Tema 2. Identificación de ataques y técnicas de intrusión

2.1 Introducción

Como se ha visto con anterioridad, en años recientes con el auge de las telecomunicaciones, la información se ha convertido en un bien de extraordinario valor para muchas personas, y su resguardo ha trascendido del ambiente puramente físico, al electrónico. Los métodos, tanto de uno como de otro bando, son cada vez más sofisticados y complejos, una guerra silenciosa, que cada día, a cada hora, se torna más grande y ardua, se libra sin que se alcance a distinguir todavía un final a la misma.

Dadas las condiciones actuales en que la red mundial se encuentra, es imprescindible tener presente la mayor cantidad de información referente a la seguridad informática sin importar si se debe proteger una PC casera o una red corporativa. A la mayoría de los piratas informáticos realmente no les interesa de qué tipo se trate; si alguien está comunicado hacia cualquier tipo de red, es una víctima potencial de sus ataques.

Es por ello, que a continuación ahondamos en los significados de vulnerabilidad y amenaza y veremos someramente algunas de las técnicas más comunes para realizar sus ataques, recurrentes, pero lo realmente desconcertante de todo esto, es que la mayoría de las veces, efectivas, y es que en la mayoría de la sociedad la cultura de seguridad informática no existe, o bien, no se ha desarrollado de manera adecuada.

2.2 Identificación de vulnerabilidades

Como se ha visto anteriormente, una vulnerabilidad "*es la potencialidad o posibilidad de ocurrencia de la materialización de una amenaza sobre un activo*". [43] (Véase figura 13.)

Figura 13. Varias amenazas acechan en Internet. [44]

Para estimar la vulnerabilidad de determinado activo o grupo, es imperativa la participación del responsable y por lo tanto, conocedor de cada activo, además, es necesario que esté suficientemente informado por algún especialista para que comprenda o imagine objetivamente, el daño potencial causado por la acción directa de una amenaza.

Se pueden considerar tres tipos de vulnerabilidades:

[43] Toni Puig, "Gestión de riesgos de los sistemas de información" en http://www.mailxmail.com/curso-gestion-riesgos-sistemas-informacion/identificacion-vulnerabilidades-impactos, 12/02/2010.
[44] Tomada de http://www.dosbit.com/tag/gusano

- Vulnerabilidad intrínseca, es aquella que sólo depende del activo y de la amenaza en sí.

- Vulnerabilidad efectiva, es la resultante luego de que se aplican las medidas correspondientes.

- Vulnerabilidad residual, es la resultante luego de que se aplican las medidas complementarias.

El grado de vulnerabilidad se mide considerando la relación entre la amenaza potencial y su riesgo a materializarse como una acción real sobre el objetivo. También se deberá calcular la frecuencia de ocurrencia a partir de hechos objetivos (estadísticas de incidentes por ejemplo). [45]

Estudios previos, han arrojado la siguiente tabla estadística que clasifican a la vulnerabilidad según la ocurrencia (Véase tabla 1):

Rango de frecuencias	Vulnerabilidad
Superior a 6 años	Muy baja
Menor a 6 años	Baja
Aproximadamente 1 año	Media
Menos de 2 meses	Alta
Menos de 1 semana	Muy alta

Tabla 1. Relación de vulnerabilidad con su tasa de ocurrencia. [46]

Las vulnerabilidades se encuentran en cualquier sistema operativo, llámense Windows, Mac OS, Unix, Linux, OpenBSD, por mencionar algunos, u aplicación. La única manera de reducir la posibilidad de que una vulnerabilidad pueda ser explotada, es permanecer vigilantes y aplicar mantenimiento frecuente al sistema, implementar una arquitectura de seguridad, controles de acceso, así como auditorías de seguridad.

2.2.1 Barrido de puertos

Esta técnica nos ayuda a detectar la situación en la que se encuentra cierto puerto, esto es, si éste se encuentra abierto, cerrado o detrás de un *firewall*. Su fin es revelar los servicios que ofrece el equipo en cuestión y las posibles vulnerabilidades dependiendo de los puertos que se encuentren abiertos. También, mediante el uso de esta técnica, es posible identificar el sistema operativo que está utilizando la máquina. Existe una multitud de programas que realizan el barrido de los puertos, siendo unos de los más conocidos Nmap. (Véase figura 14.)

[45] "Introducción a la seguridad informática" en http://es.kioskea.net/contents/secu/secuintro.php3, 12/09/2009.
[46] "Definición de Seguridad informática" en http://www.alegsa.com.ar/Dic/seguridad%20informatica.php, 12/09/2009.

Figura 14. Logo del sitio insecure.org, lugar de desarrollo de la aplicación Nmap, figura que con frecuencia se asocia al programa. [47]

Esta técnica, se divide a su vez, en las siguientes:

A. Escaneo de puertos TCP

Al establecer una conexión *TCP (Transmission Control Protocol)*, necesariamente se sigue una negociación consistente en tres pasos, regularmente conocida como *Three-way-handshake*. Este protocolo se inicia con la máquina origen que envía un paquete con la bandera SYN activada; la máquina destino responde a su vez con las banderas SYN/ACK prendidas y por último, la computadora origen envía un tercer y último paquete conteniendo la bandera *ACK*; una vez completados dichos pasos, la conexión entre los dos equipos se ha completado. (Véase figura 15.)

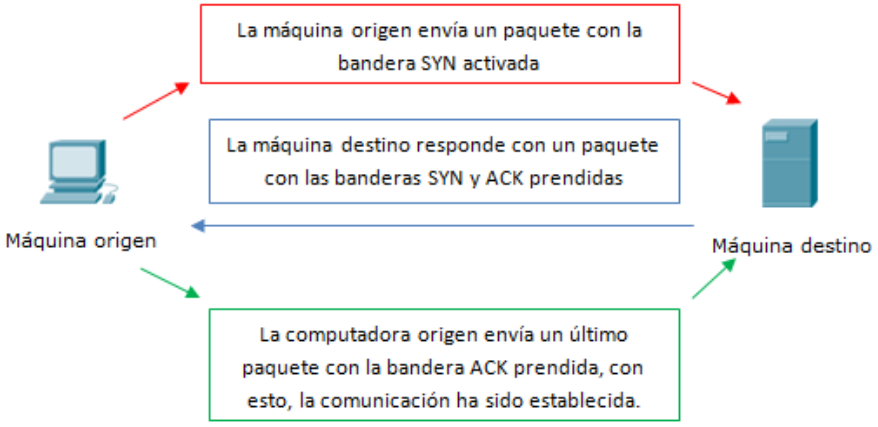

Figura 15. Representación gráfica de los pasos que se siguen durante el protocolo *Three-way-handshake*.

