptos sobre seguridad que se puedan presentar.

Es importante establecer que no importa que modelo se utilice, estas variaciones, pueden aplicarse a cualquiera de los métodos analizados y a cualquier otro, mientras tome en cuenta las fases básicas de la ingeniería de software:

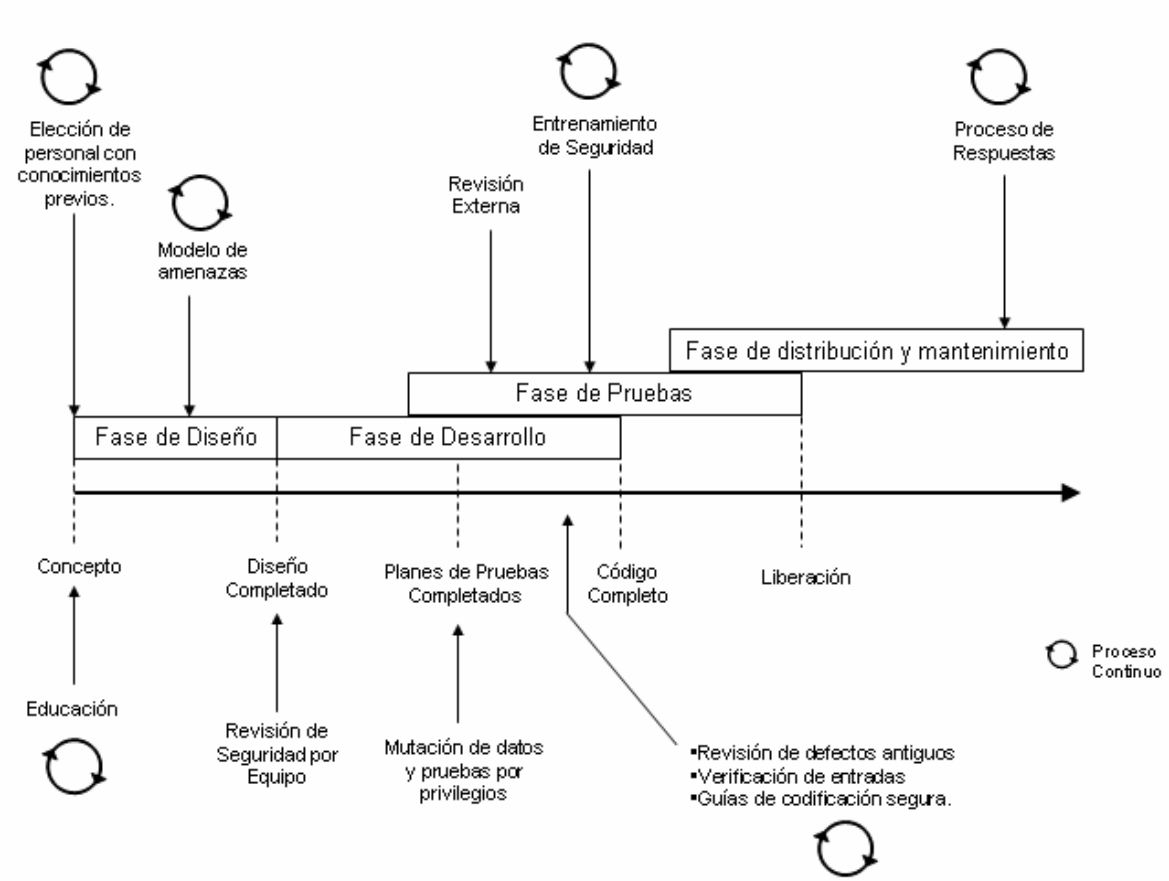


Figura 4.6 CVSI Seguro

a) Fase de Diseño

Durante la fase de diseño tenemos diversas consideraciones que anexar a nuestro modelo de ciclo de vida. Una de las más importantes es, sin lugar a dudas, la elección de nuestro equipo de trabajo.

El equipo de trabajo, debe contar con conocimientos previos de seguridad. Así como es importante que los miembros del equipo sepan programar y manejar lenguajes de programación, también es importante que sepan pensar como programar de manera segura, teniendo bases sólidas de seguridad informática. Esto es preciso, pues desde este momento, debemos pensar en no tener que llegar a corregir el producto.

De igual manera el equipo de trabajo se debe mantener al día con las últimas noticias sobre seguridad. Ésto debe ser continuo y realizado como disciplina, también se debe tener tiempo para la toma de cursos y de sesiones generales, para la discusión de asuntos de seguridad no relativos al trabajo actual, así habrá una fase de alimentación continua y se tendrán más precauciones a la hora de programar y de diseñar los futuros sistemas.

Otra de las partes fundamentales del desarrollo radica en el modelado de amenazas que se refiere a anticipar desde el diagrama de flujo, dónde radican las posibilidades de un ataque. Esto lo analizaré más adelante, debido a su gran importancia y complejidad.

Finalmente recordemos que se deben escoger los defectos y vulnerabilidades que se atacarán. En un producto ideal, se querrían solucionar todos los problemas de un software y pulirlo de tal manera que el producto final no tuviera errores. Esto, aunque es lo deseable, no es posible, ya que nosotros debemos dedicarnos a proteger todo el sistema, mientras un atacante puede escoger donde atacar y escoger siempre el lugar menos protegido. Si nosotros aumentamos la robustez de nuestra aplicación tal vez generemos un sistema muy seguro, pero sin lugar a dudas, habremos gastado mucho tiempo y dinero, y esto aumentará el costo del producto y el tiempo de producción. Recordemos una de las máximas de los sistemas: "características, costos o entrega a tiempo, escoja dos".

Generalmente los defectos que se cubren son los más probables que puedan ocurrir. Es como si asegurámaramos una alberca contra incendio (puede que suceda, pero no es muy probable), debemos organizar los errores y posibles vulnerabilidades de una manera jerárquica para poder de esta manera identificar cuales son los más importantes y como debemos de enfrentarnos a ellos.

a.1) Modelado de Amenazas

Para el modelado de amenazas se requiere que se arme un equipo de modelado. Es importante que los que estén trabajando en este equipo, sean los más capacitados, y es también importante que no sean todos los del equipo, pues se debe de mantener el orden, sin necesidad de ignorar a algunos, esto claramente no se puede lograr en grupos grandes.

Después de ello, hay que basarse en el diagrama de flujo, donde veremos cómo se debe comportar el sistema. Para ello debemos de entender perfectamente como desea trabajar la aplicación. Para ello nos podemos basar en el diagrama de flujo de datos que ya debe estar hecho. Tomemos como ejemplo un sistema sencillo hecho en el Laboratorio de Microsoft Research. Este proyecto deja que el alumno pueda revisar una simulación de cómo sería su cambio de plan de estudios, ante la revisión que ocurre cada 10 años en la Facultad de Ingeniería.

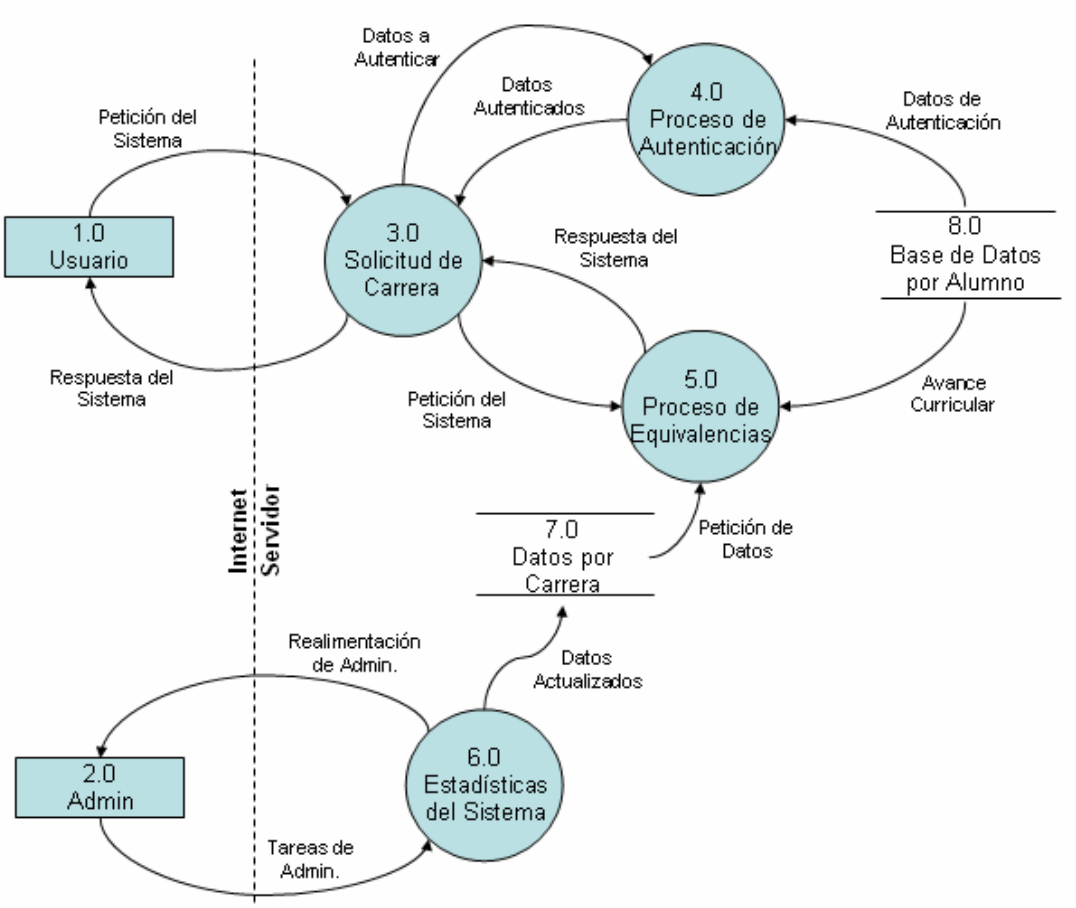


Figura 4.7 DFD de Simulador de Equivalencias

Como podemos apreciar, el sistema es bastante simple, pues a pesar que realiza varias consultas todas son sobre dos bases de datos y el algoritmo central, es un conjunto de procesos (5.0). Es notorio, que fuera de la consulta de datos cuando se hace una petición al sistema, no tenemos otro posible problema de seguridad (ya que todo depende de la manipulación de los datos de entrada). Por lo cual nos enfocaremos a esta parte (por supuesto que hay más, pero un estudio detallado queda fuera de los objetivos de este trabajo).

Un análisis de las amenazas en este preciso lugar del sistema se debe realizar, pues hemos encontrado dónde se puede realizar un ataque. El modelado de amenazas de esta sección puede quedar de la siguiente manera:

```
Amenaza 1.0
1.0 Ver datos de autenticación confidenciales en su transmisión
      1.1   Tráfico http no está protegido (y)
      1.2   Atacante monitorea el tránsito
            1.2.1 Utilización de un sniffer.
            1.2.2 Monitoreo del tránsito en el router
                  1.2.2.1  Router no está protegido (y)
                  1.2.2.2  Router comprometido
```

```
            1.2.2.3  Averiguar la clave del router
      1.2.3 Switch Comprometido
            1.2.3.1  Varios ataques al switch
```

Como podemos apreciar, al final del 1.1 tenemos entre paréntesis dos posibles causas de la vulnerabilidad, sin embargo, si no se cumplen las dos no habría peligro alguno, sin embargo se anotan por separado para darles un tratamiento diferente si es que lo necesitaran. Ya que tiene acceso el atacante al tránsito de la red, debe de hacer cualquiera de estas tres opciones, ya sea, utilizar un sniffer, o comprometer de alguna manera el router o el switch. Es así como llevamos al último nivel, en el cual se dan las causas por las cuales puede alguien monitorear el router (1.2.2.3), o comprometer al switch.

Una vez que se ha hecho este análisis hay diferentes formas de pasarlos a un informe. Una de las más aceptadas y utilizadas por Microsoft, es el modelo DREAD (**d**año potencial, **r**eproducible, **e**xplotable, usuarios **a**fectados, **d**escubrible). Este tipo de modelo lo podemos apreciar a continuación:

Amenaza 1.0		
Descripción de Amenaza	Usuarios no permitidos pueden ver el contenido del canal	
Objetivo de Amenaza	1.0 <--> 3.0	
Categoría de Amenaza	Exposición de Información	
Riesgos	Daño Potencial: 6	No hay gran daño al ver número de cuenta.
	Reproducible: 10	Es muy común que se realice varias veces.
	Explotable: 7	No es tan explotable, se requieren conocimientos previos.
	Usuarios Afectados: 10	Si el ataque tuviera éxito, todos los usuarios se verían afectados.
	Descubrible: 10	Que tan sencilla es de descubrir la vulnerabilidad
	Promedio: 8.6	
Comentarios	El ataque es probable que se dé a través de un analizador de protocolos (programa de captura de paquetes, que separa los bits significativos, identifica el protocolo, separa el tráfico y deja ver el contenido de cada paquete), pues es un ataque sencillo de realizar. Es un ataque pasivo y económico en términos de tiempo, esfuerzo y dinero.	

Las calificaciones que se les da a cada uno de los riesgos analizados son en un rango de 5 a 10, siendo el 5 una calificación reprobatoria y 10 una calificación muy alta. Este rango de calificaciones lo adopté porque nos recuerda al que tenemos en la escuela, sin embargo, es libre y se puede adaptar a las necesidades de cada quien. Y de igual manera, las calificaciones se otorgan según un juicio experto que puede variar de analista en analista, generalmente se estila que sea una calificación condensada entre todos lo que conozcan acerca de seguridad en el equipo de trabajo.

Como vemos tenemos información completa acerca de las amenazas y de los posibles riesgos que se corren, así como de un reporte completo, que permitirá a los programadores tomar cartas en el asunto y generar código seguro.

Al final de las calificaciones, tenemos el promedio. Con este valor, podremos clasificar todas las amenazas que hemos detectado y así clasificarlas e irlas solucionando según su importancia.

Es así como se deben de tratar una y cada una de las amenazas que se encuentren en el producto. Es importante que todo el equipo participe en ello aunque sea de forma indirecta, pues sólo así se logrará tener un análisis más completo, que por supuesto derivará en software seguro.

b) Fase de Desarrollo

La fase de desarrollo involucra la escritura y depuración del código, por lo que el enfoque corresponde al código y las técnicas con las cuales se realizará esta fase.

Una de las partes más importantes de esta fase es la asignación de los estándares y buenos métodos de programación. Sin estos dos elementos principales es casi imposible que se pueda realizar un software seguro. Por lo mismo se deben definir desde el principio, métodos que se seguirán por todo el equipo de trabajo. Al definir cuales serán las metodologías de trabajo (como por ejemplo, como se trataran las excepciones, o los datos de entrada), se debe tener en claro que no se debe tener una en especial para nuestro equipo de trabajo, esto es para darle universalidad a nuestro código y que cualquiera que ingrese a nuestro proyecto en cualquiera de sus etapas no tenga problemas para comprenderlo y poder modificarlo.

También es importante señalar la retroalimentación en el grupo. Una vez que alguien detecta un error o posible vulnerabilidad, es importante que lo comparta con los demás desarrolladores para que todos aprendan de la enseñanza y puedan tomar cartas en el asunto, generalmente un mismo error, se puede presentar de diferentes maneras en procesos o funciones aisladas de un mismo producto. Por lo mismo es importante que si se hace un descubrimiento de un error posible, todos estén informados al respecto.

La verificación de nuevas entradas se refiere a la revisión del código nuevo por los demás compañeros de trabajo. Esto persigue dos objetivos: el primero es que ante una revisión externa por los demás compañeros, el trabajo y la calidad de código presenta una mejoría; el segundo, es que así aseguramos que el código nuevo entre al producto con la mayor revisión posible.

La revisión de defectos antiguos es también parte importante y fundamental de un buen desarrollo. Para ello, se debe llevar a cabo una revisión cada vez que se haga una función o aplicación, en la que podemos revisar la base de datos de defectos y errores, y así tendremos la seguridad de no caer en los mismos errores que se hayan presentado en aplicaciones similares en otros proyectos. Por lo mismo es importante, que cuando se

presenten errores nuevos se actualice la base de datos, con la forma en que se presentó el error y cómo se corrigió.

Por último, es muy importante contar con una revisión externa. Esta costumbre no sólo sirve para el software, en general, siempre es bueno tener opiniones y ojos frescos y no influenciados por nuestro equipo de trabajo.

c) Fase de Pruebas

Si hay algo importante dentro del ciclo de vida, es sin lugar a dudas la fase de pruebas. Al llegar a la fase de pruebas, se podría suponer que con tantas revisiones, con un buen diseño y modelado de amenazas, el software es casi seguro en su totalidad. Esto es un error muy común, es precisamente en esta fase donde veremos que tan bueno fue nuestro análisis.

En esta fase, generalmente se acostumbra la revisión del funcionamiento del sistema. Esto no es totalmente correcto a la hora de programar software seguro, pues además de la verificación de correcto funcionamiento, se debe probar que el modelo de amenazas fue correctamente implementado. Para ello debemos probar cada una de las posibles fallas del modelo STRIDE:

	Descripción	**Pruebas**
Robo de identidad (**S**poofing Identity)	Obtención ilícita de contraseñas y nombre de usuario.	• Tratar de ver el tránsito en la red. • Tratar de adivinar claves por medio de ataques por fuerza bruta.
Alteración de datos (**T**empering with Data)	Incluye suplantación y modificación maliciosa de datos. Generalmente es por medio de ingreso ilegítimo a la base de datos del sistema.	• Tratar de obtener y reemplazar información. • Crear firmas digitales, funciones HASH, etc y verificar si son aceptadas.
Repudio (**R**epudiation)	Son aquellos ataques donde el atacante, al ser rastreado no puede ser acusado, ya que no existen elementos que lo impliquen.	• Verificar formas de entrar al sistema sin una identidad legítima. • Verificar si existen modelos de seguridad y bitácoras.
Exposición de información (**I**nformation disclosure)	Comprende que usuarios no autorizados puedan ver información para la cual no tienen los privilegios necesarios.	• Intentar ver información que sólo usuarios con más privilegios deberían tener. • Verificar si el sistema nos da información de más que nos ayude a atacar de una manera más correcta.
Negación de servicio (**D**enial of Service)	Es uno de los ataques más comunes y sencillos de realizar. En este ataque se logra negar el servicio para los usuarios legales a través de sobrecargar el proceso, la aplicación o el sitio web.	• Atacar con múltiples peticiones a la vez, para sobresaturar al sistema. • Verificar entradas de datos, para tratar de hacer una terminación no planeada del proceso.

Escalada de privilegios (**E**levation of Privilage)	En este tipo de amenaza un usuario logra tener en su cuenta privilegios que no le corresponden por medio de ataques al servicio.	• Verificar si se puede ejecutar código como datos. • Tratar de tener acceso a otros lugares en memoria por medio de ataques como desbordamiento de buffer.

Tabla 4.1 Tipos de Pruebas para el modelo STRIDE

Todo ello se debe de hacer desde un punto de vista de un atacante, para verificar si el código puede responder de manera correcta a los diferentes tipos de ataques (que sea robusto) por lo tanto, esta es la parte más importante de las pruebas, y esto hace que las pruebas sean fundamentales para que un producto pueda salir al mercado; ya que si no lo podemos probar o tiene demasiados fallos.

c.1) Ataque con Mutación de Datos

Generalmente este tipo de ataques son de los más usados y de los más importantes por probar en esta etapa. La mutación de datos consiste en manipular cualquier tipo de dato que ingrese al sistema, ya sean entradas, contenidos de archivos, los contenedores de información, etc.

Dentro de la mutación de datos hay muchísimas formas de ataques, sin embargo de las más representativas tenemos a las que se pueden apreciar en la Tabla 4.1

Mutación de Datos	Comprometiendo Datos	Contenido	Aleatorio	Contenido aleatorio que puede producir fallas en el sistema.
			Válido e Inválido	Información parcial válida que asegura su aceptación por el sistema, seguida de código arbitrario.
			Fuera de límites	Código que utiliza o pide recursos fuera de los alcances de nuestro programa.
			Signo o tipo erróneos	Datos lógicamente válidos, pero erróneos.
		Tamaño	Más grandes o más chicos	Mentir acerca del tamaño de diferentes entradas.
		Caracteres Especiales	META, SCRIPTS, HTML, etc.	Inclusión de caracteres especiales y utilizados en el código como "/*", "<!—", "//", etc.
	Comprometiendo Contenedor	Cambio de Nombre	Reemplazo de nombre para que se ejecute otro o ninguno.	
		Suplantación	Cambiar el contenido ilegítimamente.	
		Negación de servicio	Negar el acceso al archivo por cualquier medio.	
		Borrado	Desaparecer el archivo.	
		Existencia	Si necesita que un archivo esté libre, lo utiliza.	