Un escáner de puertos *TCP* envía varios paquetes SYN al equipo que se está atacando y dependiendo de la respuesta que obtenga, interpreta lo siguiente:

- Si la respuesta es un SYN/ACK, entonces el puerto está abierto.
- Si la respuesta es un paquete RST, significará que el puerto está cerrado.

[47] Tomada de http://insecure.org/

- Por último, si lo que se obtiene es un paquete *ICMP (Internet Control Message Protocol)* como puerto inalcanzable, será porque dicho puerto está protegido por un *firewall*. [48]

Realizando dicho procedimiento en los puertos conocidos se tendrá una visión bastante completa del estado del equipo víctima.

B. Escaneo de puertos UDP

Aunque el protocolo *UDP (User Datagram Protocol)* es uno no orientado a conexión, si se manda un paquete a un puerto de estos y se encuentra cerrado, se obtendrá un mensaje de puerto inalcanzable, en cambio, si no obtienen respuesta, es posible inferir entonces que el puerto está abierto, aunque, si este está protegido por un *firewall*, entonces se obtendría información errónea.

Otra opción para descubrir puertos *UDP* es mandar paquetes de una aplicación específica con el fin de generar una respuesta de la capa de aplicación del modelo *OSI (Open System Intercconection)*. Por ejemplo, se podría mandar una consulta *DNS (Domain Name System)*.

Una forma de detectar un barrido de puertos es mediante la implementación de un *IDS (Intrusion Detection System)*, el cual, si detecta la firma de un barrido de puertos, lanzará una alarma y, en conjunción al *firewall*, se pondrá la IP del atacante en una lista negra que deniegue cualquier comunicación con la misma. [49]

2.2.2 Identificación de firewalls

Se podría definir a un *firewall* como un dispositivo informático, ya sea software o hardware que protege a una red privada al definir un perímetro de seguridad, y definírsele reglas de filtrado de paquetes.

La función principal del *firewall* es la de examinar paquetes en busca de coincidencias con las reglas que se le han fijado y dependiendo de ellas, permitirles o negarles el acceso. Además, los *firewalls* pueden también generar alarmas y crear listas negras. (Véase figura 16.)

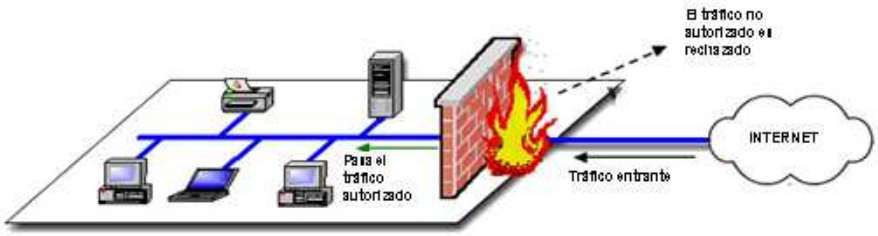

Figura 16. Forma de trabajo de un *firewall*, el paso de los paquetes es permitido o denegado según las reglas que se le hayan fijado. [50]

[48] Jayant Gadge, Anish Patil, "Port Scan Detection", en http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4772622, 30/04/2011.
[49] *Ídem*.
[50] Tomada de http://tindes.com/jj/index.php?option=com_content&view=article&id=41:seguridad-red&catid=1:actualidad&Itemid=27&0de28b3435550b272401162583c2c73f=tqyhgghphlnuox

Existen diferentes tipos de *firewall*:

- *Firewall* de filtrado por paquetes.
- *Firewall* de filtrado por estado.
- *Firewall* de filtrado por contenido o aplicación. [51]

A. Firewall de filtrado por paquetes

El *firewall* de filtrado por paquetes trabaja en la capa tres del modelo *OSI*, examina el encabezado de cada paquete analizando la procedencia y destino de los datos y en vista de esto decide si permite o no su paso.

Las reglas pueden basarse en:

- La dirección IP de origen o un intervalo de dirección IP.
- La dirección o direcciones IP de destino.
- El protocolo de red que usa.
- El número de puerto.
- El puerto de origen o destino. [52]

B. Firewall de filtrado por estado

Este permite guardar un registro de las conexiones existentes. Además, es capaz de trabajar con los protocolos tales como *IP*, *TCP*, *UDP*, *ICMP*, *FTP* e *IRC*. No trabajan filtrando paquetes individuales, sino sesiones enteras, permitiendo una optimización del trabajo de filtrado. [53]

C. Firewall de filtrado por contenido

También conocido como de filtrado por aplicación, analiza la trama a nivel de la capa de aplicación del modelo *OSI*, así, controla además de los puertos y sesiones, los protocolos que se utilizan para la comunicación evitando que se puedan suplantar servicios. Este tipo de *firewalls* implementa reglas más estrictas permitiendo un control mayor sobre las conexiones establecidas. Además, puede ser usado como servidor *proxy* para servicios tales como *HTTP (HyperText Transfer Protocol)*, *FTP*, entre otros.

Para la identificación de *firewalls* se utilizan diversas herramientas tales como el *firewalking*. El *firewalking* es una técnica que fue creada en 1998 con el objeto de descubrir las políticas de filtrado de un *firewall*. Lo que se hace básicamente es

[51] "Introducción a la seguridad informática" en http://es.kioskea.net/contents/secu/secuintro.php3, 12/09/2009.
[52] *Ídem*.
[53] "Definición de Seguridad informática" en http://www.alegsa.com.ar/Dic/seguridad%20informatica.php, 12/09/2009.

enviar un paquete con un *TTL (Time To Live)* fija, que debe expirar en el último *gateway*, antes de llegar al objetivo.

Cuando el paquete expira, y si se recibe un mensaje de error diciendo lo siguiente: exceed in-transit, significará que el paquete superó al *firewall*, pero no llegó al objetivo pues su *TTL* llegó a 0. Si no se recibe respuesta alguna, entonces el *firewall* habrá descartado al paquete puesto que no cumplía sus políticas de filtrado.

En la actualidad, esta técnica podría no funcionar, puesto que la mayoría de los *firewalls* no decrementan el *TTL* de los paquetes.

Una técnica popular para la obtención de información es el *banner grabbing*. Se le llama *banner* a la información que transmite un servicio cuando nos comunicamos con él, dicha información podría ser la versión, su nombre, etc. Así, mediante esta técnica es posible identificar los servicios que corren bajo un *firewall* y deducir, en consecuencia, cuáles son sus políticas de filtrado. [54]

2.2.3 Identificación del sistema operativo

Existen dos técnicas encaminadas a descubrir el sistema operativo que corre la máquina que está bajo ataque: la técnica activa y la pasiva. La activa opera bajo el principio de que cada sistema operativo responde diferente a una serie de paquetes mal formados. Lo que se hace es mandar a la máquina víctima estos paquetes y comparar su respuesta con una base de datos previamente hecha. Este método requiere de una conexión con la máquina víctima y su acción puede ser reconocida por un sistema *IDS*.

La identificación pasiva, por otra parte, captura paquetes provenientes del otro equipo con la ayuda de *sniffers*. Para determinar el sistema operativo del que se trata, esta técnica se basa en el principio de que todas las pilas *IP* se manejan de diferente manera para cada sistema operativo y así, analizando los paquetes capturados e identificando dichas diferencias, se podrá determinar el sistema operativo usado.

Existen áreas de los paquetes *TCP* capturados cuyo examen puede llevarnos a conocer el sistema operativo del que provienen, ejemplo de esto son:

- *TLL*. El número de saltos máximo que tiene un paquete antes de ser eliminado.
- Tamaño del *frame*.
- Si la bandera *DF (Don't Fragment)* está activada o no.

[54] "Firewalking" en http://www.webopedia.com/TERM/F/firewalking.html, 25/01/2010.

Analizando estos campos, es posible determinar el sistema operativo. Aunque no es 100% preciso, sí puede darnos un panorama bastante claro de lo que está corriendo en la máquina víctima.

Con esta técnica, puede evitarse el riesgo de ser detectado por un sistema *IDS*, además de que también pueden identificarse *firewalls* que trabajen como servidor *proxy*.

Esta técnica, como cualquiera, tiene ciertas limitaciones, como por ejemplo, el capturar paquetes de aplicaciones que construyen los suyos propios, un caso de esto es NMap. Y si se han modificado los valores del *TTL*, *DF*, entre otros, entonces no concordarán con ninguna base de datos, siendo esto una manera eficiente de protegerse contra dichos ataques. [55]

2.3 Explotación y obtención de acceso a sistemas y redes

La meta de todo atacante es el acceso al sistema víctima con los privilegios necesarios para la instalación y ejecución de software, o la modificación de archivos y su ubicación, así, hay diferentes técnicas que explotan las vulnerabilidades de aplicaciones, sistemas operativos, servicios, que nos permiten esto.

2.3.1 Robo de identidad

El robo de identidad o *phising* es básicamente un tipo de estafa en línea en el que se usan técnicas como el envío de correo no deseado (*spam*), sitios web falsos, mensajes instantáneos con el que pretenden engañar a los usuarios para que proporcionen información confidencial que puede ser desde datos de la tarjeta de crédito, cuentas bancarias, hasta contraseñas. (Véase figura 17.)

Existen varias recomendaciones para evitar ser víctima de este tipo de estafas:

- Si se recibe un e-mail o nos aparece un *pop-up* solicitando información personal o financiera, no responder ni hacer clic al enlace o vínculo del mensaje, puesto que, las compañías legítimas no solicitan este tipo de información vía e-mail.

- Por supuesto, utilizar antivirus y *firewalls* al navegar por la red.

- No enviar información personal o financiera a través del correo electrónico, puesto que no es un método seguro de transferencia de información.

- Revisar los resúmenes de las cuentas bancarias y tarjetas de crédito tan pronto como se reciban para verificar si se han imputado cargos no autorizados o no hechos.

[55] "Introducción a la seguridad informática" en http://es.kioskea.net/contents/secu/secuintro.php3, 12/09/2009.

- Tener cuidado al abrir o descargar documentos adjuntos a los mensajes recibidos, puesto que pueden contener *malware* que comprometan la seguridad de nuestro equipo. [56]

Figura 17. Relación de las organizaciones con presencia mundial, más utilizadas para realizar fraudes de *phising* en 2009. Diagrama de The Honey Pot Project. [57]

2.3.2 Engaño a firewalls e IDS's

La creación de "túneles" es una técnica popular utilizada para engañar la protección de un *firewall*. La técnica de *tunneling* consiste en encapsular un protocolo de red dentro de otro, permitiendo así superar los límites impuestos por algún *firewall.* (Véase figura 18.) Como ejemplo veamos dos aplicaciones que se comunican mediante el protocolo de transporte *TCP*. [58]

En el primer paso, las datos de la aplicación serán enviados al cliente "tunelizador" que será el encargado de encapsular estos datos dentro del protocolo *HTTP* (generalmente aceptado por la mayoría de los *firewalls*). La

[56] "¿Cómo evitar que te 'pesquen' en la red? Phising, un peligroso y moderno delito" en http://www.univision.com/content/content.jhtml?chid=9&schid=1860&secid=11068&cid=995274&pagenum=1, 10/10/2009.
[57] Tomada de http://www.pdatungsteno.com/2009/12/16/proyecto-honey-pot-colabora-en-la-lucha-mundial-contra-el-spam/
[58] "Secunia, stay secure" en http://secunia.com/, 13/10/2009.

aplicación y el cliente "tunelizador" no tienen que estar en la misma máquina, simplemente basta con que ésta sea accesible.