Tabla 4.2 Mutación de Datos

d) Fase de Distribución y Mantenimiento

Se podría pensar que en esta fase, todo el trabajo sobre seguridad se ha terminado porque estamos listos para distribuir nuestro producto. En parte es cierto, ya que cuando terminamos por fin la fase de pruebas, estamos asegurando que ya no hay más vulnerabilidades que nosotros conozcamos que comprometan la seguridad de nuestro sistema.

Sin embargo es en este momento, cuando es necesario aplicar la retroalimentación del producto. A este proceso se le puede llamar de respuesta y es el famoso mantenimiento que todos los proyectos de ingeniería deben de poseer. Por lo mismo, es importante que se tengan políticas de cómo reaccionar ante los defectos que nuestro equipo de revisión posea y ante los defectos que los mismos usuarios u otras entidades puedan encontrar en nuestro producto.

Esta política debe de seguir un proceso estable, en donde se actualice continuamente, ya sea por fechas o por temporadas, y se debe hacer llegar a todos los usuarios del producto dando una explicación del porqué de la corrección y como prevenirla.

Finalmente es necesario que los creadores del código sean los responsables de corregirlo; o al menos que las correcciones que se hagan al código sean informadas a detalle a quienes lo crearon, para asegurar así que se aprenda de los errores.

4.2. Programación Segura

Para que un software se pueda considerar seguro debe haber sido diseñado en base a un ciclo de vida seguro. Una de las partes fundamentales para que ese ciclo se pueda cumplir, es la fase de desarrollo, pues es aquí donde nacerá el software seguro.

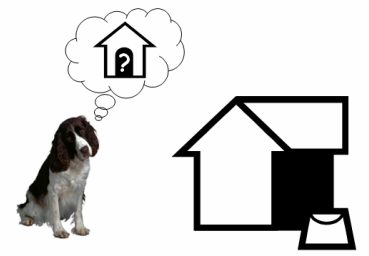
Para que el código sea seguro, se debe de diseñar e implantar de una manera correcta. ¿Qué sucede cuando un código no lo puede entender nadie más que quien lo programó? Por más brillante que sea la implementación, si nadie la entiende, no podemos asegurar que sea seguro. De igual manera aquel código que no cumple con estándares mínimos no se puede considerar un código de calidad y por lo tanto no puede ser seguro.

También es importante considerar que una vez que ya se diseñó cómo trabajará el sistema, los programadores tomen las decisiones correctas en término de eficiencia, de eficacia y sobre todo de buenos hábitos de programación. A esto se le llama programación eficiente y es muy menospreciada por los programadores técnicos o que no han tenido preparación profesional.

Nuestro papel como ingenieros y líderes de proyecto es asegurar que los programadores programen dentro de los estándares básicos y que su código no sólo sea brillante, sino fácil de entender, de mantener y de corregir.

4.2.1. Antes de Programar

Como todo proyecto de ingeniería existen diferentes maneras de resolver un problema. Sin embargo, si algo hemos aprendido a través de nuestros estudios es que primero debemos diseñar lo que vamos a programar. En esta etapa, ya sabemos lo que queremos (de la fase de diseño), ahora toca el turno a "cómo" lograremos.



Definitivamente, un error al no considerar un detalle en una estructura no es tan grave, sólo es tedioso y bochornoso, sin embargo en proyectos más grandes puede ser desastroso.

Para ello entramos en otra fase, la fase de diseño del código. Generalmente nosotros estamos acostumbrados a programar solos o a lo más tres personas a la vez. Por lo que la aproximación que utilizamos puede ser de prueba y error. Tal vez alguno de nosotros, pudo utilizar una aproximación dónde va construyendo un modelo sencillo y lo va modificando para que vaya creciendo. Esta aproximación es parcialmente correcta, sin embargo, como no se planea desde un principio la incursión de otras funciones, comportamientos o excepciones, muchas veces, la codificación se vuelve muy confusa, tediosa y sobre todo larga.

Por lo mismo, es importante tener un buen diseño de código, y analizar muy bien cuáles son los problemas a solucionar, pues es común que se pierda tiempo resolviendo el problema equivocado y lo peor del asunto no es eso, sino que no se resolvió el problema correcto dejando con ello un hueco en el sistema, esto es, una vulnerabilidad.

4.2.2. Programación Eficiente

Antes la programación estaba sujeta a una programación eficiente y optimización del uso de los recursos. Esto se debía principalmente a que, las computadoras no contaban con mucha memoria, ni espacio en disco duro, además de que los procesadores no eran tan poderosos. Hoy en día, con los nuevos lenguajes de programación, los discos duros de alto almacenaje y los ya varios gigas de memoria primaria, han hecho que el concepto de programación eficiente ha quedado un poco en desuso.

De hecho, el tiempo en que se procesan los programas es tan corto, que tendemos a no poner atención en detalles como utilizar la estructura de decisión correcta, o hacer ciclos de más. Sin embargo, en cuanto a seguridad se refiere, el concepto de programación eficiente regresa con un bono adicional. Es importante programar de una manera eficiente para evitar estar en la línea de fuego, esto es, permitiendo que nuestro software sea de alta calidad, optimice los recursos y permita un uso seguro del mismo.

a) Antiguas Creencias

Existen diversas creencias sobre la optimización de código que deben ser desechadas para tener así un punto de inicio claro sobre lo que es la optimización de código, aquí enlisto algunas de ellas.

a. *Menos líneas, más eficiencia*

No hay nada más absurdo que esta creencia. Muchos programadores novatos se ufanan de escribir en menos líneas de código un programa. Ellos creen que entre menos líneas, más eficiente se vuelve su algoritmo. Esto es falso, como siempre todo dependerá del lenguaje de programación, así como de la rutina que se estará ejecutando.

```
a(1) = 1
a(2) = 2
a(3) = 3
a(4) = 4
a(5) = 5
a(6) = 6
a(7) = 7
a(8) = 8
a(9) = 9
a(10) = 10
```

```
For i = 1 To 10
    a(i) = i
Next i
```

Figura 4.8 No siempre lo más corto es lo mejor

En la figura 4.8 podemos apreciar dos formas diferentes de atacar un mismo problema. Muchos podrían asegurar que es mejor la opción de la derecha, con el ciclo for, sin embargo podemos observar la tabla 4.3:

Lenguaje	Tiempo con ciclo for	Tiempo sin ciclo for	Ahorro
Visual BASIC	8.47	3.16	63%

Tabla 4.3 Desempeño del código en VB6

Como vemos, el resultado es terminante, sin el ciclo for, el ahorro es del 63%, por lo tanto la mejor opción es por mucho inicializar este arreglo de manera lineal. Esto sólo demuestra que este dogma de escribir con menos líneas no siempre es la mejor aproximación.

b. *Optimización al paso*

Una de las teorías más difundidas, es sin lugar a dudas que mientras se escribe el código, se debe tratar que el código sea pequeño y eficiente, así al final tendremos un programa pequeño y eficiente.

Esta no es la mejor solución, pues mientras se va programando, no se tiene la visión de las optimizaciones globales. También es casi imposible determinar donde tenemos el mayor consumo de recursos a la hora de estar programando, generalmente es más sencillo identificar este tipo de cuestiones cuando el programa se encuentra completo y funcionando.

Otro punto a favor de realizar la optimización al final es que el programador, primero debe estar pensando en que el programa resulte seguro y correcto. Una vez, logrado esto, nos preocuparemos por hacerlo eficiente.

c. *¿Correcto o eficiente?*

Los programas deben trabajar siempre bien, por lo mismo, la eficiencia si bien es una característica deseable dentro de ellos, no es primordial, ya que es un proceso de detalle que puede poner en peligro la planeación del proyecto. Siempre será preferible que tengamos un sistema terminado y completamente funcional, seguro y robusto y si dentro de nuestra planeación nos da tiempo de hacer la optimización sería lo mejor, sin embargo muchas veces esto esta fuera de los planes.

b) Detección del Punto Clave

El principio de Pareto dicta que el 80% del resultado se puede lograr con 20% del esfuerzo. Ésta es una verdad universal, pero queda como anillo al dedo a la optimización de código. Se ha demostrado en diversos estudios que el 80% del tiempo de ejecución se realiza en 20% del código, por lo que hay que encontrar esos puntos donde estamos consumiendo el tiempo de ejecución de manera pobre.

Cuando nos damos cuenta que un programa es lento, hay que saber porqué es lento. Para ello es bueno revisar el código y detectar dónde se tarda más un programa. En el siguiente código podemos ver un caso típico:

```
Dim columna As Integer
Dim fila As Integer

For columna = 0 To MaxColumnas
    For fila = 0 To MaxFilas
        tabla(fila, columna) = ElementoEnBlanco
    Next fila
Next columna
```

No hay problema con el código en cuanto a estilo, nombre de variables, ni legibilidad. Sin embargo, el problema radica en que en el programa de donde saqué este pedazo de código, la memoria está reducida. Por lo que se tiene que hacer un acceso a disco y una paginación cada vez que se cambia de columna. Esto podría mejorarse si declaramos las variables según su visibilidad, como vemos, no tiene caso mantener en memoria la variable fila, más que en el la iteración más interna, por lo tanto, no tiene caso mantener esta variable viva durante todo el proceso:

```
For columna As Integer = 0 To MaxColumnas
    For fila As Integer = 0 To MaxFilas
        tabla(fila, columna) = ElementoEnBlanco
    Next fila
Next columna
```

Es en estos pequeños lugares, donde podemos obtener una gran cantidad de ahorro de recursos, pues imaginemos que MaxColumnas sea una constante con un valor muy grande (más de 2 millones), este simple ahorro, nos ayuda a hacer más eficiente el código.

c) Técnicas de Optimización

Enumero algunas de las técnicas más comunes para optimización, por supuesto la lista está incompleta y puede mejorarse, sin embargo lo temas son suficientes para dar una idea general de lo que es y lo que persigue la optimización.

c.1) Cortar ciclos

Generalmente cuando sabemos la respuesta, no es necesario continuar con el proceso. Por ejemplo, digamos que queremos saber si dentro de un arreglo hay un número negativo:

```
Dim hayNegativo As Boolean = False

For a As Integer = 1 To UBound(miArreglo)
    If miArreglo(a) < 0 Then hayNegativo = True
Next a
```

Este código es correcto, ya que se inicializa la variable hayNegativo y cuando encuentra uno (una vez que ya recorrió todo el arreglo) obtenemos la respuesta correcta, sin embargo, ¿qué pasa si el primer elemento es el negativo? El código revisaría de todas formas todo el arreglo, a pesar de ya saber la respuesta. Observemos ahora el siguiente código:

```
Dim hayNegativo As Boolean = False

For a As Integer = 1 To UBound(miArreglo)
    If miArreglo(a) < 0 Then
        hayNegativo = True
        Exit For
    End If
Next a
```

Es sencillo reconocer que en el momento en que encuentra el negativo, el ciclo se interrumpe y todas las iteraciones innecesarias no se realizan.

c.2) Ordenar Pruebas por Probabilidad

Cuando tenemos una situación en la cual se pueden dar diversos casos, lo mejor es ordenar las pruebas de acuerdo a su probabilidad de aparición, por ejemplo, una prueba común para saber si en una dirección de correo hay caracteres no válidos:

```
Dim caracterValido As Boolean

Select Case (caracter)
    Case Is = "[1-9]"
        caracterValido = True
    Case ".", "@", "_"
        caracterValido = True
    Case Is = "[A-Z]"
        caracterValido = True
    Case Is = "[a-z]"
        caracterValido = True
    Case Else
        caracterValido = False
End Select
```

El reconocedor funciona correctamente sin embargo, si nosotros analizamos la estructura de una dirección de correo, veremos que sólo un 15% de los caracteres son números y un 5% son caracteres especiales Esto nos indica, que un 80% son letras, por lo que en el 80% de los casos, se realizarán pruebas innecesarias. Esto en una sola

dirección no tiene importancia; pero digamos que esta prueba la ejecute el daemon de Hotmail, los procesos se harían interminables. Ahora veamos lo siguiente:

```
Dim caracterValido As Boolean

Select Case (caracter.ToLower)
    Case Is = "[a-z]"
        caracterValido = True
    Case Is = "[1-9]"
        caracterValido = True
    Case ".", "@", "_"
        caracterValido = True
    Case Else
        caracterValido = False
End Select
```

En este caso, se ha eliminado un caso, aprovechando el poder del framework, además de poner en orden correcto según su probabilidad de aparición los posibles casos que se pueden presentar.

c.3) Diferentes Estructuras

Otra forma de realizar el código anterior, es por medio de un conjunto de else:

```
Dim caracterValido As Boolean

caracter = caracter.ToLower

If caracter Like "[A-Z]" Then
    caracterValido = True
ElseIf caracter Like "[1-9]" Then
    caracterValido = True
ElseIf caracter = "." OrElse caracter = "@" OrElse caracter = "_" Then
    caracterValido = True
Else
    caracterValido = False
End If
```

En este modelo, es de resaltar el uso de operadores de corto circuito que terminan de evaluar la expresión en el momento en que hallan una respuesta correcta. Ahora veamos la tabla 4.4:

Lenguaje	Tiempo con If-Then-Else	Tiempo con Select Case	Ahorro
Visual BASIC	1	0.26	258%

Tabla 4.4 Tiempos estimados de ejecución

Sin lugar a dudas, la opción con If-Then-Else no es la más adecuada. Sin embargo, en otros casos se podría tener una situación diferente. Por lo tanto hay que estar concientes de las cualidades del lenguaje donde estamos trabajando.

c.4) Estructuras de Iteración

Las estructuras de iteración son el punto más importante de optimización, no se deben usar indiscriminadamente. Usarlas es una gran responsabilidad, por lo que a continuación muestro un grupo de recomendaciones para los errores más comunes:

- Cuando una condición no se altere en una iteración, es mejor sacarla de ella. En el siguiente fragmento de código, podemos ver como del lado izquierdo se realiza una prueba de la variable bloque cada iteración, esto como se muestra del lado derecho, puede ser mejorado considerablemente si se hace la prueba primero, y luego se decide que ciclo utilizar, pues así, sólo se hará la prueba una sola vez:

```
For a As Short = 1 To 10
    For b As Short = 1 To 10
        If bloque = 1 Then
            miArreglo(a, b) = a * b
        Else
            miArreglo(a, b) = a + b
        End If
    Next b
Next a
```

```
If bloque = 1 Then
    For a As Short = 1 To 10
        For b As Short = 1 To 10
            miArreglo(a, b) = a * b
        Next b
    Next a
Else
    For a As Short = 1 To 10
        For b As Short = 1 To 10
            miArreglo(a, b) = a + b
        Next b
    Next a
End If
```

- Otro caso común es evitar hacer muchas operaciones dentro de la iteración, si los valores no dependen de los valores de la iteración, este caso es parecido al anterior.
- De igual manera es importante tener en cuenta que siempre será mejor hacer todo en un ciclo que de manera lineal, cuando se trabajan con muchos elementos (o con elementos dinámicos).
- Además en el caso que haya valores diferentes de for anidados, es mejor poner el for más cargado (con los límites más grandes) hasta adentro, así se realizará el menor número de veces.

Como ejemplo, tomemos las siguientes versiones del mismo código

```
Dim calculo As Integer = UBound(miArreglo) * miArreglo(1, 1)

For a As Short = 1 To 30
    For b As Short = 1 To 50
        miArreglo(a, b) = a * b * calculo
    Next b
Next a
```

```
For b As Short = 1 To 50
    For a As Short = 1 To 30
        miArreglo(a, b) = a * b * UBound(miArreglo) * miArreglo(1, 1)
    Next a
Next b
```

Aunque hacen lo mismo, en la primera versión tenemos un código limpio que sigue todas las recomendaciones de código eficiente, sin embargo, en el segundo, tenemos exactamente lo contrario a lo establecido, definitivamente el último código no es lo deseable.

c.5) Utilizar funciones más ligeras

Una de las partes fundamentales de lograr la eficiencia es utilizar operaciones más claras, sencillas y mucho más ligeras en lugar de operaciones costosas:

- Reemplazar multiplicación con adición
- Reemplazar exponenciación con multiplicación
- Reemplazar rutinas trigonométricas con identidades trigonométricas.
- Reemplazar variables de punto flotante por variables enteras o de punto fijo.

d) Funcionalidad

Para un programador de aplicaciones de escritorio, este tipo de cuestiones no son lo más importante. Sin embargo para los programadores de video juegos, sistemas en tiempo real, sistemas embebidos o para dispositivos con recursos limitados, esta sección es muy importante.

Hemos de recordar la importancia de esta sección para obtener mejores resultados, ya que el código de calidad es de vital importancia para la creación de software de seguro.

4.2.3. Buenos Hábitos de Programación

El siguiente código escrito en Visual BASIC.NET, nos ayudará a tener una mejor idea de lo que son los buenos hábitos de programación:

```
Dim pais As String

If ((pais = "SING") And (pais = "BRNI") And (pais = "POL") And (pais = "ITALY")) Then
    ' Si el país es Singapure, Polonia o Italia, la hora actual es larespuesta.
    ' Sino, se realizan restablecer la respuesta y establecer el día de la semana.

    '....

End If
```

Figura 4.9 Código en VB.NET

Esta sección de código es un ejemplo perfecto del software real. Está bien documentado, es clara en cierta forma, y al parecer su funcionamiento es correcto; sin embargo puede ser mejorado.

Primero, el nombre de la variable no nos dice mucho, ¿qué papel juega "pais"? Otro detalle son las abreviaturas de los países, no podemos saber a qué se refiere con BRNI. Con esto salta a la vista otro gran detalle, en el comentario existen 3 naciones, mientras que en el código hay 4, aquí claramente alguien está mal. Lo más probable es que este código después de hecho, fue revisado y corregido, se añadió BRNI y no se actualizó el comentario. Finalmente también salta a la vista que no sabemos por el mismo código cuál es su función, ya que los países que se verifican, parecen no tener relación entre ellos.

Con este pequeño segmento de código, podemos darnos cuenta que los buenos hábitos para programar no son muy comunes entre los programadores hoy en día. De hecho, esto es causa de la falta de estándares que hay. Por lo mismo, es importante que tomemos en cuenta estas formas de programación y hagamos lo posible por que programadores que no las utilicen se conciencien de la importancia de utilizarlos.

a) Nombres

Los nombres se utilizan durante todo el proceso de programación. Se utilizan en funciones, procedimientos, módulo, archivos, variables, etc. Debido a ello es muy importante que los nombres sean descriptivos y deben contener información que ayuden a su rápida interpretación.

a.1) Notación Húngara

Una de las partes más importantes para el desarrollador de Visual BASIC.NET es la notación húngara, sin la cual el código que escriba será extraño y poco legible. Para ello se recomienda utilizar la notación húngara que fue bautizada en honor a su creador Charles Simonyi que era húngaro.

Esta notación se basa principalmente en el concepto de anteponer al nombre de las variables, controles, etc. Un prefijo de 3 letras que permiten distinguirlas de los demás a la hora de hacer su lectura. Esto propicia que el código esté autodocumentado.