Después, el cliente "tunelizador" manda una petición *HTTP* estándar a un tercer agente, un "destunelizador" disfrazado como servidor WEB que tendrá el trabajo de desencapsular y mandar los datos a la aplicación destino.
Existen varias aplicaciones que permiten implementar esta técnica de evasión, por ejemplo: proxytunnel, HTTP-tunnel entre otros. para permitir crear un túnel usando el servicio de *SSH (Secure SHell)*, el cliente Putty puede servir. La herramienta NMap también implementa técnicas de evasión de *IDS's* y *firewalls*. [59]

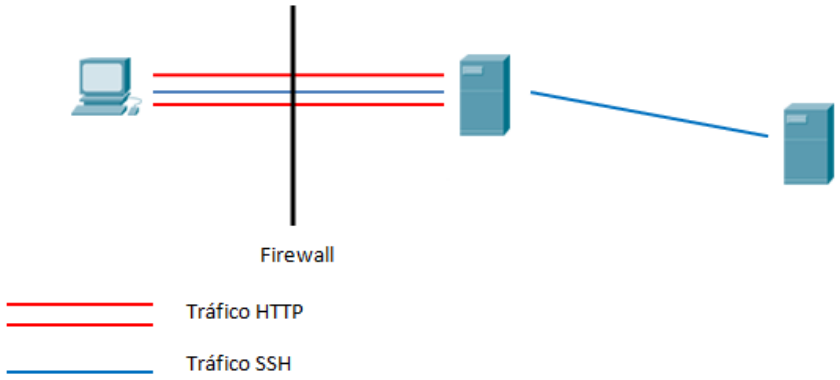

Figura 18. El tráfico *SSH* no permitido logra evadir al *firewall* encapsulado dentro de tráfico *HTTP* permitido.

2.3.3 Vulnerabilidades de software

Una vulnerabilidad de software puede definirse como un fallo o hueco de seguridad detectado en algún programa o sistema informático que puede ser utilizado bien por un virus para propagarse e infectar, o por *hackers* para entrar en los sistemas de manera no autorizada. De manera más sencilla, se trata de un fallo de diseño de algún programador que permite que un virus pueda realizar actividad maliciosa sin la necesidad de que el usuario actúe de forma directa. [60]

A continuación presentamos algunos casos de vulnerabilidad de software:

A. Buffer overflows

En un estudio generado por David Wagner, Jeffrey Foster, Eric Brewer y Alexander Aiken, mostraron que más del 50% de las vulnerabilidades explotadas eran de *buffer overflow*, también, el análisis demostraba que esta relación se incrementaba con el tiempo. [61]

[59] "Nmap – Free Security Scanner For Network Exploration & Security Audits" en http://nmap.org/, 23/10/2009.
[60] "Las vulnerabilidades de software" en http://www.pandasoftware.com/about/press/viewNews.aspx?noticia=4610, 27/09/2009.
[61] Aleph One, "Smashing the stack for fun and profit" en http://insecure.org/stf/smashstack.html, 29/09/2009.

Durante mucho tiempo se ha reconocido que los desbordamientos del *buffer* son un problema en lenguajes de bajo nivel. La esencia del problema es que los datos de usuario y la información del control de flujo del programa se mezclan en beneficio del desempeño, y los lenguajes de bajo nivel permiten el acceso directo a la memoria. C y C++ son los dos lenguajes más populares afectados por los desbordamientos del *buffer*.

Estrictamente hablando, ocurre un desbordamiento del *buffer* cuando un programa permite la escritura más allá del final del *buffer* asignado. Otra parte del problema ocurre cuando se permite que un atacante escriba en una ubicación arbitraria de memoria, fuera de la matriz de aplicación.

La pila

La pila es un tipo abstracto de datos. Una pila de objetos tiene la propiedad de que el último objeto posicionado en ella será el primero en ser removido. (Véase figura 19.)

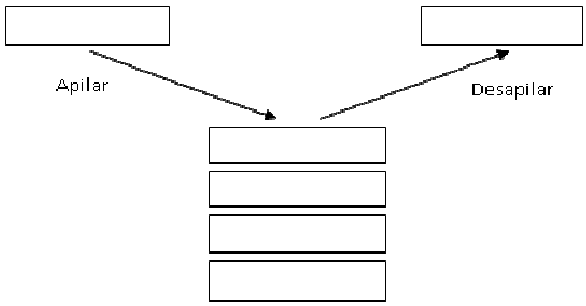

Figura 19. Representación de la organización de una pila.

Esta propiedad es comúnmente referida como último en entrar, primero en salir o LIFO en inglés (Last In, First Out queue).

Varias operaciones están definidas para la pila. Dos de las más importantes son PUSH y POP. La operación PUSH añade elementos al tope de la pila. Por el contrario, POP retira el último elemento del tope de la pila. [62]

Efectos de un desbordamiento de buffer

El efecto de un desbordamiento de *buffer* es amplio, desde la caída del sistema hasta la apropiación completa del control de la aplicación por parte del atacante; y si la aplicación se ejecuta como administrador o usuario raíz, entonces el atacante tendrá el control de todo el sistema operativo, así como de todos los usuarios que hayan iniciado sesión en el sistema. Si la aplicación desbordada es un servicio de red, el resultado del error podría ser un gusano. [63]

[62] An Zhiyuan, Liu Haiyan, "Realization of Buffer Overflow", 2010 International Forum on Information Technology an Applications en http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5635047, 30/04/2011.
[63] *Ídem.*

Explicación del desbordamiento del buffer

Al desbordamiento clásico del *buffer* se le conoce como "rompimiento de pila". En un programa compilado, la pila se utiliza para contener información de control como argumentos, a la que debe regresar la aplicación una vez que haya terminado con la función. Por desgracia, las variables que se asignan de manera local también se almacenan en la pila. Algunas veces, a estas variables se les identifica en forma imprecisa, como si se les asignara de manera estática, en lugar de la manera dinámica. La raíz del desbordamiento de *buffer* radica en que si la aplicación escribe más allá de los límites de una matriz asignada por la pila, el atacante tiene la posibilidad de especificar información de control. Y, esto es crítico para lograr su objetivo: el atacante busca la modificación de la información de control por valores que él mismo maneja.

Otra forma de *buffer overflow* que podría representar problemas de seguridad mucho más graves es el llamado *stack-smashing attack*, en el que un usuario malicioso puede tomar ventaja de un *buffer overflow* a través de la ejecución de código arbitrario. [64]

Métodos de prevención

- Reemplazar funciones peligrosas de manejo de cadenas.
- Reemplazar funciones inseguras como strcpy, strcat con las versiones válidas de cada una de estas funciones.

Reemplazar buffers de cadena de C con cadenas de C++

- Esto es más eficaz que sólo reemplazar las llamadas de C, pero es posible que cause gran cantidad de cambios en código existente, en particular si el código no está compilado como C++. [65]

B. Heap overflow

El *heap overflow* es un tipo de *buffer overflow* que se produce en un área de memoria denominada *heap*. La memoria *heap* se reserva dinámicamente con funciones tales como malloc() (subrutina para el ejercicio de asignación de memoria dinámica en los lenguajes de programación C y C++, es la abreviatura del inglés Memory Allocation) y puede ser sobreescrita de forma similar a la que se produce en los *buffer overflow*, es decir, en situaciones en las que el programador no verifica correctamente el tamaño de los datos que copia. [66]

Esta técnica es generalmente más difícil de explotar que el *buffer overflow*, por esta razón, es recomendable que los desarrolladores sigan como regla, no asignar de forma estática espacio para *buffer*.

---

[64] "Las vulnerabilidades de software" en http://www.pandasoftware.com/about/press/viewNews.aspx?noticia=4610 , 27/09/2009.
[65] *Ídem*.
[66] *Ídem*.

C. Vulnerabilidad de formato de cadena

Definición

La principal causa del error de formato de cadena es aceptar sin validar la entrada proporcionada por el usuario. En C es posible utilizar errores de formato de cadena para escribir en ubicaciones de memoria arbitrarias. El aspecto más peligroso es que esto llega a suceder sin la necesidad de manipular bloques de memoria adyacentes. Esta capacidad de diseminación permite a un atacante eludir protecciones de pila, e incluso modificar partes muy pequeñas de memoria. El problema también llega a ocurrir cuando las cadenas se leen a partir de una ubicación no confiable que controla el atacante. [67]

Localización de problemas de formato de cadena

Cualquier aplicación que tome una entrada del usuario y la pase a función de formateo está en riesgo. Un ejemplo muy frecuente de este problema sucede junto con aplicaciones que registran la entrada de usuario. Además, algunas funciones tal vez implanten formateo de manera interna.

En C habrá que buscar funciones de la familia printf. Entre los problemas que habrá de localizar se encuentran los siguientes:

```
printf (entrada_usuario);
fprintf (STDOUT, entrada_usuario);
```

Si se encuentra con una función con el siguiente aspecto:

```
fprintf (STDOUT, msg_format, arg1, arg2);
```

Se necesitará verificar dónde se almacenan la cadena a la que hace referencia msg_formato y si está protegida.

Métodos de prevención

El primer paso es nunca pasar entradas de usuario directamente a una función de formateo, además de asegurarse de hacerlo en cada nivel de manipulación de salida formateada. Como nota adicional, las funciones de formato tienen una sobrecarga de trabajo importante. Quizá sea conveniente escribir:

```
fprintf (STDOUT, buf);
```

la anterior línea de código no solamente es peligrosa, sino que también consume mucho ciclos de *CPU (Central Processing Unit)* adicionales.

[67] *Ídem.*

El segundo paso que habrá de darse consiste en asegurar que las cadenas de formato que utiliza la aplicación lean desde lugares confiables y que las rutas a las cadenas no sean controladas por el atacante. [68]

D. Race condition

"La condición de carrera se presenta cuando dos o más procesos leen y escriben en un área compartida y el resultado final depende de los instantes de ejecución de cada uno. Cuando una situación de este tipo se produce, y acciones que deberían ser ejecutadas en un periodo de tiempo fijo (atómicas) no lo hacen, existe un intervalo de tiempo durante el que un atacante puede obtener ciertos privilegios, así como leer y escribir en archivos protegidos, y en definitiva, violar las políticas de seguridad del sistema". [69]

Como ejemplo, veamos el código simplificado de un programa que almacena información en un archivo perteneciente a un usuario con privilegios de *root* para un sistema operativo Unix.

Ejemplo 1.

```
if (access ( archivo, W_OK) == 0) {
        open ();
        write ();
}
```

En una ejecución normal, si el usuario no tiene privilegios suficientes para escribir en el archivo, la llamada a acces () devolverá un -1, no permitiendo ni la apertura ni la escritura de archivo. Pero, ¿qué es lo que sucedería si el archivo cambiara entre la llamada a access()?. Imaginemos que después de la llamada a access(), y justo antes de que se ejecute la llamada a la función open(), el usuario borra el archivo original y lo enlaza por ejemplo a la dirección /etc/passwd: el programa estará escribiendo información en el archivo de contraseñas de los sistemas Linux.

En esta clase de situaciones, donde un programa comprueba la situación de un objeto, y luego ejecuta cierta acción asumiendo que la situación se mantiene cuando en realidad no es así, se denomina TOCTTOU (Time Of Check To Time Of Use). [70]

Qué hacer para evitar esto

Se pueden usar descriptores de archivo en lugar de nombres; para el ejemplo anterior, deberíamos usar una variante a la llamada access() que trabaje con descriptores en lugar de nombres de archivo. Con esto, se consigue que aunque se modifique el nombre del archivo, el objeto al que accedemos sea el mismo

[68] *Ídem.*
[69] Tanenbaum, Andrew. "Distributed Operating Systems" ED. Prentice Hall, 1995.
[70] Antonio Villalón, "Condiciones de carrera", *Seguridad en Unix y redes* en http://es.tldp.org/Manuales-LuCAS/SEGUNIX/unixsec-2.1.pdf, 30/09/2009.

siempre. Además, es conveniente invertir el orden de las llamadas, es decir, invocar primero a open() y después a la variante de access().

E. SQL injection

Se trata de una vulnerabilidad a nivel de la validación a las entradas a la base de datos de una aplicación. Es, en otras palabras, la posibilidad de inyectar sentencias *SQL (Structured Query Languaje)* arbitrarias en, por ejemplo, formularios estándar publicados en un sitio web. [71]

Para iniciar, demos un repaso a los comandos comunes a todas las distribuciones de *SQL*. (Véanse tablas 2, 3 y 4.)