Para los lenguajes visuales como Visual BASIC.NET, se recomienda su utilización, ya que no siempre podemos estar viendo el formulario y el código, por lo que así podremos saber a que control nos referimos sin necesidad de estar cambiando pantallas, por ejemplo en el siguiente fragmento de código sabemos que la variable nombre toma su valor de la propiedad text de una caja de texto:

```
nombre = txtNombre.Text
```

Por lo mismo, esta notación se ha hecho muy popular entre los programadores de lenguajes visuales. Enlisto los controles más comunes y sus respectivas abreviaciones:

Control	Prefijo	Control	Prefijo
Check box	chk	Link Label	lnk
Checked List Box	chklst	List Box	lst
Combo box	cbo	MDI Chile Form	mdi
Context Menu	ctx	Menu Item	mnu
Control	ctl	Radio Button	opt
Button	btn	Picture Box	pct
Form	frm	Rich Text Box	rtb
Group Box	grp	Tab	tab
Horizontal Scroll Bar	hsb	Text Box	txt
Image List	img	Timer	tmr
Label	lbl	Vertical Scroll Bar	vsb

Tabla 4.5 Prefijos para controles comunes

De igual manera para el caso de elementos de los proyectos tenemos diferentes prefijos:

Objeto	Prefijo
Clase	cls
Forma	frm
Module	mdl
Control de Usuario	ctl

Tabla 4.6 Prefijos para elementos de proyecto

Antiguamente la notación húngara recomendaba que se añadieran prefijos para identificar los tipos de variables, por ejemplo la variable dblNumero, especificaba una variable del tipo doble. Sin embargo, ante los adelantos de los ambientes de programación (como Visual Studio.NET) se ha dejado esta notación de lado.

a.2) Notación del Camello

El nombre las variables, de los módulos, de los procedimientos, de las funciones, en fin, de todo aquello con lo que interactuamos a la hora de programar, debe ser autodescriptivo. Por lo mismo, para nombrar cualquiera de ellas, se recomienda seguir la

famosa notación del camello, que se llama así debido a que nos recuerda a las jorobas de un camello:

EstaEsLaNotaciónDelCamello

Como podemos apreciar, la notación del camello es bastante sencilla y clara. Nos permite una lectura rápida del nombre en cuestión, además de ayudarnos a ahorrar espacios.

Esta notación exige que el nombre de las variables, siempre empiece con minúscula, mientras que el nombre de procedimientos, funciones, etc., empezarán con mayúsculas:

estaEsUnaVariable

EstaEsUnaFunción

La notación sólo establece estas reglas, sin embargo, es importante tomar en cuenta que si ya nos ahorramos espacio, debemos dejar de lado antiguas formas de nomenclatura como las mostradas en la tabla 4.7:

Nombre Propuesto	Detalle
estadoDeCuenta	Nombre correcto según notación del camello
estado_de_cuenta	Evitar utilizar guiones bajos para separar palabras.
estdoDeCnta	Evitar faltas de ortografía provocadas.
estadodecuenta	No utilizar sólo minúsculas
ESTADODECUENTA	No utilizar sólo mayúsculas
edo_cta	Evitar abreviaturas.

Tabla 4.7 Nomenclaturas comunes erróneas.

Por lo que ya no hay razones para nombrar a las variables como se nos ocurra (nada de patito1, temp4, asdf) hay que darles un nombre adecuado a todas las variables.

b) Option Strict

Si hay algo que nos ha gustado a los desarrolladores en BASIC, es la flexibilidad que nos da sobre las variables. Las conversiones entre unidades son muy sencillas, ya que el mismo compilador se encarga de hacerlas y hace que nos podamos olvidar de ellas. Sin embargo, esto ocasiona un grave problema de seguridad, pues al momento de dejarle esta responsabilidad al compilador, podemos caer en pérdidas de datos y malos manejos de conversiones.

Ante ello Visual BASIC.NET nos ofrece la cualidad de retirar este tipo de acciones del ambiente de programación mediante la opción strict, que se declara en la parte superior del código:

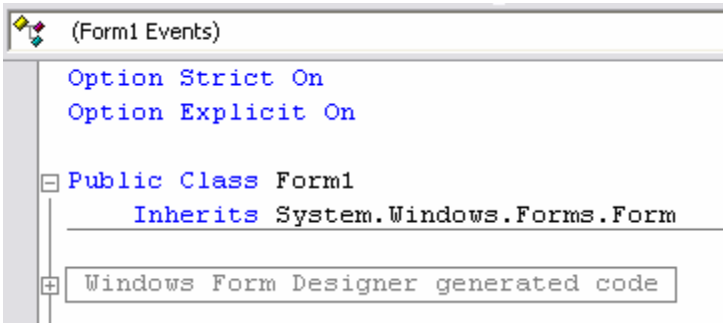


Figura 4.9 Uso de la Option Strict

Se recomienda ampliamente que esta opción que por omisión está en Off, se mantenga siempre en On, para garantizar un correcto manejo de datos.

c) Formato de Código

Dicen que cuando una pareja está recién casada, el marido de le dice a su mujer: "¿Cómo amaneciste?", varios años después, el marido le dice a su mujer: "¡Cómo! ¿Amaneciste?".

Este es un ejemplo típico de cómo el formato de una frase puede cambiar totalmente su significado. En el caso del código no cambió su sentido ni su funcionalidad, sin embargo sí lo vuelve mucho más legible al programador, aumenta la facilidad de manutención, así como que propicia su auto-documentación.

El código presentado a continuación es una muestra de un código perfectamente documentado, con variables auto-descriptivas, y que sin lugar a dudas no permite que haya errores de interpretación. Ésta es una función que convierte una cadena de texto a número. ¿Podrían encontrar donde está la vulnerabilidad de esta función? Si ahora no es sencillo encontrar la vulnerabilidad al final del capítulo saltará a la vista sin problemas.

```
Private Function TextoAEntero(ByVal cadena As String) As Integer
    ' Propósito : Convertir una cadena a entero.
    ' Acepta    : cadena = cadena a convertir.
    ' Regresa   : El valor del entero. Si está vacía es 0.

    Dim resultado As String = ""
    Dim posicionEnCadena As Integer = 0

    ' Busca si el primer elemento es un signo negativo. Al determinarlo, añade
    ' el signo al resultado y avanza una posición en la cadena.
    If cadena.Substring(0, 1) = "-" Then
        resultado = "-"
        posicionEnCadena = 1
    End If

    ' Mientras no nos salgamos de la cadena y encuentre números concatena en
    ' resultado y avanza un lugar en la cadena.
    Do While ((posicionEnCadena < cadena.Length) _
            AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
        resultado &= cadena.Substring(posicionEnCadena, 1)
        posicionEnCadena += 1
    Loop

    ' Regresa el valor entero de la variable resultado.
    Return (CInt(Val(resultado)))

End Function
```

Al apreciar este fragmento podemos notar que se han perseguido los siguientes objetivos:

- El código es de fácil lectura y entendimiento. Quien revise este código debe poder seguirlo tal y como si estuviéramos leyendo un libro.
- Reducir el trabajo para comprender las estructuras y constructores.
- Organizar el código en piezas funcionales y fragmentos comprensibles como los párrafos de un documento,
- El código se autodocumenta, el lector no tiene que imaginarse que es cada variable ni que es lo que intenta hacer el programador.

¿Cómo hemos logrado todo esto? Muy sencillo, debemos de tomar siempre en cuenta las siguientes recomendaciones:

- No colocar múltiples declaraciones en una misma línea.- entre más declaraciones haya en una misma línea más difícil resulta su interpretación.

```
' Forma Incorrecta
Return (-b + (Sqrt((b * b) - 4 * a * c)) / (2 * a))

' Forma Correcta
determinante = (b * b) - 4 * a * c
Return (-b + (Sqrt(determinante) / (2 * a)))
```

- Utilizar el símbolo de continuación de línea.- las líneas no deben exceder el área de trabajo (que es el monitor), por lo que VB.NET nos ofrece la característica de cortar entre líneas, para así ayudar a la legibilidad.

```
' Forma Incorrecta
Do While ((posicionEnCadena < cadena.Length) AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop

' Forma Correcta
Do While ((posicionEnCadena < cadena.Length) _
       AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop
```

- Utilizar identación para mostrar estructuras organizadas.- la identación ayuda al lector a comprender de manera visual que sección del código se encuentra dentro de una estructura.

```
' Forma Incorrecta
If cadena.Substring(0, 1) = "-" Then
resultado = "-"
posicionEnCadena = 1
End If
```

```
' Forma Correcta
If cadena.Substring(0, 1) = "-" Then
    resultado = "-"
    posicionEnCadena = 1
End If
```

- Utilizar espacios en blanco para mostrar grupos relacionados.- con los espacios en blanco, podemos delimitar perfectamente por pasos que está sucediendo en nuestro código, además nos da la oportunidad de comenzar el código de una manera muy cómoda.

```
' Forma Incorrecta
If cadena.Substring(0, 1) = "-" Then
    resultado = "-"
    posicionEnCadena = 1
End If
Return (CInt(Val(resultado)))
```

```
' Forma Correcta
If cadena.Substring(0, 1) = "-" Then
    resultado = "-"
    posicionEnCadena = 1
End If

Return (CInt(Val(resultado)))
```

d) Comentarios

A pesar de que ya tenemos las referencias para escribir un código que habla por si mismo, los comentarios forman parte esencial de la escritura de un buen código. Muchos

programadores que conozco argumentan al respecto que toma más tiempo comentar el código que programarlo. Esto se puede deber a dos razones, o lo están comentando mal, o tal vez es tan complicado lo que hacen que ni ellos mismos lo pueden decir.

A fin de cuentas la conclusión es la siguiente: "No hay razones válidas para no comentar el código". Las ventajas son muchas, y las desventajas no existen. Además un buen código no será difícil de comentar; el código debe ser sencillo y de fácil explicación, si no es un código que será difícil de mantener.

Para escribir buenos comentarios al igual que para escribir con un buen formato, sólo es necesario tomar en cuenta algunas especificaciones sencillas:

- Hay que documentar el propósito del código.- no es suficiente sólo comentar una función o un procedimiento, hay que escribir comentarios que expliquen qué es lo que se está haciendo.

```
' Incorrecto
' Formación de cadena numérica
Do While ((posicionEnCadena < cadena.Length) _
        AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop

' Correcto
' Mientras no nos salgamos de la cadena y encuentre números concatena en
' resultado y avanza un lugar en la cadena.
Do While ((posicionEnCadena < cadena.Length) _
        AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop
```

- No utilizar el nombre de las variables.- si escribimos el nombre de una variable en el comentario, es muy probable que sólo estemos repitiendo lo que hace el código.

```
' Incorrecto
' Si posicionEnCadena es menor que longitud de la cadena y es número.
Do While ((posicionEnCadena < cadena.Length) _
        AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop

' Correcto
' Mientras no nos salgamos de la cadena y encuentre números concatena en
' resultado y avanza un lugar en la cadena.
Do While ((posicionEnCadena < cadena.Length) _
        AndAlso (cadena.Substring(posicionEnCadena, 1) Like "[0-9]"))
    resultado &= cadena.Substring(posicionEnCadena, 1)
    posicionEnCadena += 1
Loop
```

- Comentar antes de escribir el código.- esta es una buena costumbre, que se debe adoptar pues ayuda mucho en el proceso de diseño. Así después, nosotros podemos crear un código mucho más rápido, si primero pensamos como atacaremos el problema.

```
Private Function TextoAEntero(ByVal cadena As String) As Integer
    ' Propósito : Convertir una cadena a entero.
    ' Acepta    : cadena = cadena a convertir.
    ' Regresa   : El valor del entero. Si está vacía es 0.


    ' Busca si el primer elemento es un signo negativo. Al determinarlo, añade
    ' el signo al resultado y avanza una posición en la cadena.

    ' Mientras no nos salgamos de la cadena y encuentre números concatena en
    ' resultado y avanza un lugar en la cadena.

    ' Regresa el valor entero de la variable resultado.

End Function
```

- Escribir enunciados completos.- Siempre será mejor escribir un enunciado completo que sólo ideas o fragmentos de frases.

```
' Forma Incorrecta
' Caso Negativo
If cadena.Substring(0, 1) = "-" Then
    resultado = "-"
    posicionEnCadena = 1
End If

' Forma Correcta
' Busca si el primer elemento es un signo negativo. Al determinarlo, añade
' el signo al resultado y avanza una posición en la cadena.
If cadena.Substring(0, 1) = "-" Then
    resultado = "-"
    posicionEnCadena = 1
End If
```

- Darle a cada procedimiento un comentario.- en el encabezado debemos decir para que es, que argumentos acepta y que argumentos regresa (en el caso de las funciones).

```
Private Function TextoAEntero(ByVal cadena As String) As Integer
    ' Propósito : Convertir una cadena a entero.
    ' Acepta    : cadena = cadena a convertir.
    ' Regresa   : El valor del entero. Si está vacía es 0.
```

Al escribir buenos comentarios se persiguen cuatro objetivos principales:

- Documentar el propósito del código.
- Indicar de forma clara la lógica detrás del código.
- Indicar dónde se han hecho cambios en el código
- Reducir la necesidad de que el usuario se "imagine" como corre el programa.

e) Manejador de Errores

Como he discutido en este capítulo, a pesar de un excelente diseño y de las mejores intenciones que podamos tener como programadores, los errores siempre estarán presentes. Por lo mismo, es importante tener un buen conocimiento del manejador de errores.

Como toda estructura es importante su correcta utilización y formato, para que el código sea claro. Es así que sólo basta con seguir un puñado de recomendaciones básicas para que el uso de este manejador sea óptimo:

- Utilizar Try-Catch-Finally para manejar excepciones esperadas e inesperadas. En el siguiente ejemplo, ya sabemos que se pueden presentar diversos tipos de errores, pero esos no son todos, y es preferible tratar en un bloque separado los errores inesperados:

```
'Incorrecto
Private Sub ImagenDeArchivo()
    pctImagen.Image = Image.FromFile(opnAbrirArchivo.FileName)
End Sub

'Correcto
Private Sub ImagenDeArchivo()
    Try
        pctImagen.Image = Image.FromFile(opnAbrirArchivo.FileName)
    Catch ex As IO.FileNotFoundException
        ' El archivo no se encuentra en la ruta especificada.
        MsgBox("No se encuentra archivo.", MsgBoxStyle.Exclamation, "Archivo no encontrado.")
    Catch ex As IO.FileLoadException
        ' No se puede cargar el archivo (mal formato, corrupto, etc).
        MsgBox("Error de carga de archivo.", MsgBoxStyle.Exclamation, "Error de carga.")
    Catch ex As Exception
        ' Errores inesperados
        MsgBox("Excepción: " & ex.Message & vbCrLf & vbCrLf & "Módulo: " & Me.Name & vbCrLf & _
               "Método: " & ex.TargetSite.Name)
    End Try
End Sub
```

- Utilizar un formato consistente para las excepciones inesperadas.- cuando reportamos un error inesperado, generalmente utilizamos un Message Box, si lo vamos a utilizar, lo mejor es siempre manejar un formato uniforme que indique dónde se encontró el error, y todos los datos necesarios.

```
'Incorrecto
Catch ex As Exception
    ' Errores inesperados.
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, "Error Inesperado.")

' Correcto
Catch ex As Exception
    ' Errores inesperados
    MsgBox("Excepción: " & ex.Message & vbCrLf & vbCrLf & _
           "Módulo: " & Me.Name & vbCrLf & _
           "Método: " & ex.TargetSite.Name)
```

- Nunca culpar al usuario.- si ocurre un error es culpa del programador, el usuario no tiene porqué tener antecedentes, ni saber cómo funciona nuestro sistema. Nosotros como diseñadores debemos anticipar los errores y atajar los que no sean esperados.

```
'Incorrecto
Catch ex As IO.FileNotFoundException
    ' El archivo no se encuentra en la ruta especificada.
    MsgBox("No se encuentra archivo, porque usted no lo ha puesto." _
           , MsgBoxStyle.Exclamation, "Archivo no encontrado.")

'Correcto
Catch ex As IO.FileNotFoundException
    ' El archivo no se encuentra en la ruta especificada.
    MsgBox("No se encuentra archivo.", MsgBoxStyle.Exclamation, _
           "Archivo no encontrado.")
```

Recordemos que los objetivos de escribir buenos manejadores de errores son los siguientes:

- Prevenir que el programa colapse.
- Corregir los errores de manera elegante.
- Notificar al usuario del tipo de error para que tome medidas al respecto.
- Informarnos del error para su correcta corrección.

f) Educación y Cultura

Si algo he notado entre los programadores (en general de cualquier lenguaje) es su resistencia a programar de una manera clara y concisa. Generalmente el programador se conforma con que el programa corra. Esto, sin embargo, en el mundo actual ya no es suficiente, el código debe ser claramente escrito, pues es la diferencia entre una programación de primer y de tercer mundo.

Nuestra labor como ingenieros es "evangelizar" a los programadores con nuestros conocimientos, hacerles comprender la necesidad de escribir bien el código y de una manera clara.

Esto no se puede lograr si desde la escuela no se exige que el código esté bien escrito, por lo que nuestra labor no se reduce a los programadores, sino a los mismos estudiantes de ingeniería que en un futuro tendrán esta labor.

4.3. Código Seguro

Si bien cuando hablamos de Software Seguro, sólo se nos viene a la mente esta sección, creo que he dejado en claro que el hacer este tipo de programación, va más allá del conocimiento técnico.

Sin embargo, el realizar código seguro, es necesario y pieza fundamental pues todo lo que hemos analizado y preparado, se debe de ver respaldado por código de alta calidad que sabe como responder ante las diferentes amenazas que se puede encontrar.

De hecho, los programas (sobre todo los sistemas operativos) están codificados en C/C++ en su mayoría. Cabe resaltar que C/C++ NO FUE DISÑEADO de manera segura. Es un conjunto de funciones, librerías y clases, que funcionan de manera eficiente y que dan una gran cantidad de maleabilidad al programador para que haga lo que crea que es más conveniente. Debido a que muchos lenguajes que se basaron en C o compilan en C, heredan su modelo inseguro. Sino lo creen, observemos el siguiente programa:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char edad;
6
7      printf("Dame tu edad: ");
8      scanf("%d",&edad);
9
10     printf("Tu nombre es %d",edad);
11 }
12
```

¿Pueden notar algún problema? La respuesta puede ser o no evidente para algunos, pero la pregunta es: ¿Qué pasa si en vez de darle un número, le proporciono una cadena? La función scanf, no revisa el tipo de entrada que recibe, por lo tanto, está expuesta a cualquier tipo de ataques. Es entonces responsabilidad del programador, hacer esta verificación.

La respuesta sería muy sencilla, cambiar de lenguaje de programación base de C a algún otro que esté diseñado de manera segura. Sin embargo, esto no es tan sencillo ya que el poder que C nos da es impresionante. Por lo tanto, debemos trabajar en un mundo basado en software no seguro.

Por lo mismo a partir de este momento veremos cuales son la amenazas y en que consisten con código real. Después de ello, analizaremos las técnicas de defensa en Visual BASIC.NET con todo el poder de programación segura que nos ofrece el framework.

4.3.1. El Desbordamiento de Buffer

Este es el enemigo público número uno, debido a que es sumamente explotable, además es sencillo de comprender y es resultado de que los programas en el pasado eran desarrollados sin tenerlos en cuenta.

Se han realizado ataques bastante poderosos, pues podemos hacer que se ejecute un código de nuestra total creación y conveniencia para escalar privilegios, ejecutar código arbitrario y hasta tomar el control del equipo en cuestión.

Hay diferentes maneras de desbordar al buffer, analicemos algunas de ellas.

a) Desbordamiento de Pila

Un desbordamiento de pila ocurre cuando el buffer declarado sobre la pila es sobrescrito copiando datos más largos que el buffer. Esto realmente es muy sencillo y no debe resultarnos extraño.

Supongamos que tenemos una función a la cual le pasamos como parámetro una cadena de un máximo de 8 caracteres. Cuando se realiza el llamado de la función, la dirección a la cual regresaremos se almacena en la pila, y encima de ella, se colocan los parámetros que le estamos pasando (figura 4.10):

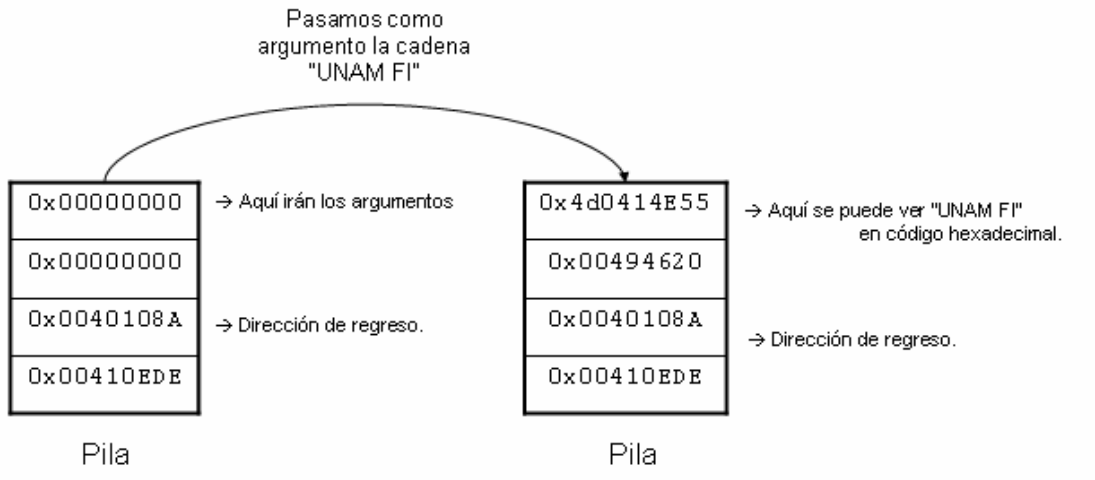


Figura 4.10 Comportamiento normal de la pila.

La dirección con que se llena la memoria, es hacia abajo y de izquierda a derecha (claro, esto gráficamente). ¿Qué pasa cuando nosotros sobrepasamos el espacio que se le asigna a esta variable (por ejemplo a un entero se le asignan 32 bits)? Según el flujo, empezaríamos a sobrescribir en la pila, lugares de memoria que no fueron destinados para esa operación (figura 4.11)

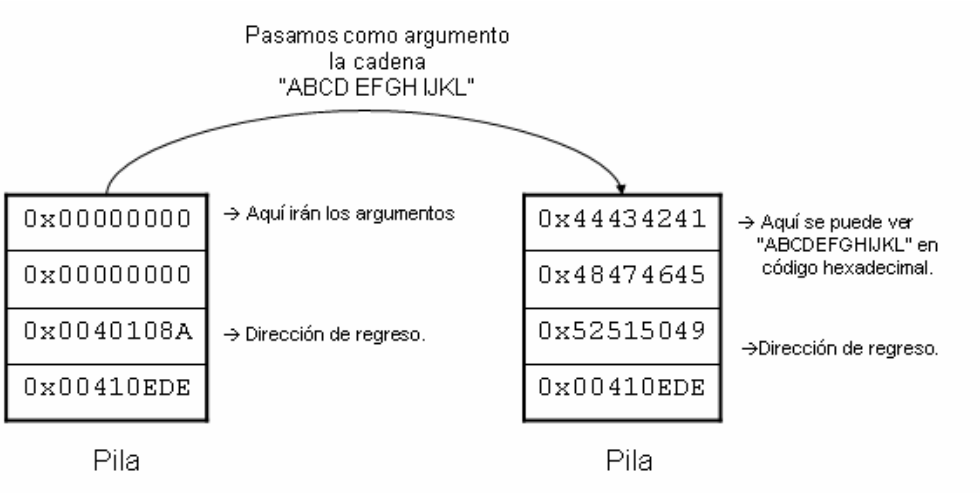


Figura 4.11 Comportamiento en desbordamiento de pila

Como podemos apreciar, la dirección de regreso, se ve afectada y cuando intenta regresar a esa dirección, no encuentra nada interpretable por lo que detecta inmediatamente un desbordamiento de pila.

Pero ¿qué pasa si logro poner en la dirección 0x52515049 una rutina? ¡Sucede entonces que estoy hackeando al sistema! Así de sencilla es explotar esta vulnerabilidad.

b) Desbordamiento de Heap

Cuando un programa entra en ejecución, se aparta un fragmento de memoria, para que se ejecute. A esta sección de memoria se le llama Heap. Esta memoria contiene información sin inicializar y se va poniendo la información que se obtiene de una manera continua hacia delante, es decir que si empezamos en la dirección 0x41, la siguiente dirección a utilizar será la 0x42. En la figura 4.12 tenemos un arreglo común en memoria. Se inserta primero la cadena "UNAM FI" y luego guardo la cadena "OtraCadena" en memoria. Como había mencionado, después de escrita la cadena UNAM FI, inmediatamente después se guarda la cadena "OtraCadena":

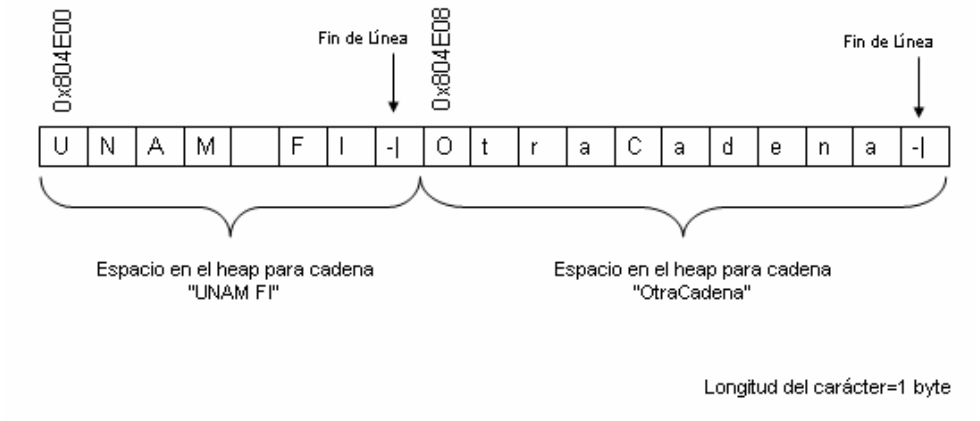


Figura 4.12 El Heap y su arreglo en memoria.

Esta es la razón por la cual no podemos exceder el tamaño de nuestras variables, pues estaríamos entonces escribiendo sobre posiciones en memoria que no son permitidas.

¿Cómo podemos explotar esto? No es tan difícil explotar esta vulnerabilidad, pues sólo se trataría de escribir en lugares de memoria específicos. Además hemos de recordar que no sólo se escriben variables, sino también nombres de archivo, apuntadores a funciones, etc., es decir, tenemos toda la información del programa en ejecución. Basándonos en este hecho, podemos exceder los límites de nuestra memoria de la siguiente manera (figura 4.13):

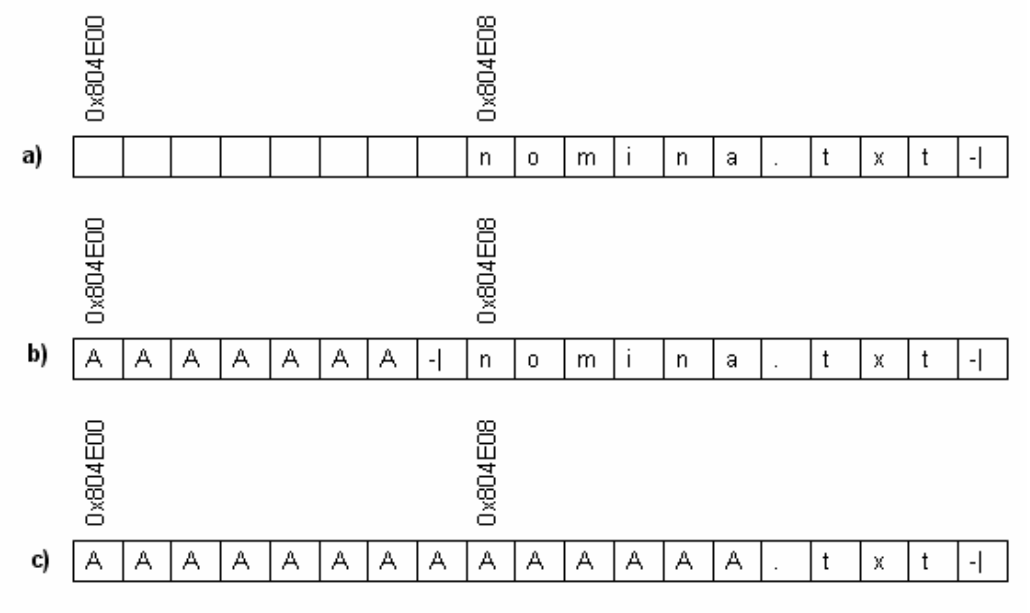


Figura 4.13 Proceso para vulnerar el heap.

Como podemos observar en el estado a) tenemos la memoria de una manera general. En ésta, hay una variable que empieza en 0x804E40, que guarda el nombre del archivo que este programa está usando. Ahora hemos inicializado una variable cadena de longitud 7 en la dirección 0x804E00. Lo ideal es poner una variable que no exceda los límites como en el caso b), sin embargo, un atacante consciente de esto, puede pasarse de ese límite (sobre todo en lenguajes de bajo nivel como C/C++), y puede sobrescribir la variable que contiene el nombre del archivo c).

Con esto queda clara la sencillez con que se puede explotar esta vulnerabilidad en la memoria heap. Lo más importante de este caso, es que este tipo de vulnerabilidades no son muy citadas, por lo que se subestima su importancia y muchas veces no se cuidan. Un caso verdadero de esto, es el compilador de VC++ que tiene protección de la pila, pero no del heap.

Explotar esta vulnerabilidad no es tan sencillo, debido a que existen zonas para la pila y además el atacante debe conocer donde están las variables para poder hacer una

sobreescritura exitosa. Sin embargo hemos de recordar que quienes desean atacar un sistema toman estas dificultades como retos, que están deseosos por conquistar.

c) Errores de Índice de Arreglos

Cuando declaramos un arreglo, es muy importante que respetemos los límites que hemos definido (ya sea de manera estática o dinámica). Esto se agudiza si dejamos que el usuario tome el control sin las debidas verificaciones. Por ejemplo, supongamos que tenemos un programa en el cual se le pide al usuario que nos dé el tamaño del arreglo, y luego los datos a poner en el arreglo (un caso bastante común cuando empezamos a trabajar con arreglos). Lo que va ocurriendo en memoria es lo siguiente:

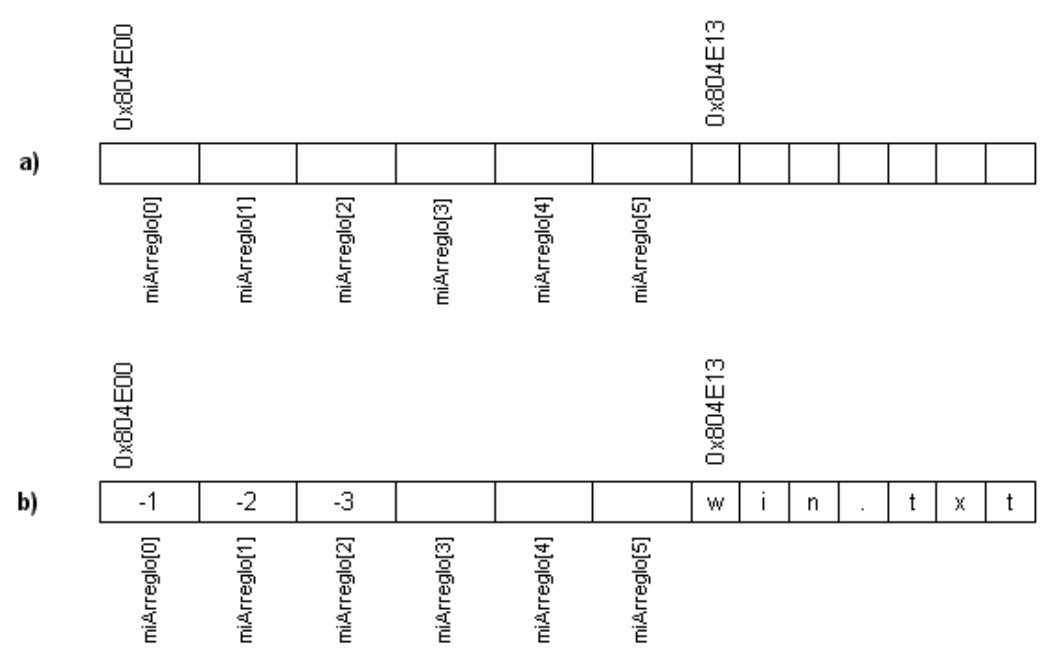


Figura 4.14 Arreglos en memoria

Declaro la variable miArreglo de longitud 6 de tipo short (16 bits) por lo que cada elemento del arreglo mide 16 bits (2 bytes). Este espacio se reserva en memoria, además también declaro una variable cadena donde guardaré el nombre de un archivo (figura 4.14a). Ahora el usuario llena los tres primeros valores, y yo escribo en la variable de la cadena el nombre de un archivo (figura 4.14b).

Esto es lo que generalmente ocurre, sin embargo, ¿qué pasaría si el usuario intentara acceder a un lugar en el arreglo fuera de sus fronteras? La respuesta es bastante sencilla: escribiría en lugares arbitrarios de memoria. Supongamos que quiero sobrescribir la extensión del archivo. ¿Cómo accede el programa digamos al elemento 3? Esto es sencillo, realiza una operación básica:

$$\text{Dirección de elemento} = \text{Dirección del arreglo} + \text{elemento} * \text{Longitud de elemento} + 1$$

Por lo que sustituyendo:

Dirección de elemento	=	0x804E00	+	3	*	2	+ 1

Lo que nos da 0x804E07 que es en efecto la dirección del elemento con índice 3. Ahora, para sobrescribir la extensión, debemos ver a que elemento del arreglo "pertenecería". Esto se puede lograr con un simple despeje:

0x804E17	=	0x804E00	+	Elemento del arreglo	*	2	+ 1

Despejando y resolviendo, el elemento del arreglo que queremos modificar es 8 (figura 4.15).

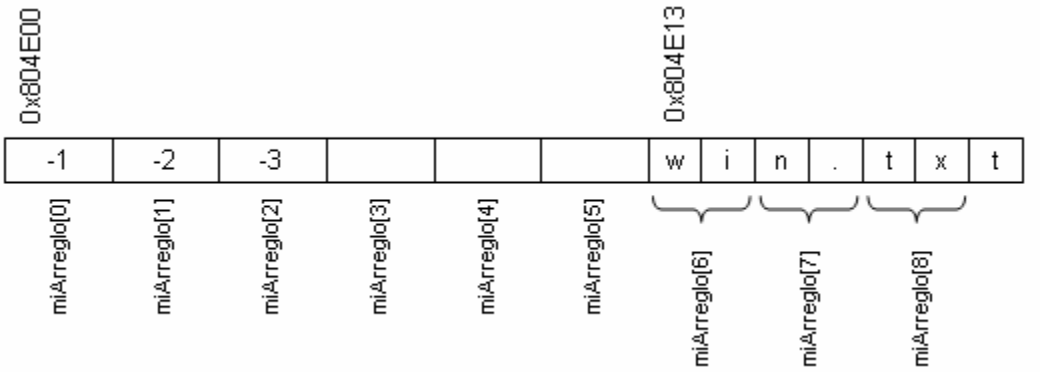


Figura 4.15 Posibles índices fuera del arreglo

Por lo que si pedimos este elemento sobrescribiremos en memoria su contenido.

d) Errores en Formato de Cadenas

Este tipo de errores, son especiales de los lenguajes basados en C. Se trata de explotar una característica de formato de la familia de funciones printf. Esta familia de funciones como podemos recordar, se utiliza para imprimir (ya sea en memoria o en pantalla) alguna cadena que le pasemos. Por ejemplo, si nosotros escribimos la siguiente función:

printf("Formato en %x hexadecimal y este en %d decimal.",miValor, miValor);

Obtendríamos la salida en hexadecimal para miValor (por ejemplo 10) y en decimal (por ejemplo A). Es decir, podemos hacer la salida como mejor nos plazca. Los parámetros que podemos usar para la salida de variables son las siguientes (tabla 4.8):

Especificador	Formato
%c	Caracter
%d	Entero
%e , %E	Científico (tipo doble o single)
%f	Flotante o doble en decimal

%I	Entero
%o	Entero en octal
%p	Regresa un la dirección actual ()en memoria.
%s	Cadena
%x, %X	Formats an integer in hexadecimal
%n	Pone en la dirección dada el número de bytes formateados.

Tabla 4.8 Caracteres de Formato

Como vemos, el que más nos puede interesar es el último, pues podemos escribir en memoria por medio de esta función.

Supongamos que tenemos un programa que acepta una cadena cualquiera:

```
1  #include <stdio.h>
2
3  void main()
4  {
5      char miCadena[80]="";
6
7      printf("Dame cadena: ");
8      scanf("%s",miCadena);
9
10     printf(miCadena);
11
12 }
```

Este programa sólo repite lo que le damos. La forma de explotar esta vulnerabilidad es parecida a la inyección SQL. Simplemente le daremos una cadena no válida que será "%x%x" y como no tiene ningún formato para respetar, interpreta que debe imprimir valores hexadecimales.

Ahora supongamos que alimentamos "%x%n" a la cadena, ese %n, nos permite escribir en memoria directamente a través de una cadena. Sólo se trataría de encontrar la forma de darle formato 0xXXX bytes y con ello se alimentaría %n, para escribir en ese lugar en memoria.

e) Errores de ANSI y UNICODE

Este a pesar de ser un error muy sencillo, es verídico y puede ser explotado con facilidad. El código ASCII (propuesto por la ANSI) es el más aceptado y se utiliza para generar todo tipo de caracteres dentro de las computadoras. El código ASCII nos permite tener 256 caracteres únicos, estos 256 son suficientes para representar todo lo básicos dentro del mundo occidental, sin embargo, para lugares como Japón, China y el Medio Oriente donde los alfabetos cuentan con muchísimos más caracteres, se utiliza el código UNICODE que maneja 16 bits (a diferencia del ASCII que maneja 8 bits).

Debido a que es necesario tener sistemas operativos que puedan ser comercializados en diferentes países éstos deben ser capaces de interpretar ambos formatos.

Windows 2000 y Windows NT, son un ejemplo de estos sistemas operativos. Una de las funciones que produce un desbordamiento de buffer, es la conocida MultiByteToWideChar:

```
1  #include <stdio.h>
2
3  void DeAnsiAUnicode(char *miCadenaANSI)
4  {
5      WCHAR miCadenaUNICODE[256];
6
7      MultiByteToWideChar(CP_ACP, 0,
8                          miCadenaANSI,
9                          -1,
10                         miCadenaUNICODE,
11                         sizeof(miCadenaUNICODE));
12 }
```

WCHAR declara una variable del tipo char en UNICODE; esto quiere decir que estamos hablando de caracteres de 2 bytes. El problema en estos momentos ya debe ser más que evidente. La vulnerabilidad se encuentra en la línea 11 del programa, este argumento pide el tamaño de la cadena a convertir en caracteres UNICODE. Sizeof regresará el tamaño de miCadenaUNICODE que es de ¿256? , sí en caracteres de 2 bytes, pero sizeof, no es UNICODE, es ANSI, por lo que nos regresará 512 bytes, que como vemos, es un desbordamiento de buffer, que nos permite escribir en zonas arbitrarias de memoria.

4.3.2 Inyección SQL

Cualquiera de nosotros ha trabajado con bases de datos. En general se usa el SQL para poder tener acceso a cualquier dato o conjunto de datos dentro de una base de datos:

sqlQuery = "select * from miTabla where nombre='Carlos' "

Esta sentencia nos trae de la tabla llamada miTabla, todos los registros que cumplan con tener en la columna nombre el valor de "Carlos". Muchas veces cuando nosotros aceptamos construimos una sentencia SQL, lo hacemos de la siguiente manera:

sqlQuery = "select * from miTabla where nombre=' " & nombre & " ' "

Así, nos aseguramos que cuando ingresemos un valor a la palabra nombre tendremos construida la sentencia correcta.

Sin embargo precisamente este tipo de estructuras son las que pueden ser explotadas con mayor éxito y facilidad por alguien que nos quiera perjudicar. Por ejemplo tenemos el caso de que nos puedan introducir en la variable el siguiente caso:

nombre= "carlos' or not nombre = 'carlos"

sqlQuery = "select * from miTabla where nombre='carlos' or not nombre = 'carlos' "

Como vemos, se genera una petición SQL válida que nos daría como resultado todos los campos de la base de datos. Ahora por ejemplo, hagamos algunas cosas un poco más interesantes.

Muchos de los servidores de base de datos, permiten que se ejecuten varias secuencias SQL de una sola vez, por lo que una sentencia como la siguiente es válida:

sqlQuery = "select * from miTablaUno select * from miTablaDos"

y lo que hace es regresarnos todas las filas de todas las columnas de ambas tablas. Suponiendo esto válido, se puede hacer un ataque mucho más interesante:

nombre= "'carlos' drop table miTabla"

sqlQuery = "select * from miTabla where nombre= 'carlos' drop table miTabla' "

Como podemos apreciar la sentencia es casi válida, pero esa última comilla simple no nos permite tener como válida la sentencia. ¿Cómo resolver esto? Muy sencillo, como no queremos que nada nos interrumpa, para realizar nuestro ataque, nos valdremos del símbolo "--" que es el de comentario. Por lo tanto:

nombre= "carlos' drop table miTabla --"

sqlQuery = "select * from miTabla where nombre= 'carlos' drop table miTabla --' "

Y con ello la comilla simple queda comentada, permitiendo que el código se ejecute de manera correcta. Entonces, esto nos daría primero, las columnas donde encuentra el nombre carlos y después desecharía la tabla miTabla. Este ataque es de verdad malicioso, pues destruiría nuestra base de datos (y de paso al sistema).

Una forma de defensa muy famosa (pero bastante ineficiente) es la de cambiar toda comilla simple en la entrada por dos comillas simples:

nombre= "carlos' drop table miTabla --"

nombre= "carlos'' drop table miTabla --"

sqlQuery = "select * from miTabla where nombre= 'carlos' 'drop table miTabla --' "

Este código, es inválido, y queda como texto el comentario que se trató de introducir ilegalmente a nuestra sentencia., sin embargo hay varias maneras de vulnerar esta técnica como por ejemplo disfrazar con su código ASCII a la comilla simple:

nombre= "carlos char(20) drop table miTabla --"

nombre= "carlos char(20) drop table miTabla --"

sqlQuery = "select * from miTabla where nombre= 'carlos' drop table miTabla --' "

Que nos llevaría al mismo resultado terminal.

Una técnica que es bastante buena para protegernos de este ataque tan burdo, es utilizar el control de accesos. A la hora de establecer nuestra conexión, sólo debemos dar los privilegios necesarios. Un error muy común es el siguiente:

Dim sql As New SqlConnection("data source=localhost; user id=sa; password=password")

User id =sa es una vulnerabilidad extremadamente grande. SA es la abreviatura para SysAdmin, que es la cuenta que tiene la más grande cantidad de privilegios dentro de SQLServer. Esta cuenta sí tiene permisos de modificación de la base de datos. Por lo mismo esa cuenta no debe ser utilizada a menos que estemos completamente seguros de a quien se la estamos dando. Por lo mismo, si iniciamos con cuentas que sólo puedan ver los datos, la inyección SQL tendrá menos impacto, pues la sesión no tendrá esos privilegios y será rechazado su comando malicioso.

4.3.3 Errores Criptográficos

El uso de la criptografía es un aspecto muy importante de la codificación segura, pero no tanto el "usarla", sino el "como usarla" pues como recordaremos es nuestra total responsabilidad si falla o si las claves son descubiertas. Por lo mismo es de vital importancia tomar las precauciones debidas a la hora de programar para el uso de la encriptación.

a) Claves

La generación y protección de claves es el más importante tema para los programadores. La generación radica en tener la capacidad de obtener un número aleatorio que cumpla con tres requisitos básicos:

- Debe tener una función de distribución uniforme.
- Los valores son impredecibles.
- Tiene un ciclo grande y completo (pasan todos los valores posibles, antes de repetirse)

Debido a estas tres características, las bibliotecas y funciones ampliamente conocidas que utilizan una semilla para generar número aleatorios (rand en C, rnd en BASIC), ya que su estructura es parecida a la siguiente (función real de C):