Comandos DCL (Data Control Languaje)

Grant	Utilizado para otorgar permisos
Revoke	Utilizado para revocar permisos
Deny	Utilizado para denegar el acceso

Tabla 2. Comandos DCL, permiten al administrador controlar el acceso a los datos contenidos en una base de datos. [72]

Comandos DDL (Data Definition Languaje Statements)

Create	Utilizado para crear tablas, campos e índices
Drop	Empleado para eliminar tablas e índices
Alter	Utilizado para modificar tablas agregando campos o cambiando su definición.

Tabla 3. Comandos DDL, permiten a los usuarios llevar a cabo las tareas de definición de las estructuras que almacenarán los datos, así como los procedimientos o funciones que permitan consultarlos. [73]

Comandos DML (Data Manipulation Languaje Statements)

Select	Utilizado para conmutar registros de la base de datos para satisfacer un criterio determinado.
Insert	Utilizado para cargar lotes de datos a la base de datos en un única operación
Update	Utilizado para modificar los valores de los campos y registros especificados.
Delete	Utilizado para eliminar registros de una tabla de la base de datos.

Tabla 4. Comandos DML, permiten al usuario llevar tareas de consulta o manipulación de datos. [74]

[71] "Definición de Seguridad informática" en http://www.alegsa.com.ar/Dic/seguridad%20informatica.php, 12/09/2009.
[72] *Ídem.*
[73] *Ídem.*
[74] *Ídem.*

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que se desean seleccionar o manipular. (Véase tabla 5.)

Cláusulas

From	Es utilizada para especificar la tabla de la cual se van a seleccionar los registros
Where	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
Gruoup by	Utilizada para separar los registros seleccionados en grupos específicos.
Having	Utilizada para expresar la condición que debe satisfacer cada grupo
Order by	Utilizada para ordenar los registros seleccionados de acuerdo a un orden específico

Tabla 5. Relación de cláusulas y su uso. [75]

Operadores de comparación

Son operadores, en su mayoría binarios, que nos permiten comprar variables devolviendo un valor booleano a 1, si se cumple la condición y 0, en caso contrario. (Véase tabla 6.)

Operadores de comparación

<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
=	Igual que
Between	Para especificar un intervalo de valores
Like	Utilizado para comparar modelos
In	Utilizado para especificar registros en una base de datos.

Tabla 6. Los operadores de comparación verifican la relación entre dos variables. [76]

Veamos ahora, algunos ejemplos de sentencias *SQL*:

Ejemplo 1.

*SELECT * FROM Tabla;*

Lo que hace esta sentencia es devolver todos los registros que se hallen en la tabla llamada Tabla.

[75] *Ídem.*
[76] *Ídem.*

Ejemplo 2.

UPDATE Tabla SET password = '12345' WHERE user = 'admin'

En esta ocasión, se fijará como contraseña, para el usuario admin el valor 12345.

Ahora bien, es importante destacar la forma en que *SQL* ejecuta las instrucciones, puesto que es un punto fundamental si se quieren construir sentencias *SQL* a inyectar. Independientemente de la complejidad de la sentencia, ésta siempre se ejecutará por partes, es decir, secuencialmente una detrás de otra.

El aumento de páginas que requieran autenticación por parte del usuario, llenado de formularios, encuestas, suscripciones, por mencionar algunas, no hace más que ampliar extraordinariamente el espectro y potencial peligrosidad de esta clase de ataques, debido a que, en prácticamente el 100% de los casos, todos los argumentos recogidos serán pasados como una consulta a una base de datos, la cual será la encargada de mostrar al usuario los resultados de la misma.

A base de ingresar sentencias *SQL*, en el campo de autenticación de alguna página web, es que podemos terminar accediendo a la base de datos de la misma, con diferentes privilegios.

Un ejemplo simple de este tipo de ataque sería el siguiente:

En la parte donde se pide teclear el usuario de alguna página que requiera autenticación, podríamos introducir:

Usuario: '; drop table usuario- -

Con un poco de suerte, es decir, con los privilegios necesarios, la tabla "usuario" sería borrada, lo que traería como consecuencia que a partir de entonces, ningún usuario pueda autenticarse. Un ejemplo simple de ataque *DoS* usando *SQL* injection.

La comilla simple es interpretada por el manejador de bases de datos como un terminados de caracteres, lo que pasará es en el motor *SQL* interpretará que la cadena pasada como argumento empieza después de la primer comilla simple y termina con la segunda comilla.

Para ser más claros, veamos otro ejemplo.

Usuario: Gus'tavo
Password: 1234

Lo que sucederá es que el manejador interpretará los datos que se le han introducido como:

username: 'Gus'
password: 1234

e intentará ejecutar el resto de la consulta, es decir, el 'tavo', lo que dará por resultado, un error, puesto que no significa nada para *SQL*.

En nuestro ejemplo anterior, después de la comilla simple vino la sentencia drop table usuario, seguido de un doble guión: - -, esta sentencia le indica a *SQL* que lo que viene después, es un comentario, así, que lo ignorará y posiblemente evitaremos un error de sintaxis.

Así, con estos simples ejemplos es que podríamos ganar acceso a una base de datos mal protegida. Por supuesto, no son estas las únicas posibles sentencias inyectables, existen un sinfín más, incluso, se podría combinar con otras técnicas de ataque, como por ejemplo, el uso de *sniffers*.

Aunque en la mayoría de los lenguajes de programación, se pueden implementar medidas de seguridad contra este tipo de ataques, la realidad es que muchos de los desarrolladores no se preocupan por implementar seguridad en sus sistemas, resultando en pérdidas graves de información y activos.

2.4 Ataques a contraseñas

Hoy en día, el ataque a contraseñas es el ataque más recurrente puesto que no se requiere de grandes conocimientos técnicos para llevarlo a cabo. Además, la mayoría de los usuarios no suelen recurrir a contraseñas robustas, esto es, que incluyan letras minúsculas y mayúsculas, números y símbolos, sino que es muy común encontrarse como contraseña un nombre, una fecha significativa, una palabra sacada del diccionario, etc. y también, es común encontrar la misma contraseña, para varios pedidos de acceso, todo lo cual, simplifica en gran medida el trabajo al atacante.