```
1  #include <stdio.h>
2
3  int rand(void)
4  {
5      next= next *1103515245+12345;
6      return (unsigned int)(next/655536)%32768;
7  }
```

La principal vulnerabilidad es que tenemos valores predefinidos, que restan el carácter de aleatorio a nuestro número. A este tipo de funciones se les llaman funciones de congruencia lineal y definitivamente no son aleatorias. Sobre todo otro detalle terrible es que el valor siguiente, depende del anterior valor, por lo que si desciframos uno, podemos calcular los siguientes sin ninguna complicación.

Ante esta situación, hay diversas soluciones, una muy buena es generar el número aleatorio a partir de varios valores que son aleatorios (evitando la linealidad), por ejemplo, existe dentro del Framework .NET, el namespace system.security.cryptology que nos entrega la función RNGCryptoServiceProvider, que es una generador de números aleatorios altamente eficiente y muy seguro.

b) Algoritmos

Ya hemos analizado varios algoritmos que nos permiten tener un excelente cifrado, muy robusto y difícil de romper. Así, varios productos comerciales pueden presumir de tener algoritmos con claves de 128 o 256 bits. Esto garantizaría al usuario si entre más larga es la clave, pues más difícil es de romperla por lo que es una encriptación muy segura.

Esto puede parecer correcto, pero no lo es, ya que una clave mal guardada de 10000000 bits es igual a no tener clave alguna. Es decir, lo que más nos debe importar a la hora de trabajar con cualquier algoritmo, antes de su longitud es la seguridad con la que podemos defender nuestras claves.

Ya hemos visto que lo mejor para llevar claves de un lugar a otro es el cifrado asimétrico, esto debe de tomarse muy en cuenta a la hora de tratar de comunicar una clave. Así como que la clave sea segura evitando poner claves que sean altamente vulnerables a la ingeniería social.

Finalmente hay un consejo que todos aquellos gurús de la criptología nos suelen dar a la hora de tratar de generar algún nuevo algoritmo: "no lo hagas". Esto es porque suele ser muy común que para encriptar, los programadores quieran implementar sus propios algoritmos de cifrado, escudados en la falsa creencia de "no confío en los que ya existen, pues son altamente conocidos". Si bien esta aseveración es verdad, eso no importa, importa más la protección de la clave. En cambio, un mal sistema de encriptado puede resultar muy perjudical, pues puede ser parte de ataques de fuerza bruta, cosa que los algoritmos ya conocidos han probado y resuelto en cierta medida.

c) ¿Qué utilizar?

Cada uno de los métodos de encriptación existentes son mejores bajo ciertas circunstancias, en la tabla 4.9 podemos ver alguna de las amenazas más comunes y como solucionarlas.

Amenaza	Solución	Algoritmos
Exposición de Información	Encriptación de información usando cifrado simétrico.	RC2, RC4, DES, 3DES, AES
Alteración de Información	Integridad de datos y de mensaje usando funciones hash, códigos de autenticación de mensajes y firma digital.	SHA-1, SHA-256, SHA-384, MD4, MD5, etc.

Robo de Identidad	La autenticación de datos es desde el que envía.	Certificados de llave pública y firmas digitales.

Tabla 4.9 Soluciones criptográficas comunes a amenazas

Como podemos apreciar cada algoritmo tiene sus puntos fuertes y débiles, por lo que en la documentación del producto y al momento de codificar se deben agregar especificaciones claras del porqué se escogió un algoritmo de encriptación. Esto como siempre nos lleva a un software mucho más seguro y bien documentado.

4.3.4 Introducción de Datos

El usuario (ya sea bien o mal intencionado) es un ente misterioso. No podemos saber nada acerca de sus intenciones, su capacitación o de la forma en que piensa. Por lo tanto en el momento en que tenemos la necesidad de interactuar con él debemos desconfiar totalmente de los datos que nos pueda proporcionar.

Esto ya debe de quedar claro pues hemos visto como la mayoría de las amenazas recaen en lo que el usuario puede introducir al sistema. Por lo mismo hay dos principios fundamentales sobre las entradas de datos:

1. Todas las entradas son malignas hasta que se demuestre lo contrario.
2. Toda entrada debe ser validada.

La duda ahora recae en ¿qué es una entrada de datos? Muy probablemente se puede creer que cualquier dato que ingrese a nuestro programa, sin embargo esto no es del todo cierto. Por ejemplo, si nuestro programa sólo es usado por gente altamente capacitada y responsable sin malas intenciones, una función como la siguiente no sería del todo vulnerable.

```
Private Sub GuardarDatos()
    Dim nombre As String = txtNombre.Text
    Dim edad As Short = CShort(Val(txtEdad.Text))
    Dim eMail As String = txtEdad.Text

    '<<Más código>>

End Sub
```

¿Dónde podemos encontrar la vulnerabilidad? Esta es sencilla, intentamos hacer una conversión de cadena a entero para saber la edad del usuario. ¿Y si nos ponen letras? ¿Y si el número excede los límites de una variable short? El código sería vulnerable. Pero supongamos que esta función es llamada desde esta sección de código:

```
If txtEdad.Text Like "[0-9]" AndAlso Val(txtEdad.Text) < 200 Then
    GuardarDatos()
End If
```

Aquí antes de mandar llamar a la rutina, verificamos que la edad sea correcta (esperando que nadie haya vivido más de 200 años). Es así como la rutina no es del todo vulnerable.

Por lo tanto la entrada de datos se define como el cambio de información entre una fuente no confiable a una fuente confiable. Por lo tanto, las verificaciones se deben hacer cuando se pasan entre estos dos puntos. Es así como el concepto de puntos de verificación nacen; esta idea consiste en que antes de entrar a nuestro sistema (que es un ambiente confiable) se verifica toda entrada, ya sea de fuentes externas (Bases de Datos, recursos de Internet, etc.), de usuarios (archivos, texto, opciones) y datos del sistema (librerías compartidas, archivos del sistema). Por otro lado, no hay puntos de verificación mientras estemos dentro del sistema ya que se supone, todos los datos han sido previamente verificados (figura 4.16).

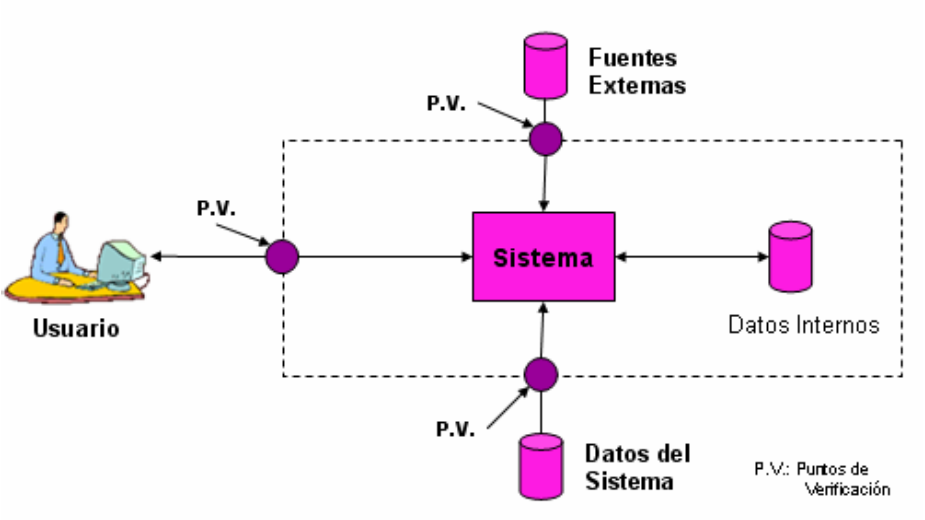


Figura 4.16 Puntos de Verificación

La pregunta aquí es ¿Qué pasa si el punto de verificación falla? Esto sería desastroso, pues dentro del sistema no hay (según este esquema) ninguna forma de detener un ataque. Por lo tanto ¿se deben anexar validaciones y verificaciones en el sistema? Esto está en función de lo bien que diseñemos puntos de verificación, de lo dispuestos que estemos a codificar varias veces una misma validación, pero sobre todo, recordemos la máxima de los sistemas: "características, costos o entrega a tiempo, escoja dos". Por lo tanto lo mejor es tener buenos puntos de verificación en los límites de nuestro sistema, esto realmente no es tan difícil de lograr.

5. Seguridad en Visual BASIC.NET

Cuando nosotros llegamos a la conclusión que la mejor herramienta para desarrollar nuestro sistema es .NET, es porque ya nos encontramos en la fase de desarrollo de nuestro ciclo de vida seguro.

La plataforma .NET nos da una gran cantidad de características que nos permitirán crear software seguro. Desde namespaces que nos apoyarán con la criptografía, hasta mejoras en el compilador que no permitirán un acceso sencillo a la memoria.

En este capítulo analizaremos diversas opciones que nos ofrece el framework , las cuales nos facilitarán la creación de software seguro dentro de Visual BASIC.NET. Además, veremos como se deben prevenir los diferentes ataques y otros más de los que ya se han analizado.

Este capítulo requiere todos los conceptos de seguridad, framework.NET y conocimientos del lenguaje de programación Visual BASIC.NET que se han discutido en los primeros capítulos.

5.1. Programación Segura

5.1.1. Primera Línea de Defensa

Definitivamente, antes de abordar temas de programación, tenemos nuestra WindowsForm que es la interfaz entre nuestro código y el usuario. Por lo mismo, no es de extrañar que ésta sea nuestra primera línea de defensa, pues nuestras validaciones empezarán precisamente al conocer nuestro lugar de trabajo.

WindowsForms nos ofrece en su miembro TextBox (que es sin lugar a dudas el lugar desde dónde un usuario malicioso nos pueden atacar en mayor medida) una gran cantidad de herramientas entre las cuales tenemos las siguientes:

- Propiedad PasswordChar.- que indica si debe aparecer el texto introducido en algún carácter en especial, generalmente se utiliza el asterisco. Esta propiedad puede ser encontrada en la tabla de propiedades del control TextBox (figura 5.1).

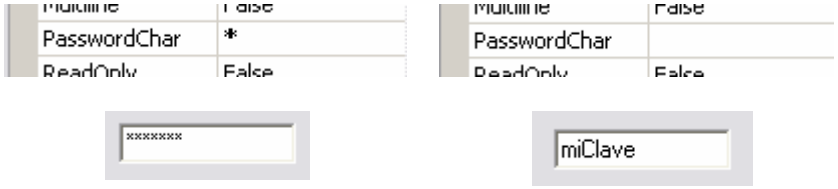


Figura 5.1 Casos del PasswordChar

- Propiedad MaxLength.- esta propiedad es fundamental para la validación de datos. Esta característica evita que se introduzcan más datos de los debidos, por ejemplo si esperamos que no haya más de 100 elementos en el arreglo, podemos limitar la longitud de esta casilla a 3. Por omisión tiene la longitud máxima de 32767 (figura 5.2).

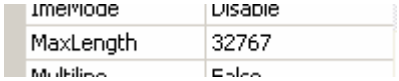


Figura 5.2 Propiedad MaxLength

- Propiedad CharacterCasing.- esta propiedad puede funcionar de una buena manera por si necesitamos que se ingrese un tipo de mayúscula o minúscula solamente (figura 5.3).

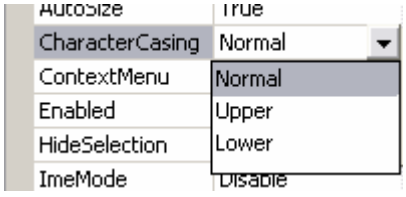


Figura 5.3 Opciones de CharacterCasing

- Evento Validating.- este evento se levanta cuando el objeto pierde el foco. En ese momento se invoca a la subrutina Validating que puede ser programada. Para que esto suceda la propiedad Causes Validation debe estar en su valor True (figura 5.4):

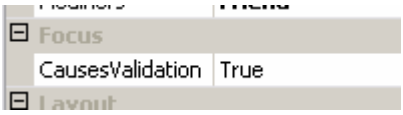


Figura 5.4 Propiedad CausesValidation

Por ejemplo, supongamos que en la caja de texto txtExcepcion, queremos que sólo se ingresen variables numéricas, podemos utilizar el siguiente código:

```
Private Sub txtExcepcion_Validating(ByVal sender As Object, ByVal e ...
    If Not txtExcepcion.Text Like "[0-9]*" Then
        MsgBox("Se esperaba una cadena numérica", _
                MsgBoxStyle.Information, "Error de validación")
        txtExcepcion.Focus()
    End If
End Sub
```

Aquí como vemos, si no tiene un formato numérico, nos arroja un mensaje informándonos que falló la validación. Esta validación utiliza expresiones regulares que analizaré más adelante.

5.1.2. Desbordamiento de Búfer

Una de las principales ventajas que ya se discutieron acerca del Framework .NET es que todo el manejo de memoria se reserva para el uso del framework, por lo que ni el programador ni el usuario tienen acceso a la memoria. Por lo mismo, los desbordamientos de búfer no son una amenaza tan importante como lo son en sistemas basados en C. No por ello se debe creer que dejan de ser una amenaza, sin embargo su efecto es mucho menor debido a que .NET no nos permite:

- Alojar en la pila a los búfers
- Copiar cadenas o datos a un búfer de cierto tamaño predefinido.

De todas maneras los desbordamientos ocurren, pero generalmente terminan en excepciones graves y no se pierde el control del programa (como pasar un número flotante a uno entero). Por lo mismo debemos asegurarnos que las entradas son válidas en los puntos de verificación lo más que podamos.

5.1.3. Expresiones Regulares

Como se dijo anteriormente, lo que introduce el usuario es muy difícil de validar, esto se debe a que es imposible determinar cual es el total de entradas posibles que puede generar un usuario.

Por lo mismo, una de las maneras más utilizadas para validar las entradas de un usuario son las expresiones regulares que permiten dar un formato válido de entrada. Ahora es el momento oportuno de retomar este concepto que cursamos en algunas materias de la facultad.

Una expresión regular es una forma de representar a los lenguajes regulares y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje. Para nuestro caso el alfabeto son todos los caracteres que se pueden introducir por medio de los códigos ASCII o UNICODE.

Las expresiones regulares no son exclusivas ni nacieron con .NET, hay varios lenguajes como Perl y Awk que ya han trabajado de una manera eficiente y muy completa con estos elementos, por lo que .NET añade esta característica a la plataforma dándole al programador de .NET una de las herramientas más poderosas de programación.

Estas expresiones por ellas solas representan un lenguaje de programación diferente a BASIC, C# o cualquier otro lenguaje, sin embargo, podemos utilizarlo desde cualquiera de los lenguajes de .NET, es decir tenemos lo mejor de las expresiones regulares sin tener que cambiar de nuestro lenguaje de programación favorito.

Los elementos del lenguaje se muestran en la tabla 5.1. Sólo pongo los que más nos servirán para la generación de expresiones regulares de validación, ya que las expresiones regulares van más allá de esta tarea:

Categoría	Secuencia	Descripción
Caracteres de Escape	Cualquiera	Los caracteres se comparan con ellos mismos.
	\b	backspace
	\t	tabulador
	\r	enter
	\n	nueva línea
	\e	escape
Clases de Caracteres	.	Cualquier carácter
	[aeiou]	Cualquier carácter entre los corchetes.
	[^aeiou]	Cualquier carácter que no esté entre los corchetes.
	[a-z]	Para rangos de caracteres.
	\w	Cualquier carácter alfanumérica
	\s	Espacio
	\d	dígito
Caracteres especiales	^	Inicio de cadena
	$	Fin de cadena
	|	Disyunción
Cuantificadores	*	Una o más veces
	+	Al menos una vez
	?	No más de una
	{n}	Exactamente n veces

	{n,}	Por lo menos n veces
	{n,m}	De n a m veces

Por ejemplo, si deseamos una expresión regular que represente a todas las cadenas que repitan el valor ab una o más veces (ab, abab, abab, etc) tendremos la siguiente:

(ab)*

O por ejemplo si queremos una expresión que acepte todas aquellas cadenas que comiencen con un número y terminen con otro número:

\d.*\d

Es decir, tenemos la flexibilidad de tener una gran cantidad de patrones a utilizar, siempre y cuando conozcamos al lenguaje.

Una de las preguntas que ya deben quedar contestadas es si conviene más validar utilizando expresiones regulares o programando una función que haga esto, veamos los siguientes fragmentos de código:

```
Private Function EsAlfanumerico(ByVal cadena As String) As Boolean
    Dim caracteresValidos As String = "abcdefghijklmnopqrstuvwxyz0123456789"
    Dim downCadena As String = cadena.ToLower
    Dim caracterABuscar As String

    For posicionEnCadena As Integer = 0 To cadena.Length - 1
        caracterABuscar = downCadena.Substring(posicionEnCadena, 1)
        If caracteresValidos.IndexOf(caracterABuscar) = -1 Then
            Return False
        End If
    Next posicionEnCadena

    Return True
End Function
```

Este es el caso de una manera de detectar si una cadena es alfanumérica sin utilizar expresiones regulares. Definitivamente, resulta un poco enredada la manera en como valida la cadena. Ahora veamos la misma función con expresiones regulares:

```
Private Function EsAlfanumerico(ByVal cadena As String) As Boolean
    Dim patron As New System.Text.RegularExpressions.Regex("[a-z0-9]*")

    If patron.IsMatch(cadena.ToLower) Then
        Return False
    End If

    Return True
End Function
```

Definitivamente, la segunda forma es mucho más sencilla y entendible, ya que se crea un patrón con la expresión regular y luego se utiliza un método de este objeto para determinar si coincide con el patrón dado.

Como podemos apreciar, el poder de las expresiones regulares es muy grande, pues nos ahorra grandes líneas de código y nos evita generar ciclos complejos para evaluar una entrada. Por lo mismo, su valor para la validación de entrada de datos es inmenso y la importancia de su aprendizaje aumenta mucho más.

5.1.4. Negación de Servicios

Las técnicas para negar servicios son muy comunes y el framework está desprotegido de ellas. Una negación de servicios ocurre cuando no podemos utilizar el servicio que fue creado.

Un ejemplo que todos los amantes del fútbol recordamos es la final del campeonato clausura 2004. Ese partido fue disputado entre el Guadalajara y la Universidad Nacional, y causó una expectación sin precedentes. Ticketmaster tenía la labor de poner a la venta a través de su servicio de Internet y telefónico los boletos para los partidos. En esta ocasión hubo tantas peticiones a las 11am (hora que empezaba la venta) que la página simplemente no soportó la carga (se estiman más de 100 000 peticiones a esa exacta hora) y a todos los usuarios (incluyéndome) se nos negó el servicio. Como podemos apreciar, tanta gente, resultó en un ataque al servidor en cuestión, que al no ser programado adecuadamente, sufrió los efectos.

La negación de servicios puede clasificarse según su origen en:

- Colapso de aplicación.- el más conocido, si la aplicación falla, nadie más la puede usar.
- Memoria insuficiente.- sucede cuando alguna función, utiliza demasiado espacio en memoria, al coparla con información.
- CPU insuficiente.- resulta cuando alguna función, llena de sobremanera el CPU forzando a que el sistema no responda.
- Recurso insuficiente.- depende del recurso, un ejemplo clásico es cuando el disco duro se llena.

Por ejemplo, tomemos el siguiente segmento de código:

```
Dim contador As Integer = 99

Do While (contador < 100)
    contador = contador
Loop
```

Este caso, resultaría en una negación de servicio del tipo CPU insuficiente, pues entraríamos en un ciclo eterno (¿cuántas veces no nos ha pasado esto?). En este caso la forma más sencilla de remediarlos es detectar en nuestro código casos en los cuales podamos tener un ciclo infinito y arreglarlo.

Ahora tenemos este caso:

```
Dim numeroEntero As Integer = txtNombre.Text
```

Si nos ingresan un texto (como esperaríamos de una caja de texto con este nombre), el código se colapsaría, generando una negación de servicio del tipo Colapso de Aplicación.

De igual manera, la forma de evitarlo es a través de una correcta validación de nuestras entradas, así como la verificación de nuestro código que pueda colapsarse de alguna manera.

En el caso siguiente, tenemos un caso típico de insuficiencia de memoria:

```
Private Function CrearArreglo(ByVal tamaño As Int64) As Int64()
    Dim miArreglo(tamaño - 1) As Int64

    For a As Short = 0 To (tamaño - 1)
        miArreglo(a) = a
    Next a
End Function
```

Supongamos que tamaño ha sido debidamente validado. ¿Qué sucede si nos dan el máximo valor de una variable Int64? ¡¡¡¡Tendríamos un arreglo de 9 223 372 036 854 775 807 valores!!!! Seguramente a menos que tengamos memoria infinita podríamos tener ese tipo de arreglo en ella, por lo tanto, la memoria se vuelve insuficiente.

Para este tipo de ataques una medida sencilla pero eficiente es declarar un valor máximo, así, no tendremos de este tipo de ataques:

```
Private Function CrearArreglo(ByVal tamaño As Int64) As Int64()
    Dim miArreglo(tamaño - 1) As Int64
    Const MAXIMO_TAMAÑO = 100

    If tamaño <= MAXIMO_TAMAÑO Then
        For a As Short = 0 To (tamaño - 1)
            miArreglo(a) = a
        Next a
        Return miArreglo
    Else
        Throw New ArgumentException("Número máximo de elementos rebasado")
    End If
End Function
```

Otro caso en el cual podemos tener un ataque del tipo recurso insuficiente es aquel en el cual, anexamos a un archivo la información de este arreglo:

```
Private Function EscribirNumeros(ByVal tamaño As Int64) As Boolean
    Dim miArreglo(tamaño - 1) As Int64
    Dim numeroDeArchivo As Integer = FreeFile()

    FileOpen(numeroDeArchivo, "arreglo.txt", OpenMode.Append)

    For a As Short = 0 To (tamaño - 1)
        PrintLine(numeroDeArchivo, a.ToString)
    Next a

    FileClose(numeroDeArchivo)

    Return True
End Function
```

Caso similar al anterior, si nos pasan un número enorme como parámetro, tendremos un archivo que puede medir varios gigas de información, esto podría terminar en que el disco se llene así negándonos servicio por falta de espacio en disco duro. También, como en el caso anterior es necesario validar el número máximo de elementos permitidos:

```
Private Function EscribirNumeros(ByVal tamaño As Int64) As Boolean
    Dim miArreglo(tamaño - 1) As Int64
    Dim numeroDeArchivo As Integer = FreeFile()
    Const MAXIMO_TAMAÑO = 100


    FileOpen(numeroDeArchivo, "arreglo.txt", OpenMode.Append)

    If tamaño <= MAXIMO_TAMAÑO Then
        For a As Short = 0 To (tamaño - 1)
            PrintLine(numeroDeArchivo, a.ToString)
        Next a
        Return True
    Else
        Throw New ArgumentException("Número máximo de elementos rebasado")
    End If

    FileClose(numeroDeArchivo)

    Return False
End Function
```

5.1.5. Ataques a Directorios

Muy parecido a la inyección SQL, este tipo de ataque se basa en la posibilidad de poder navegar por los directorios desde la línea de comandos. Por ejemplo, supongamos que tenemos nuestro proyecto en:

C:\Mis Proyectos\Cotizador\frmGuardarArchivo.vb

Y desde ahí aceptamos el nombre de un archivo para guardar la información que hemos obtenido. Por ejemplo, si nos dan "miArchivo.txt" tendremos en nuestro directorio "cotizador" un archivo con este nombre. Ahora supongamos que ingresan la siguiente línea:

..\..\..\..\..\Windows\control.exe

Como nosotros no hemos validado la entrada, se sobrescribiría el archivo control.exe en el directorio de Windows, es decir ¡¡nos quedaríamos sin panel de control!!

La forma de operación es muy sencilla, cada "..\" quiere decir, sube de directorio. Cuando se interpreta esta línea, sucede lo siguiente:

C:\Mis Proyectos\Cotizador	**"..\..\..\..\..\Windows\control.exe"**
C:\Mis Proyectos	**"..\..\..\..\Windows\control.exe"**
C:	**"..\..\..\Windows\control.exe"**
C:	**"\Windows\control.exe**
C:\Windows	**"\control.exe"**
C:\Windows\control.exe	

Y por lo tanto, el ataque tiene éxito.

Ante este tipo de ataques, la forma más sencilla de protección, es aquella, donde podemos valernos de la función Application.StartUp.Path que nos devuelve el directorio donde estamos trabajando, de esta manera al compararlo con el nombre del archivo completo que obtenemos a través de la función Path.GetFullPath(). Por lo tanto, el framework, nos da las herramientas necesarias para protegernos de este tipo de ataques:

```
Private Sub GuardarArchivo(ByVal nombreArchivo As String, ByVal texto As String)
    Dim rutaCompleta As String = Path.GetFullPath(nombreArchivo)
    Dim numeroDeArchivo As Integer = FreeFile()

    If Path.GetDirectoryName(rutaCompleta).ToLower = Application.StartupPath.ToLower Then
        FileOpen(numeroDeArchivo, "nombreArchivo", OpenMode.Output)
    End If
End Sub
```

En este caso, tenemos el uso de ambas funciones para establecer que el directorio suministrado sea el correcto. En la figura 5.1 podemos apreciar qué sucedería con las diferentes variables del programa cuando se suministra una cadena como la propuesta en el ejemplo anterior:

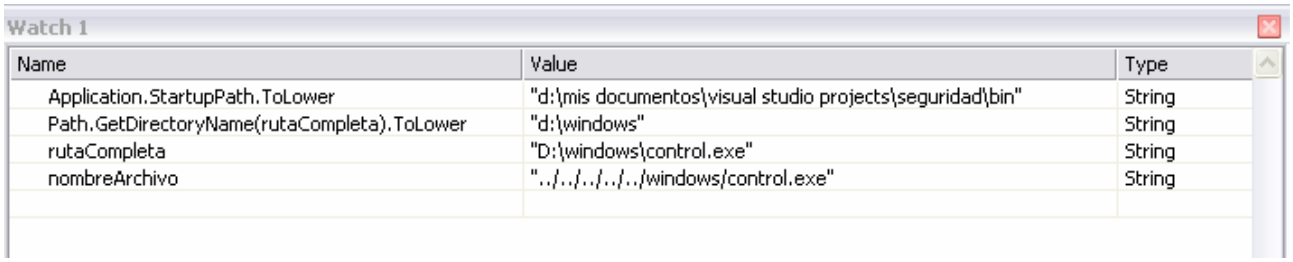


Figura 5.1 Ventana de Watch para función GuardarArchivo

En la figura se ve con claridad que la prueba fallaría pues a pesar de que se le ha enviado un directorio y archivo válido, los directorios de inicio de proyecto (Application.StartupPath) y el directorio donde planea escribir (Path.GetDirectory) no son iguales, por lo que nos estamos saliendo de nuestra área de trabajo.

Por lo mismo esta técnica es bastante efectiva para evitar que el usuario malicioso ande navegando por toda nuestra computadora.

5.1.6. Inyección SQL

Dos son las técnicas más importantes para la defenderse de este tipo de ataques, ya que cada proveedor maneja sus propias características y propiedades (por ejemplo Microsoft Access no acepta varias sentencias en una misma línea):

- Validar la entrada de datos
- Parametrizar las búsquedas

a) Validaciones

Una de las formas más sencillas (y por cierto bastante efectivas) de prevenir un ataque por inyección SQL. La técnica es sencilla, no permitir por ningún motivo la inserción de caracteres no permitidos como la comilla simple, los paréntesis, etc.

Este es un caso muy común para el uso de expresiones regulares, por ejemplo, supongamos que queremos que ingresen un nombre, podemos crear una función como la siguiente:

```
Private Function EsNombre(ByVal cadena As String) As Boolean
    Dim patron As New System.Text.RegularExpressions.Regex("^[a-z]+$")

    If patron.IsMatch(cadena.Trim.ToLower) Then
        Return False
    End If

    Return True
End Function
```

Con esta validación, estamos seguros que no se insertarán más caracteres que estrictamente los válidos, además de garantizar que se tiene el total de la línea por los

caracteres de inicio y fin de la misma. Esto en conjunto con las propiedades propias de la TextBox, pueden hacer una línea de defensa sumamente robusta.

b) Parámetros

Como vimos los ataques de SQL se basan en la vulnerabilidad más importante que la mayoría de los programadores novatos de bases de datos cometen, la concatenación para formar sentencias SQL. Por lo mismo, esta técnica está dirigida precisamente a este tipo de vulnerabilidades.

Ello consiste en anexar una variable que será sustituida por el mismo manejador de base de datos, por lo que todo lo que se encuentre en la variable será tratado sin excepción como argumento, asegurando así que no se ejecute absolutamente nada que no sea lo deseado (muy parecido a alguna forma de operación de PHP). Veamos la siguiente instrucción SQL:

```
Dim sqlQuery As String = "SELECT nombreEmpresa FROM Empresa WHERE idEmpresa= @idEmpresa"
```

Esta instrucción tiene en lugar de una concatenación una variable @idEmpresa que indica al manejador que lo que se encuentre como valor de esa variable será sustituido. Cuando manejamos ADO.NET, debemos crear un objeto SQLCommand, donde tendremos esta opción y agregaremos las variables de la siguiente manera:

```
cmmEmpresa.Parameters.Add("@idEmpresa", txtNombre.Text)
```

Por lo mismo, tendremos total control sobre lo que se pase como parámetro y se asegurará que no habrá vulnerabilidades del tipo Inyección SQL.

Aquí tenemos otro ejemplo 100% funcional:

```
Dim cmmSelect As New SqlClient.SqlCommand

cmmSelect.CommandText = "SELECT nombre, ap_mat, ap_pat "
cmmSelect.CommandText &= "FROM datosgenerales "
cmmSelect.CommandText &= "WHERE password=@ClaveDeUsuario "
cmmSelect.CommandText &= "AND login=@IDUsuario "

cmmSelect.Parameters.Add("ClaveDeUsuario", txtClave.Text)
cmmSelect.Parameters.Add("IDUsuario", txtIDUsuario.Text)
```

5.1.7. Criptografía

El Framework .NET nos ofrece para esta tarea, el namespace System.Security.Cryptography en él podemos encontrar absolutamente todo lo necesario para poder utilizar los algoritmos más frecuentes ya sean simétricos o asimétricos.

En la figura 5.1 podemos apreciar las principales clases dentro de este namespace:

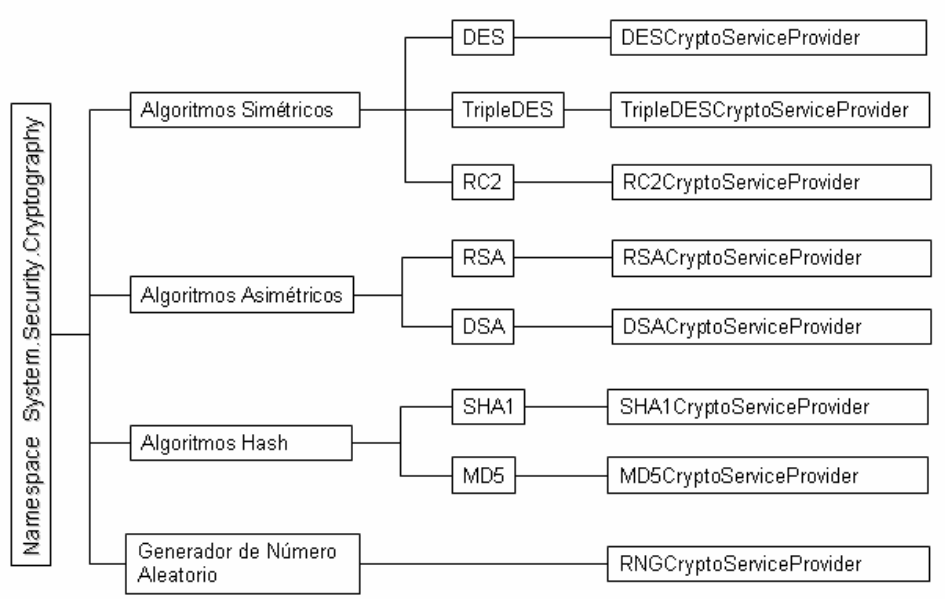


Figura 5.1 Principales Clases del Namespace Cryptography

Como podemos apreciar, tenemos a nuestra disposición suficientes elementos para poder trabajar de una manera segura con el poder de la criptografía a nuestra mano. Dentro de los algoritmos simétricos tenemos al poderoso DES, al 3DES y RC2, sólo por citar algunos. Dentro de los Asimétricos tenemos a los dos más importantes, que son RSA y DSA. Y por si no fuera suficiente con ello, tenemos algoritmos Hash que si bien no son del todo algoritmos de encriptación cumplen con diversas funciones para la seguridad. Finalmente tenemos a un generador de números aleatorios que es 100% seguro, a diferencia de algunos otros que ya se han analizado.

.NET utiliza la librería CryptoAPI para generar los algoritmos que podemos ver en la figura 5.1 (y muchos más). Esta librería contiene a todos los CryptoServiceProvider que son los encargados de contener los algoritmos necesarios para la encriptación.

La plataforma nos facilita muchísimo las cosas, tanto así, que podemos encriptar ya sean archivos, datos de entrada, constantes del programa, etc. Todo ello gracias a una clase llamada CryptoStream donde se realizará la encriptación de cualquier flujo de datos. Esta clase funciona como se muestra en la figura 5.2:



Figura 5.2 Modo de funcionamiento de la clase CryptoStream

Como vemos, el CryptoStream contiene al algoritmo de encriptación y al flujo de datos. Es aquí donde se realizan las operaciones de cifrado y descifrado, y por lo tanto es muy importante en el proceso de encriptamiento.

a.1) Encriptación Simétrica

En este caso analizaremos como encriptar utilizando el afamado algoritmo DES, a continuación presento un código típico:

```
1  Function EncriptarDES(ByVal mensajeEnClaro As String, ByVal clave As String) As String
2      ' Recibe:       La clave y el mensaje en claro para poder encriptar.
3      ' Regresa:      El criptograma obtenido.
4      ' Descripción:  Realiza el algoritmo simétrico DES.
5
6      ' Se crea el algoritmo de encriptamiento
7      Dim crypto As New DESCryptoServiceProvider
8
9      ' Convertimos el mensaje en claro a un arreglo de bytes
10     Dim codificacionASCII As New ASCIIEncoding
11     Dim byteMensajeEnClaro() As Byte = codificacionASCII.GetBytes(mensajeEnClaro)
12
13     ' Obtenemos el vector inicial (IV) y la clave de 8 bytes (64bits)
14     Dim salt(0) As Byte
15     Dim claveDerivada As New PasswordDeriveBytes(clave, salt)
16     Dim byteClaveDerivada() As Byte = claveDerivada.GetBytes(8)
17
18     ' Añadimos las claves y vector inicial al algoritmo
19     crypto.Key = byteClaveDerivada
20     crypto.IV = claveDerivada.GetBytes(8)
21
22     ' Creamos un flujo donde poner el resultado de la encriptación
23     Dim flujoResultado As New MemoryStream
24
25     ' Creamos un flujo de encriptamiento
26     Dim encriptador As New CryptoStream(flujoResultado, crypto.CreateEncryptor, _
27                                         CryptoStreamMode.Write)
28    ' Escribimos el mensaje en claro en el flujo de encriptamiento
29     encriptador.Write(byteMensajeEnClaro, 0, byteMensajeEnClaro.Length)
30     encriptador.FlushFinalBlock()
31
32     ' Regresamos la encriptación obtenida en forma de cadena
33     Return Convert.ToBase64String(flujoResultado.ToArray)
34 End Function
```

Como podemos observar, es bastante sencillo, obtener dentro de .NET un código 100% funcional que nos permita tener un algoritmo de encriptamiento muy robusto.

Siguiendo las recomendaciones antes descritas, el código está autodocumentado, por lo que será muy sencillo comprender el flujo de datos. Primero se reciben el mensaje en claro y la clave a utilizar (línea 1), que por cierto, no importa que sea de más de 8 bytes. Después de esto, viene el primer bloque, donde indicamos que utilizaremos el proveedor de servicios DES (línea 7).

En las líneas 14-16, resalta la importancia de tener tanto el mensaje en claro como la clave en bytes, pues estos serán ingresados al flujo en forma de bits. Además, se añaden al proveedor de servicios la clave y el vector de inicio (líneas 19-20).

Finalmente, creamos un flujo en memoria (línea 23) para guardar el resultado y es precisamente en la línea 28 donde ocurre la encriptación, cuando utilizamos el método write del CryptoStream, que al escribir en el flujo encriptado, escribe ya cifrado el texto en claro.

La función es por demás sencilla, pero muy poderosa. Lo más importante, es que una vez que se ha entendido este proceso, éste se repite en su mayoría para los demás proveedores de servicio, como veremos en ejemplos posteriores.

a.2) Encriptación Asimétrica

En este caso utilizaremos las propiedades del framework para generar claves privadas y públicas, para así tener un objeto RSA y poder utilizarlo para cifrar y descifrar mensajes. Por lo mismo, es importante que hagamos primero un generador de claves como el que se muestra a continuación:

```
1  Function CrearClaves() As String
2      ' Regresa:      Una cadena representando el objeto RSA con ambas claves.
3      ' Descripción:  Genera una objeto RSA de donde poder obtener ambas claves.
4
5      ' Definimos un nuevo objeto RSA
6      Dim rsa As New RSACryptoServiceProvider
7
8      ' Regresamos la cadema que representa al objeto RSA
9      Return rsa.ToXmlString(True)
10
11 End Function
```

En la línea 1 del código, tenemos la declaración de la función seguida de una amplia explicación de la manera en como funciona el código. En la línea 6 definimos un objeto RSA. El objeto RSA, a la hora de ser creado trae con él la clave pública y la clave privada. Por lo mismo en la línea 9 devolvemos del objeto RSA una cadena con formato XML que representa a todo el objeto RSA, el atributo True especifica que debe anexar a la información del objeto la clave privada.

Este objeto lo debemos tratar con mucho cuidado y lo debemos almacenar, pues contiene a este ente central del algoritmo, por lo que para descifrar lo que sea encriptado con esta clave, debemos tener al objeto tal cual lo regresa la función.

Siguiendo la lógica del algoritmo, ahora utilizaremos una clave pública. Como la forma en que nos enviarán esta clave, es por medio de otro objeto RSA, es importante tener una función que nos devuelva:

```
1  Function ObtenerClavePublica(ByVal cadenaRSA As String) As String
2      ' Recibe:      Una cadena representando el objeto RSA con ambas claves.
3      ' Regresa:     Una cadena representando el objeto RSA con la clave pública.
4      ' Descripción: Obtiene la clave pública de un objeto RSA.
5
6      ' Creamos un objeto RSA
7      Dim rsa As New RSACryptoServiceProvider
8
9      ' Anexamos al objeto RSA la cadena con el objeto
10     rsa.FromXmlString(cadenaRSA)
11
12     ' Regresamos la clave pública en forma de objeto RSA
13      Return rsa.ToXmlString(False)
14
15 End Function
```

Esta función, recibe la cadena que representa a un objeto RSA, crea un objeto RSA (línea 7) y reemplaza la clave pública que trae con la que tenemos de argumento (línea 10) para regresarnos la clave pública. El hecho de pasarle el parámetro False, a diferencia de la función pasada, es para indicarle que no contiene una clave privada.

Muy bien, ahora que ya tenemos ambas claves, es hora de ponernos a encriptar. Para ello, tendremos que crear una función como las que hemos hecho antes, que en una sola línea de código podamos encriptar:

```
1  Function CifrarRSA(ByVal mensajeEnClaro As String, ByVal clavePublica As String) As String
2      ' Recibe:      El mensaje en claro y la clave pública para realizar el encriptado RSA.
3      ' Regresa:     El criptograma resultado del algoritmo RSA.
4      ' Descripción: Aplica el algoritmo RSA al mensaje en claro y genera el respectivo
5      '              criptograma.
6
7      Dim rsa As New RSACryptoServiceProvider ' Creamos un objeto RSA
8      Dim byteMensajeEnClaro() As Byte ' Versión en bytes de mensajeEnClaro
9      Dim byteCriptograma() As Byte ' Versión en bytes de criptograma
10     Dim codificacionASCII As New ASCIIEncoding ' Objeto para conversión de variables
11
12     ' Anexamos la clave pública al objeto RSA
13     rsa.FromXmlString(clavePublica)
14     ' Rellenamos las versión en bytes con la original
15     byteMensajeEnClaro = codificacionASCII.GetBytes(mensajeEnClaro)
16     ' Encriptamos el mensaje en claro con el objeto RSA
17     byteCriptograma = rsa.Encrypt(byteMensajeEnClaro, False)
18     ' Limpiamos el RSA
19     rsa.Clear()
20
21     ' Devolvemos el criptograma
22     Return Convert.ToBase64String(byteCriptograma)
23
24 End Function
```

Como siempre, los parámetros se describen solos, recibimos el mensaje en claro y la clave pública. Después declaramos 4 variables (líneas 7-10) que están debidamente comentadas. En la línea 13 insertamos la clave pública en el objeto rsa, luego en la línea 15 convertimos el mensaje en claro a bytes. La línea más importante es la 17, dónde realizamos la encriptación, mediante el objeto RSA. El parámetro False, sirve para darle

mayor portabilidad al algoritmo. Finalmente, en la línea 19 limpiamos el objeto y devolvemos el criptograma como cadena de texto.

Ahora que tenemos la función de encriptado, necesitamos la de descifrado. En este caso, se deben pasar la clave privada del usuario. Como recordamos, no podemos tener claves privadas, por lo que tendremos que enviarle un objeto que represente el RSA del usuario.

La forma de operación es muy similar a todas las anteriores. Primero recibimos el criptograma y la cadena que representa al objeto RSA (línea 1) luego, declaramos las mismas variables que en el cifrado (líneas 7-10) y luego insertamos el objeto RSA recibido en el objeto RSA creado (línea 13) y cambiamos a bytes el criptograma (línea 15) y desciframos en la línea 17, tal y cual lo hicimos con el caso anterior; finalmente devolvemos el mensaje en claro.

```
1  Function DescifrarRSA(ByVal criptograma As String, ByVal cadenaRSA As String) As String
2      ' Recibe:      El criptograma y un objeto RSA para realizar el descifrado RSA.
3      ' Regresa:     El mensaje en claro resultado del algoritmo RSA.
4      ' Descripción: Aplica el algoritmo RSA al criptograma y genera el respectivo
5      '              criptograma.
6
7      Dim rsa As New RSACryptoServiceProvider ' Creamos un objeto RSA
8      Dim byteMensajeEnClaro() As Byte ' Versión en bytes de mensajeEnClaro
9      Dim byteCriptograma() As Byte ' Versión en bytes de criptograma
10     Dim codificacionASCII As New ASCIIEncoding ' Objeto para conversión de variables
11
12     ' Rellenamos el objeto RSA vacío con el recibido
13     rsa.FromXmlString(cadenaRSA)
14     ' Rellenamos las versión en bytes con la original
15     byteCriptograma = Convert.FromBase64String(criptograma)
16     ' Desencriptamos el criptograma con el objeto RSA
17     byteMensajeEnClaro = rsa.Decrypt(byteCriptograma, False)
18     ' Limpiamos el RSA
19     rsa.Clear()
20
21     ' Devolvemos el criptograma
22     Return codificacionASCII.GetString(byteMensajeEnClaro)
23
24 End Function
```

Como podemos apreciar, el encriptado y descifrado con algoritmos simétricos es bastante sencillo cuando nos valemos del poder del framework. Es importante notar que aquí se deben manejar todo tipo de precauciones para mantener las claves en secreto.

a.3) Algoritmos Hash

Si bien, no se consideran del todo desarrollados para los procesos de encriptación, pues sólo se pueden hacer en un sentido, estos algoritmos sirven para las tareas de autenticación de una manera excelente.

Su modo de operación es bastante sencillo, pues una cadena de texto sólo puede dar como resultado una y sólo una cadena de transformación. Sin embargo, esta cadena de transformación no puede devolvernos la cadena original.

Esto sirve de sobremanera para la autenticación, pues uno de los usos más solicitados es el de guardar contraseñas. Es decir, en la base de datos, no guardamos la contraseña que escogió el usuario, sino sólo su cadena de transformación. De esta manera, cuando el usuario teclea su clave, ésta es transformada y se compara con la cadena de transformación guardada en la base Así, si la base está comprometida, el usuario malicioso, no tendrá acceso a las claves, sino a las cadenas de transformación, las cuales por ellas mismas no sirven de nada.

El framework .NET trae las librerías necesarias para generar estas cadenas de transformación de una manera rápida y eficiente. En este ejemplo veremos el método de SHA1 que es un método bastante genérico y muy poderoso par aplicaciones de escritorio:

```
1  Function CrearSHA1(ByVal cadena As String) As String
2      ' Recibe:       La cadena a transformar.
3      ' Regresa:      El hash obtenido.
4      ' Descripción:  Realiza el algoritmo SHA1 sobre la cadena de entrada.
5
6      Dim codificacionASCII As New ASCIIEncoding ' Objeto para conversión de variables
7
8      ' Guarda la entrada en un arreglo de bytes
9      Dim byteCadena() As Byte = codificacionASCII.GetBytes(cadena)
10
11     ' Creamos un objeto del tipo Sha1 que generará el criptograma.
12     Dim sha1 As New SHA1CryptoServiceProvider
13
14     ' Crea y llena la variable hash con el resultado de la conversión.
15     Dim hash() As Byte
16     hash = sha1.ComputeHash(byteCadena)
17
18     ' Regresa la cadena transformada
19     Return Convert.ToBase64String(hash)
20
21 End Function
```

Como podemos observar, se recibe una cadena cualquiera y lo primero es declarar el objeto para la conversión de variables (línea 6) y la versión en bytes de la cadena de entrada (línea 9). Luego creamos un objeto sha1, que como hemos visto, es el encargado de la encriptación (ya que forma parte de APICrypto) y en la línea 16 realizamos la "encriptación" que nos arroja la cadena de transformación que devolvemos en forma de cadena (línea 19).

5.1.8. Manejador de Errores

La estructura y forma de operación del manejador de errores ya se han visto, ahora es momento de analizarlos un poco más a fondo. Para ello analizaremos dos casos que

son de vital importancia para que nuestra aplicación no caiga en negación de servicios, éstos son: manejadores anidados y la generación de errores personalizados.

Para explicar estos dos temas haremos uso de la siguiente función:

```
Private Function GeneraExcepcion() As Integer
    Dim valor As Integer = 1

    Return (valor / 0)
End Function
```

Que genera una excepción de desbordamiento del tipo división entre cero. Esta función siempre genera una excepción sin importar las situaciones en la que sea llamada.

a) Manejadores Anidados

Una de las características más importantes de los manejadores de errores, es que podemos anidarlos, para darle una estructura lógica a nuestro programa:

```
1  Private Sub MuestraDeTry()
2      Try
3          MsgBox("Estamos por generar el error.")
4          Try
5              Dim resultado As Integer = GeneraExcepcion()
6              MsgBox("Como se genera error no vemos este mensaje.")
7          Catch ex As Exception
8              MsgBox("Se ha generado un error de divisón entre cero." & vbCrLf & vbCrLf & _
9                     "Descripción: " & ex.Message, MsgBoxStyle.Exclamation, "Error Detectado")
10         End Try
11         MsgBox("El error fue generado.")
12     Catch ex As Exception
13         MsgBox("Se ha generado un error." & vbCrLf & vbCrLf & "Descripción: " _
14                & ex.Message, MsgBoxStyle.Exclamation, "Error Detectado")
15     End Try
16 End Sub
```

En este ejemplo, la subrutina MuestraDeTry, contiene dos Manejadores anidados. La estructura es sencilla, primero ingresamos a un primer bloque try y visualizamos un mensaje que nos dice que vamos a generar un error (línea 3), luego entramos al segundo bloque try y generamos una excepción con la función GeneraExcepcion, esta excepción es "cachado" por el bloque Try (línea 7) y visualizamos el mensaje dentro de este bloque, evitando así el colapso del programa (línea 8). Este bloque Try, desecha la excepción (pues ya le hemos dado el trato que merece) y regresa el control al bloque Try anterior que nos muestra el mensaje de confirmación de error (línea 11), terminando finalmente con la subrutina.

Como podemos apreciar, es de esta manera como siempre podremos tratar todo tipo de excepción sin afectar el flujo de nuestro programa de ninguna manera. Por lo mismo tenemos una gran flexibilidad en cómo trataremos las diferentes excepciones que obtengamos o generemos.

Otra medida de seguridad es el encerrar en un bloque try catch todo nuestro programa, ya sea desde un módulo que contenga a la función main, o desde un botón que generalmente dispara todos los procedimientos de nuestro programa, con ello aseguramos que aunque ocurra una excepción en cualquier lugar de nuestro programa, siempre será resuelta por lo menos de una manera decorosa.

```
Sub main()
    Try
        IniciarAplicacion()
    Catch ex As Exception
        MsgBox("Se ha generado un error inesperado." & vbCrLf & vbCrLf & _
               "Descripción: " & ex.Message, MsgBoxStyle.Exclamation, "Error Detectado")
    End Try
End Sub
```

b) Generando Errores

¿Cómo probar que nos hemos quedado sin memoria? Una opción es llenarla y cuando esto ocurra, detectar el error y tratar con él. Generalmente esta aproximación no sólo suena muy agresiva sino muy poco práctica. Por lo mismo el Framework .NET nos proporciona la posibilidad de lanzar excepciones personalizadas, que pueden servir para que nosotros podamos generar un error cuando creamos que sea necesario y así pode trabajar a nuestras anchas. Así, de esta manera, podemos simular la excepción que nos indica que ya no contamos con más memoria:

```
Throw New OutOfMemoryException
```

Esta instrucción nos arroja exactamente la misma excepción que se generaría al quedarnos sin memoria, sin la necesidad de agotarla en nuestra memoria. Potro caso interesante es el hecho de que podemos arrojar cualquier tipo de excepción, sin necesidad de que realmente exista:

```
Private Sub CasosDeExcepciones(ByVal tipoExcepcion As Integer)
    Try
        Select Case (tipoExcepcion)
            Case 1
                Throw New ArgumentException
            Case 2
                Throw New OverflowException
            Case 3
                Throw New BadImageFormatException
            Case 4
                Throw New OutOfMemoryException
            Case 5
                Throw New Exception("Nombre de usuario no válido.")
            Case Else
                Throw New Exception("Excepción no conocida.")
        End Select
    Catch ex As Exception
        MsgBox("Se ha generado un error." & vbCrLf & vbCrLf & "Descripción: " _
                & ex.Message, MsgBoxStyle.Exclamation, "Error Detectado")
    End Try
End Sub
```

En este fragmento de código tenemos una subrutina que genera excepciones y una excepción personalizada, que nos indica que el nombre no es correcto, además de una última excepción que indica que la excepción no es conocida.

Esto también nos puede ser de utilidad por si queremos que a pesar que suceda un error de un tipo, mostremos que es otro. Esto principalmente podría servir para despistar al atacante, pues recordemos que no debemos dar demasiada información de nuestra forma de operación a nadie, pues esto sólo le indica al atacante como está funcionando su ataque:

```
1  Private Sub MuestraDeTry()
2      Try
3          MsgBox("Estamos por generar el error.")
4          Try
5              Dim resultado As Integer = GeneraExcepcion()
6          Catch ex As Exception
7              Throw New NullReferenceException
8          End Try
9          MsgBox("El error fue generado.")
10     Catch ex As Exception
11         MsgBox("Se ha generado un error." & vbCrLf & vbCrLf & "Descripción: " _
12                 & ex.Message, MsgBoxStyle.Exclamation, "Error Detectado")
13     End Try
14 End Sub
```

Aquí, a pesar de que tenemos un error de desbordamiento de buffer, informamos que fue un error de referencia nula (como que no exista un recurso). Hay que poner atención que en este caso, la línea 9 no se mostraría ya que sólo se arrojó una excepción que será "cachada" por el match exterior.

Finalmente, es importante mencionar que a pesar del poder de las excepciones, siempre será mejor establecer barreras que eviten que se genere la excepción, y que ésta sea nuestra última barrera de defensa, pues la creación de la excepción es lenta y consume muchos ciclos del procesador, lo que podría ocasionar que nuestro sistema consuma más recursos de los necesarios.

Conclusiones

Después de haber explorado con detenimiento el presente trabajo puedo llegar a una gran cantidad de conclusiones. Todas ellas parte de un proceso, no sólo del estudio y creación de esta tesis, sino también de experiencias personales en las cuales, he tenido la oportunidad de poner en práctica algunas de las técnicas estudiadas y he visto como se desarrollan.

Primero quisiera retomar los objetivos y analizarlos para comprobar si se ha logrado conseguirlos:

- Explorar y diseñar las estrategias a seguir para la creación de software seguro desde su diseño hasta su liberación.
- Analizar porqué VB.NET es una herramienta de desarrollo profesional.
- Desarrollar una guía de costumbres para el desarrollo de aplicaciones seguras.

Estos fueron los objetivos que impulsaron a la realización de esta tesis. Así que analizaré uno por uno como se fueron atacando para lograr su realización.

El primero de ellos consiste en: "explorar y diseñar las estrategias a seguir para la creación de software seguro desde su diseño hasta su liberación". Este objetivo en particular queda explorado y diseñado en el capítulo 4.1 "Ingeniería de Software", donde se analizan todos los aspectos relacionados a las estrategias para la creación de software seguro. Este tipo de estrategias van enfocadas, para su mejor apreciación, desde la planeación y formación del equipo de trabajo, hasta la liberación del producto. Por ello, tenemos una guía que si bien sencilla, puede hacer la diferencia entre un sistema seguro y otro que no lo sea.

El segundo de los objetivos: "Analizar porqué VB.NET es una herramienta de desarrollo profesional" queda mostrado desde el análisis de la plataforma .NET, en el capítulo uno, que nos demuestra de una manera sencilla, como este nuevo elemento de software, nos ayuda a separar, de una manera eficiente y funcional, el sistema operativo del ambiente de operación del software diseñado. Además de especificar los puntos básicos de operación, que van desde la máxima portabilidad, hasta el uso de características uniformes de seguridad para cualquier lenguaje de programación orientado a .NET. Por lo mismo, apoyándose en el capítulo dos, podemos entender el poder de la nueva versión de este lenguaje de programación, que queda demostrado en el capítulo 5, cuando tenemos la oportunidad de ver código aplicado, y darnos cuenta como el Framework .NET y Visual BASIC se han fusionado de una manera amigable y sencilla, dando la mayor productividad que jamás halla existido para este lenguaje de programación.

El tercero de los objetivos “desarrollar una guía de costumbres para el desarrollo de aplicaciones seguras” queda analizado en las secciones 4.2 y 4.3 donde analizo varias creencias antiguas con respecto a la creación de software, se revisan algunos temas básicos como estandarización de código, creación de código eficiente y algunos otros temas de diseño de software que apoyan a que el programador piense antes de escribir su código en todas las características que debe aportar, de tener buenas costumbres al programar y de diseñar su aplicación antes de empezar a programar. Esto sin lugar a dudas, es una de las características más difíciles de lograr, ya que en general los programadores en México no cuentan con esta educación y se resisten a cambiar sus métodos de programación, por lo mismo, es aún más importante esta sección, pues al comprenderla y emplearla, es posible crear escuela práctica en los grupos de trabajo donde demostremos su funcionalidad.

Es así como todos y cada uno de los objetivos quedan cubiertos dentro de este trabajo de tesis, que ha sido el resultado una amplia investigación en diversas publicaciones que tratan de darle un enfoque teórico o práctico a la creación de software, pero que no habían sido reunidos como un todo, y que a través del estudio de los diferentes materiales me pude percatar que eran más que trabajos aislados, que eran todo un compendio de buenas costumbres, métodos y técnicas para la creación de aplicaciones seguras.

Esto se pudo poner en práctica en diversas circunstancias, si no bien en su totalidad, sí parcialmente y los resultados son por demás interesantes.

Durante un curso del Lenguaje de Programación Visual BASIC.NET, que tuve el honor de impartir, se presentó la oportunidad de presentar las normas básicas para la estandarización de código, la cual fue muy bien aceptada entre los estudiantes (todos ellos estudiantes de la Facultad de Ingeniería de la UNAM), quienes mencionaron la facilidad con que podían seguir los ejemplos en clase y notaron una mejor relectura de código, una vez pasado algún tiempo.

Otro ejemplo claro es el proyecto de Simulación de Avance de Créditos (del cual se presentó su modelo en la sección 4.1), y estuvo a cargo de dos estudiantes de los cursos antes mencionados, que adoptaron los estándares y algunas otras especificaciones de “primera línea de defensa” para la realización de su proyecto, del cual recibieron una amplia aprobación por diversas instancias de la Facultad de Ingeniería.

Finalmente la última experiencia con este tipo de estrategias, la pude obtener, al impartir una sesión al grupo de “Fundamentos de Seguridad Informática” en la cual, se mostraron las características básicas tratadas en el capítulo cinco del presente trabajo. La aceptación fue grande, y los alumnos mostraron interés al ver cómo ante sencillos procedimientos se podía llegar a un software mucho más robusto.

Es así, como este trabajo ha cumplido los objetivos de manera práctica y teórica, y además permite esperar un uso didáctico en la materia de Software Seguro que se

impartirá a partir del semestre 2006-2, ya que todo el material presentado es sin lugar a dudas un compendio de información nada despreciable para lograr los objetivos de esta materia.

Finalmente quisiera recalcar la importancia que ha tenido esta tesis en mi formación profesional, ya que me ha permitido desarrollar varios puntos importantes. Entre ellos, destaco el conocimiento adquirido de la plataforma .NET que al principio de este proyecto era nulo, y que después de éste, he logrado dominar de cierta manera, lo cual me permite completar mi formación ingenieril con formación técnica; que como sabemos es extremadamente importante para el mundo laboral.

También destaco el hecho de que he ampliado mis conocimientos en la elaboración de software seguro, que hasta antes del presente trabajo eran prácticamente inexistentes, pero que debido al estudio y seguimiento de los conceptos básicos de seguridad, me es posible comprender de manera amplia.

Finalmente el más importante de todas las aportaciones que me ha dado el presente trabajo, es la aplicación de muchísimos de mis conocimientos adquiridos en la facultad, pues tuve la oportunidad de aplicarlos al estudiar temas avanzados que requerían una formación teórica importante y que los mismos autores llegaban a recalcar que era imprescindible para su correcta asimilación y aprovechamiento.

Glosario

ALGOL (Algorithmic Language).- se denomina Algol a un lenguaje de programación muy conocido en los años 60.

ANSI (American National Standards Institute).- el Instituto Nacional Estadounidense de Estándares es la principal organización encargada de promover el desarrollo de estándares tecnológicos en Estados Unidos. ANSI es miembro de la Organización Internacional para la Estandarización (ISO) y de la Comisión Electrotécnica Internacional (International Electrotechnical Commission, IEC).

API (Application Programming Interface).- una Interfaz de Programación de Aplicaciones es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla.

Archivos adjuntos.- archivos que se anexan a un correo electrónico para su distribución por este medio.

ARPANET (Advanced Research Projects Agency Network).- esta red de computadoras fue creada por encargo del Departamento de Defensa de los Estados Unidos como medio de comunicación para los diferentes organismos estadounidenses en 1969.

ASCII (American Standard Code for Information Interchange).- El Código Estadounidense Estándar para el Intercambio de Información es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno. Creado aproximadamente en 1963 por el Comité Estadounidense de Estándares (ASA) como una evolución de los conjuntos de códigos utilizados entonces en telegrafía.

Auditor.- persona encargada de evaluar si los procesos o productos cumplen con las características de un estándar en específico y determina si deben o no ser certificados.

AWK.- es un lenguaje de programación de propósito general que fue diseñado para procesar datos basados en texto, ya sean archivos o flujos de datos. El nombre AWK deriva de los apellidos de los autores: Alfred V. Aho, Peter J. Weinberger, y Brian W. Kernighan.

Beta.- modelo prototipo que generalmente se pone al alcance de usuarios calificados o avanzados para que se realicen pruebas y puedan ser reportados todos los errores que haya en la aplicación.

Clases.- Las clases son una descripción abstracta de un grupo de objetos que comparten características (atributos) y operaciones (métodos).

Cobol (COmmon Business Oriented Language).- nació en 1960, del deseo de crear un lenguaje de programación "universal", que pudiera ser usado en cualquier computadora y que estuviera orientado principalmente a los negocios, es decir, a la llamada informática de gestión.

Compilador.- un compilador acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente independiente, que puede ejecutarse tantas veces como se quiera. Este proceso de traducción se conoce como compilación.

Copernic.- motor de búsqueda en múltiples buscadores de internet. Hace múltiples búsquedas paralelas y las muestra al usuario de una manera agradable.

Cuadro de diálogo modal.- se refiere a cuadros de texto en los cuales no pueden perder el foco hasta que la ventana se halla cerrado. Generalmente se utilizan cuando el sistema necesita asegurarse que se ha leído un mensaje (por ejemplo error fatal).

Daemon.- es un programa asociado a los sistemas operativos UNIX que se ejecuta en segundo plano y realiza tareas sin que el usuario tenga que intervenir. Su equivalente en los sistemas Microsoft es un servicio .NET.

Dataware.- aplicación de e-business que permite capturar, gestionar y compartir todos los activos de conocimiento de una organización incluyendo documentos y bases de datos corporativas

Distribución uniforme.- se le denomina así a cualquier función de probabilidad en la cual la probabilidad de que todos los elementos que la forman tengan la misma probabilidad de ser escogidos.

ECMA (European Computer Manufacturers Association).- es una asociación fundada en 1961 dedicada a la estandarización de sistemas de información. Desde 1994 ha pasado a denominarse ECMA Internacional.

Edit Plus.- es un editor de texto de 32 bits, de html y para lenguajes de programación para el ambiente Windows. Ofrece diversas características qu elo hacen superior a cualquier editor de texto plano, como resultado en colores, emparejamiento de paréntesis, etc.

FORTRAN (Formula Translation).- es un lenguaje de programación desarrollado en los años 50 y activamente utilizado desde entonces. Se utiliza principalmente en aplicaciones científicas y análisis numérico.

Goteo de información.- se presenta cuando un sitio web, revela información importante, en comentarios del programador o mensajes de errores, que ayudan al atacante a vulnerar el sistema.

Herencia.- es una de las características que mas se destaca de la programación orientada a objetos, gracias a esta, es posible especializar o extender la funcionalidad de una clase, derivando de ella nuevas clases.

IDE (Integrated Development Environment).- el entorno de desarrollo integrado recibe este nombre, porque es una aplicación que reúne desde un procesador de texto, hasta herramientas de depuración para un lenguaje de programación en particular, además de permitir una relación amigable con el compilador del mismo.

Instanciar.- creación de un miembro u objeto a través de una clase.

Internet.- Internet es una red de redes a escala mundial de millones de computadoras interconectadas con el conjunto de protocolos TCP/IP. También se usa este nombre como sustantivo común y por tanto en minúsculas para designar a cualquier red de redes que use las mismas tecnologías que Internet, independientemente de su extensión o de que sea pública o privada.

Intérprete.- acepta programas escritos en un lenguaje de alto nivel, los analiza y los ejecuta bajo control del propio intérprete. En este caso, no se genera un programa equivalente en otro lenguaje, como ocurre con un compilador por lo que, si se desea repetir la ejecución del programa, es preciso volver a traducirlo.

IP.- una dirección IP es un número que identifica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP.

IRC.- Internet Relay Chat, es el sistema de comunicación por teclado que reúne diariamente a millones de personas de todo el mundo en internet. Llamado simplemente Chat, éste sistema de comunicación permite enviar archivos mientras se "chatea" Los programas más conocidos son Pirch, Microsoft Comic Chat y mIRC32.

Java.- es una plataforma de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales.

Lenguaje de cuarta generación.- Son lenguajes que se relacionan menos con procedimientos y que son aun mas parecidos al ingles que los lenguajes de tercera generación. Algunas características incluyen capacidades de consulta y base de datos, de creación de códigos y capacidades gráficas. Primera generación (lenguajes máquina); Segunda generación (lenguaje ensamblador); tercera generación (transición entre segunda y tercera generación).

Máquina Virtual.- emula un sistema de hardware completo, desde el procesador a la tarjeta de red, en un entorno de software aislado e independiente, que permite el funcionamiento simultáneo de sistemas operativos que, de otro modo, serían

incompatibles. Cada sistema operativo se ejecuta en su propia partición de software aislada.

MDI (Multiple Document Interface).- Forma en la cual se arreglan dentro de una ventana madre, varias ventanas hijas (mdiChild) que permiten trabajar con varias ventanas sobre un mismo espacio de trabajo.

Metadatos.- conjunto de información (datos) que acompañan a los auténticos datos informando de aspectos relacionados.

Migración.- en computación se le denomina a actualizar un programa, aplicación o sistema de un lenguaje de programación a otro.

mIRC.- ver IRC.

Motor de búsqueda.- programa o aplicación que se encarga de buscar en Internet sitios que correspondan a los deseados por el usuario (el más famoso es Goodle).

Objeto.- componente o código de software que contiene en sí mismo tanto sus características (campos) como sus comportamientos (métodos). Instancia de una clase.

Open BSD (Berkeley Software Distribution).- es un sistema operativo libre tipo Unix, multiplataforma. Es un descendiente de NetBSD, centrado en seguridad y criptografía. Este sistema operativo, se concentra en la portabilidad, cumplimiento de normas y regulaciones, corrección, seguridad proactiva y criptografía integrada. OpenBSD incluye emulación de binarios para la mayoría de los programas de los sistemas SVR4 (Solaris),FreeBSD, GNU/Linux, BSD/OS, SunOS y HP-UX.

Outsourcing.- se le llama así al proceso por el cual una determinada empresa contrata los servicios de otra para que desarrollen un producto solicitado. Para ello, pueden contratar sólo al personal o contratar tanto el personal como los recursos.

Panacea.- es un mítico medicamento capaz de curar todas las enfermedades, o incluso de prolongar indefinidamente la vida. Fue buscada durante siglos, especialmente en la Edad Media.

Paradigmas.- es una forma de representar y manipular el conocimiento. Representan un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

Perl (Practical Extraction and Report Language).- es un lenguaje de programación desarrollado por Larry Wall y optimizado para el escaneo de texto arbitrario de archivos, para tareas de administración de sistemas basado en scripts portable a casi cualquier plataforma. Es muy utilizado para escribir CGIs. Uno de sus elementos mas potentes son las expresiones regulares.

Pirch.- ver IRC.

Polimorfismo.- se denomina a la capacidad del código de un programa para ser utilizado con diferentes tipos de datos u objetos. También se puede aplicar a la propiedad que poseen algunas operaciones de tener un comportamiento diferente dependiendo del objeto (o tipo de dato) sobre el que se aplican.

Pretty Good Privacy.- una técnica muy popular de encriptación de correos electrónicos. Desarrollado por Phil Zimmerman para asegurar la confidencialidad y la integridad de los mensajes de correo electrónicos y el almacenamiento seguro de archivos.

Programación Orientada a Objetos.- es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas.

Router.- dispositivo utilizado para conectar o segmentar redes. Son comúnmente utilizados en redes que utilizan comunicación TCP/IP. Los routers se utilizan para conectar pequeñas redes a redes más grandes o viceversa.

Scripts.- también conocidos como lenguajes interpretados. A diferencia de los lenguajes compilados, en los lenguajes interpretados el código no necesita ser preprocesado mediante un compilador, eso significa que el ordenador es capaz de ejecutar la sucesión de instrucciones dadas por el programador sin necesidad de leer y traducir todo el código.

Sniffer.- programa que se encarga de monitorear el tráfico (paquetes) de una red.

Sobrecarga de operadores.- en Programación Orientada a Objetos se le denomina así a una propiedad de los lenguajes para aceptar funciones con el mismo nombre, pero con diferentes parámetros.

SQL (Structured Query Language).- el Lenguaje de Consulta Estructurado es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Reúne características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

Suit.- conjunto de aplicaciones que conforman un grupo que trabaja entre sí (por ejemplo Office).

Switch.- es un dispositivo de interconexión de redes de ordenadores/computadoras que opera en la capa 2 (nivel de enlace de datos) del modelo OSI (Open Systems Interconection). Un switch interconecta dos o más segmentos de red, funcionando de manera similar a los puentes (bridges), pasando datos de una red a otra, de acuerdo con la dirección MAC de destino de los datagramas en la red.

UNICODE.- es una norma de codificación de caracteres. Su objetivo es asignar a cada posible carácter de cada posible lenguaje un número y nombre único, a diferencia de la mayor parte de los juegos ISO como el ISO-8859-1, que sólo definen los necesarios para un idioma o zona geográfica.

XML (eXtensible Markup Language).- desarrollado por el World Wide Web Consortium (W3C), es una versión simple de SGML. Su objetivo principal es conseguir una página web más semántica. Aunque una de las principales funciones con las que nace sería suceder al HTML, separando la estructura del contenido y permitiendo el desarrollo de vocabularios modulares, compatibles con cierta unidad y simplicidad del lenguaje, tiene otras aplicaciones entre las que destaca su uso como estándar para el intercambio de datos entre diversas aplicaciones o software con lenguajes privados como en el caso del SOAP. Al igual que el HTML, se basa en documentos de texto plano en los que se utilizan etiquetas para delimitar los elementos de un documento. Sin embargo, XML define estas etiquetas en función del tipo de datos que está describiendo y no de la apariencia final que tendrán en pantalla o en la copia impresa, además de permitir definir nuevas etiquetas y ampliar las existentes.

XOR.- operación lógica binaria, donde cuando a o b son falsos, la respuesta es verdadera.

Bibliografía y Mesografía

Capítulo 1 – Introducción a la Plataforma .NET

- Munro, Jay; “The Microsoft .NET Development Environment“ http://www.extremetech.com/article2/0,3973,9852,00.asp
- Microsoft Press Pass; “Microsoft Outlines Vision for Future of Visual Studio .NET And Microsoft .NET Framework http://www.microsoft.com/presspass/press/2003/Jul03/07-29InnovationListPR.mspx
- Ryan, Vincent; “What Next for .NET?” http://www.newsfactor.com/story.xhtml?story_id=21344
- Microsoft Press Pass; “Microsoft Launches XML Web Services Revolution With Visual Studio .NET and .NET Framework” http://www.microsoft.com/presspass/press/2002/feb02/02-13RevolutionPR.mspx
- Microsoft Press Pass; “Microsoft Releases New Versions of Windows Server, Visual Studio .NET and SQL Server” http://www.microsoft.com/presspass/features/2003/apr03/04-24trio.mspx
- Murphy, Craig; “TechEd 2005 - What’s new in .NET 2.0” http://www.craigmurphy.com/blog/?p=98
- Krill, Paul; “Microsoft makes changes to Visual Studio 2005” http://www.infoworld.com/article/05/08/29/HNvschanges_1.html
- Platt, David. “Introducing Microsoft .NET”. “Introduction”, 3th Edition, Microsoft Press
- Platt, David. “Introducing Microsoft .NET”. “.NET Objects”, 3th Edition, Microsoft Press
- Platt, David. “Introducing Microsoft .NET”. “Data Access in .NET”, 3th Edition, Microsoft Press
- Platt, David. “Introducing Microsoft .NET”. “Threads”, 3th Edition, Microsoft Press
- Balena, Francesco, “Programming Ms Visual BASIC.NET”, “Introducing Microsoft .NET”; P. 2-24 2nd Edition Ed. Microsoft Press 2003

Bibliografía y Mesografía

Capítulo 2 – Visual BASIC.NET

- Marconi, Andrea M.; Historia de KBASIC (documentación de kBASIC)
  http://www.linuxfocus.org/Castellano/January2003/article277.shtml
- "QBASIC – Art History Online Referente Guide"
  http://www.arthistoryclub.com/art_history/QBASIC
- Hudson, Daniel P.; "A Brief History of the Development of BASIC"
  http://www.fys.ruu.nl/~bergmann/history.html
- "Historia de Visual BASIC"
  http://personal.telefonica.terra.es/web/oscarmartinez/_articlescas/article1.htm
- "Microsoft's Visual BASIC"
  http://www.microsoft.com/vBASIC
- Mack, George; "The History of Visual BASIC and BASIC on the PC"; 4th edition revision 1, copyright January 2002, May 2004
  http://dc37.dawsoncollege.qc.ca/compsci/gmack/info/VBHistory.htm
- "Microsoft Developer's Network"
  http://msdn.microsoft.com
- Balena, Francesco, "Programming Ms Visual BASIC.NET", "The Basics"; P. 2-97 2nd Edition Ed. Microsoft Press 2003

Bibliografía y Mesografía

Capítulo 3.- Fundamentos de Seguridad Informática

- Sealed Media; “Using SealedMedia to Achieve ISO 17799 compliance” http://www.sealedmedia.com/products/ISO17799.asp
- International Organization for Standardization http://www.iso.org
- Common Criteria Portal http://www.commoncriteriaportal.org/
- Wikipedia; “TCSEC” http://en.wikipedia.org/wiki/TCSEC
- Virus Prot; “HISTORIA DE LOS VIRUS” http://www.virusprot.com/Historia1.html
- Astorga, “Criptografía” http://rinconquevedo.iespana.es/rinconquevedo/criptografia/introduccion.htm
- Red Segura; “Criptografía – Historia” http://www.redsegura.com/Temas/CRhistoria.html
- Ruth, Andy and Hudson. Kart, “Security+, Certification”, “General Networking and Security Concepts”, 1st Edition, Microsoft Press
- Tulloch, Mitch. “Microsoft Encyclopedia of Security”. 1st Edition, Microsoft Press
- Stallings, William “Cryptography and Network Security”, “Classical Encryption Techniques”; P. 24-29 4th Edition, Ed. Prentice Hall
- Stallings, William “Cryptography and Network Security”, “Principles of Public Key Cryptosystems”; P. 259-267 4th Edition, Ed. Prentice Hall
- Tanenbaum, Andrew S. “Computer Networks”, “Network Security”; P. 721-833 4th Edition, Ed. Prentice Hall PTR

Bibliografía y Mesografía

Capítulo 4.- Software Seguro

- Ghezzi, Carlo; et al, “Fundamentals of Software Engineering”, “Software Engineering: A Preview”; P. 1-14 1st Edition, Ed. Prentice Hall 1991
- Ghezzi, Carlo; et al, “Fundamentals of Software Engineering”, “Software: It’s Nature and Qualities”; P. 17-41 1st Edition, Ed. Prentice Hall 1991
- Ghezzi, Carlo; et al, “Fundamentals of Software Engineering”, “Software Design”; P. 1-14 1st Edition, Ed. Prentice Hall 1991
- Bently, Jon, “Programming Pearls”, “Preliminaries”; P. 1-58 2nd Edition, Ed. Addison Wesley 2003
- Bently, Jon, “Programming Pearls”, “Performance”; P. 59-112 2nd Edition, Ed. Addison Wesley 2003
- Kernighan, Brian W., Pike, Rob, “The Practice of Programming”, “Style”; P. 1-27 Ed. Addison Wesley 2004
- Kernighan, Brian W., Pike, Rob, “The Practice of Programming”, “Design and Implementation”; P. 61-84 Ed. Addison Wesley 2004
- McConnell, Steve, “Code Complete”, “Laying the Foundation”, P. 3-72 2th Edition. Ed. Microsoft Press 2004
- McConnell, Steve, “Code Complete”, “Creating High-Quality Code”, P. 73-236 2th Edition. Ed. Microsoft Press 2004
- McConnell, Steve, “Code Complete”, “Code Improvements”, P. 463-648 2th Edition. Ed. Microsoft Press 2004
- Tlloch, Mitch. “Microsoft Encyclopedia of Security”. 1st Edition, Microsoft Press
- Tanenbaum, Andrew S., “Modern Operating Systems” ”, “Security”, P. 583-664 2th Edition. Ed. Prentice Hall 2001
- Kernighan, Brian W.y Dennis, Ritchie M., “El lenguaje de Programación C”, “Tipos, Operadores y Expresiones”, P. 39-59 2da Edición. Ed. Pearson Educación 2001
- Howard, Michael and LeBlanc, David, “Writing Secure Code”, “Contemporary Security”; P. 3-125 2th Edition Ed. Microsoft Press 2003
- Howard, Michael and LeBlanc, David, “Writing Secure Code”, “Secure Coding Techniques”; P. 127-412 2th Edition Ed. Microsoft Press 2003
- Howard, Michael and LeBlanc, David, “Writing Secure Code”, “Security Testing”; P. 567-614 2th Edition Ed. Microsoft Press 2003
- Foxall, James, “Practical Standards for Microsoft Visual BASIC.NET”; “Design”, P. 3-34, 1st Edition Ed. Microsoft Press 2003
- Foxall, James, “Practical Standards for Microsoft Visual BASIC.NET”; “Conventions”, P. 35-88, 1st Edition Ed. Microsoft Press 2003

- Foxall, James, “Practical Standards for Microsoft Visual BASIC.NET”; “Coding Constructs”, P. 89-202, 1st Edition Ed. Microsoft Press 2003

Bibliografía y Mesografía

Capítulo 5 – Seguridad en Visual BASIC.NET

- Foxall, James, “Practical Standards for Microsoft Visual BASIC.NET”; “Conventions”, P. 35-88, 1st Edition Ed. Microsoft Press 2003
- Howard, Michael and LeBlanc, David, “Writing Secure Code”, “Secure Coding Techniques”; P. 127-412 2th Edition Ed. Microsoft Press 2003
- Bond, James and Robinson, Ed, “Security for MS Visual BASIC.NET”, “Encryption”; P. 3-26 1st Edition Ed. Microsoft Press 2003
- Bond, James and Robinson, Ed, “Security for MS Visual BASIC.NET”, “Ensuring Hack Resistant Code”; P. 121-224 1st Edition Ed. Microsoft Press 2003
- Balena, Francesco, “Programming MS Visual BASIC.NET”, “Regular Expresions”; P. 307-325 2nd Edition Ed. Microsoft Press 2003

Bibliografía y Mesografía

Glosario

- Wikipedia
  http://en.wikipedia.org/wiki/TCSEC
- Tulloch, Mitch. “Microsoft Encyclopedia of Security”. 1st Edition, Microsoft Press
- Tulloch, Mitch and Tulloch Ingrid. “Microsoft Encyclopedia of networking”. 2nd Edition, Microsoft Press