Existen diversas prácticas recomendadas para elegir una buena y robusta contraseña, como las siguientes:

- Contraseñas largas.
- Mezclar mayúsculas y minúsculas.
- Incluir caracteres especiales, como por ejemplo, $, -, #, %,.
- Mezclar letras y números.
- Utilizar reglas nemotécnicas:
  i. Pensar en una frase: Este es mi proyecto de tesis.
  ii. Tomar las iniciales: Esmpdt.
  iii. Añadir complejidad: Esmpdt -> 3$mpd7

iv. Mezclar mayúsculas y minúsculas: 3$mPD7 [77]

2.4.1 Ataques por fuerza bruta y diccionario

Se denomina ataque de fuerza bruta a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar la correcta. Existe una variedad de herramientas para todos los sistemas operativos que permiten este tipo de técnica. [78]

Los ataques por fuerza bruta, dado que utilizan el método de prueba y error, son muy costosos en lo que se refiere a tiempo computacional. Este tiempo es directamente proporcional a la complejidad con que la contraseña haya sido construida.

Además de dicho ataques, existen los llamados ataques de diccionario, que se centra en la práctica no recomendada que siguen algunos usuarios de elegir contraseñas que tengan algún significado, así, los ataques por fuerza bruta prueban usando cualquier combinación posible, mientras que los ataques de diccionario prueban con palabras conocidas.

Adicionalmente a estos dos ataques, existe un tercero que utiliza las características de los dos anteriores y es conocido como ataque híbrido, este tipo de ataques apuntan a contraseñas compuestas por una palabra tradicional seguida de una letra o número, por ejemplo, si alguien usa la contraseña 'perro123'. [79]

2.4.2 Herramientas

Existe una multitud de herramientas desarrolladas para llevar a cabo este tipo de técnicas, pero aquí veremos algunas de las más comúnmente utilizadas.

- Herramienta: Essential Net Tools.

  Contramedidas: Complejidad de contraseñas, *firewalls*.

  Descripción: Aunque el conjunto de herramientas de Essential Net Tools está completamente dirigido a realizar diagnósticos de redes y monitorización de conexiones, también pueden ser usadas para llevar a cabo test de penetración mediante ataques de contraseñas. Esta aplicación incluye varias herramientas indispensables para la administración y monitorización de redes. Entre ellas:

[77] "Ataques a contraseñas" en http://serdis.dis.ulpgc.es/~a013775/asignaturas/iiaso/curso0708/trabajos/seguridad/crack/ataquecontrasena.pdf, 03/10/2009.
[78] "Contraseñas" en http://es.kioskea.net/contents/attaques/passwd.php3, 03/10/2009.
[79] Saikat Chakrabarti, Mukesh Singhal, "Password-Based Authentication: Preventing Dictionary Atacks", University of Kentucky en http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4249815, 30/04/2011.

- NetStat. Muestra la lista de conexiones de red de una computadora, información sobre puertos *TCP* y *UDP* abiertos, direcciones IP y estados de las conexiones. (Véase figura 20.)

- NBScan. Es un escáner de *NetBIOS*. Esta herramienta ofrece una *interfaz* de usuario gráfica y de fácil administración del archivo lmhosts y funciones de escaneo paralelo, lo que permite comprobar una red clase C en un minuto.

- PortScan. Es un escáner de *TCP* que permite buscar la red en busca de puertos activos.

- Shares. Monitoriza y crea *logs* de conexiones externas de recursos compartidos, listas de compartidos locales, también permite conectar a recursos remotos.

- LMHosts. Es un editor de los archivos lmhosts integrados con NBScan.

- SysFiles. Un editor de archivos de sistema: servicios, protocolo, redes, hosts y lmhosts.

- NetAudit (*NetBIOS* Auditing Tool). Permite mejorar la seguridad de comprobación de la red.

- RawSocket. Ofrece la capacidad de establecer conexiones de bajo nivel *TCP* y *UDP* para resolver y comprobar los diferentes servicios.

- TraceRoute y Ping. Estas utilidades se presentan de forma personalizable y con resultados convenientemente mostrados.

- NSLookup. Permite convertir direcciones IP a *hostnames* y viceversa, obtener los alias, y mejorar las consultas *DNS* avanzadas.

- ProcMon. Muestra una lista de los procesos ejecutándose. Otras funciones incluyen generación de reportes en *HTML*, texto y formatos delimitados por comas, posibilidad de compartir direcciones de *IP* rápidamente entre diferentes herramientas, y mucho más. [80]

[80] "Laboratorios: Hacking – Técnicas y contramedidas – Ataques por fuerza bruta (Brute Force) II" en http://labs.dragonjar.org/laboratorios-hacking-tecnicas-y-contramedidas-ataques-por-fuerza-bruta-brute-force-ii, 04/10/2009.

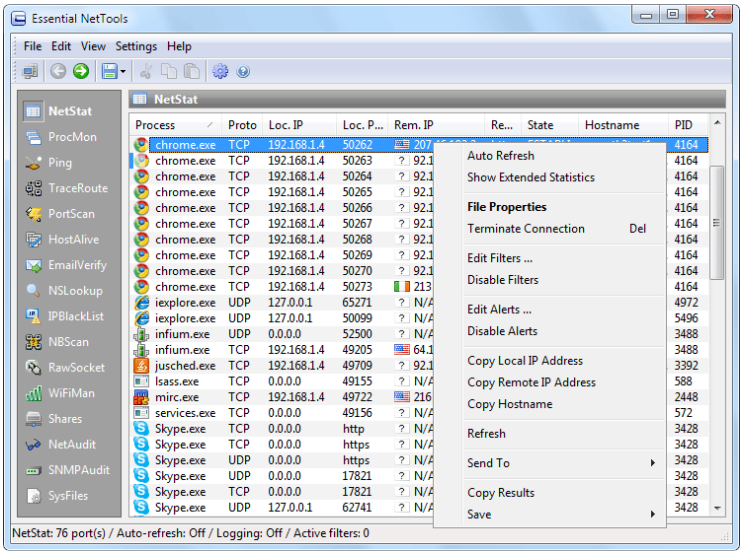

Figura 20. Imagen de la herramienta Essential NetTools, utilizando la herramienta NetStat. [81]

- Herramienta: John the Ripper.

  Contramedidas: Complejidad de contraseñas, políticas robustas de formación de contraseñas.

  Descripción: John the Ripper es un programa de criptografía que aplica fuerza bruta para descifrar contraseñas. Es una herramienta de seguridad muy popular, ya que permite a los administradores de sistemas comprobar que las contraseñas de los usuarios son suficientemente buenas. (Véase figura 21.)

  John the Ripper es capaz de auto detectar el tipo de cifrado de entre muchos disponibles, y se puede personalizar su algoritmo de prueba de contraseñas. Eso ha hecho que sea uno de los más usados en este campo.

  Características:

  - Optimizado para muchos modelos de procesador.
  - Funciona en muchas arquitecturas y sistemas operativos.
  - Ataques de diccionario y por fuerza bruta.
  - Muy personalizable (es software libre).
  - Permite definir el rango de letras que se usará para construir las palabras, y las longitudes.
  - Permite parar el proceso, y continuarlo más adelante.

[81] Tomada de http://www.tamos.com/htmlhelp/nettools/overview.htm

- Permite incluir reglas en el diccionario para decir cómo han de hacerse las variaciones tipográficas.

- Se puede automatizar.

John the Ripper usa un ataque por diccionario: tiene un diccionario con palabras que pueden ser contraseñas típicas, y las va probando todas. Para cada palabra, la cifra y la compara con el *hash* a descifrar. Si coinciden, entonces es la palabra correcta. [82]

Esto funciona bien debido a la poca cultura de seguridad de la mayoría de los usuarios. Además, este software también prueba con variaciones de estas palabras: les añade números, signos, mayúsculas y minúsculas, cambia letras, combina palabras, entre otras (ataque híbrido).

Además ofrece el típico sistema de fuerza bruta en el que se prueban todas las combinaciones posibles, sean palabras o no.

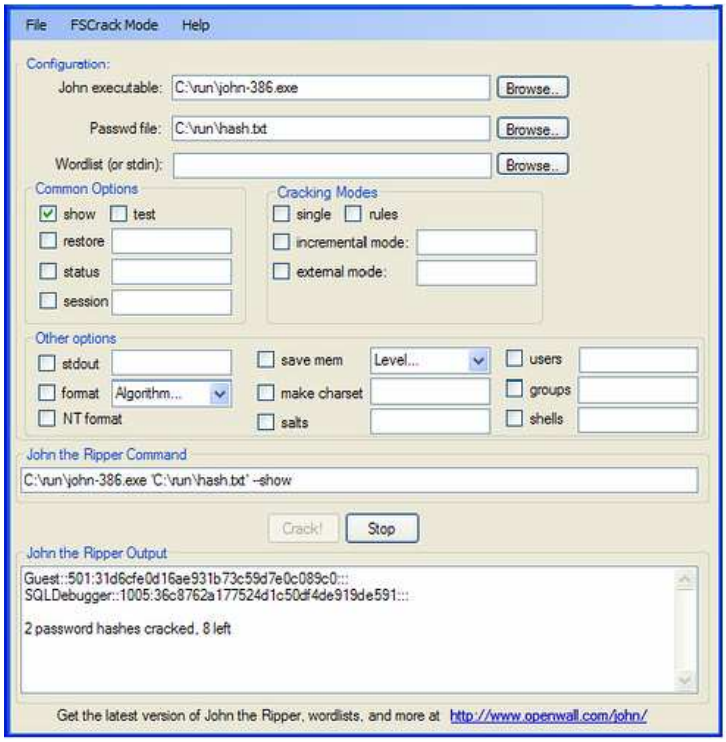

Figura 21. Pantalla principal de John the Ripper. [83]

- Herramienta: BrutusAET 2.

  Contramedidas: Complejidad de contraseñas.

[82] "Laboratorios: Hacking – Técnicas y contramedidas – Ataques por fuerza bruta (Brute Force) II" en http://labs.dragonjar.org/laboratorios-hacking-tecnicas-y-contramedidas-ataques-por-fuerza-bruta-brute-force-ii, 04/10/2009.

[83] Tomada de http://farm3.static.flickr.com/2311/2105464366_99b1b3d9a3.jpg

Descripción: Cuenta con una *interfaz* gráfica amigable, los protocolos soportados son, entre otros: *HTTP* (Autenticación básica), *HTTP* (HTML Form/CGI), *FTP*, *Telnet*, etc.

Características:

- Motor gradual de autenticación.
- Hasta 60 conexiones simultáneas por objetivo.
- Modos configurables de ataque por fuerza bruta.
- Secuencias altamente configurables de autenticación.
- Posibilidad de salvar la sesión de ataque para retomarla después.

BrutusAET2, al igual que John The Ripper, es capaz de utilizar ataques por diccionario, híbridos y de fuerza bruta. (Véase figura 22) [84]

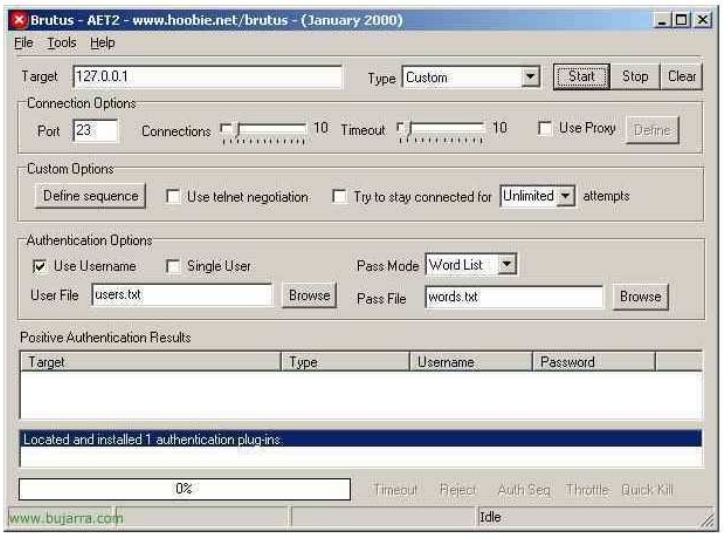

Figura 22. Pantalla principal de BrutusAET2 [85]

- Herramienta: THC-Hydra.

  Contramedidas: *Firewalls*, monitoreo de sesiones, complejidad de las contraseñas.

  Descripción: Dicha herramienta realiza diferentes ataques de fuerza bruta o diccionario a varios protocolos como son: *Telnet*, *HTTP*, *FTP*, entre muchos otros. [86] (Véase figura 23.)

[84] "Brutus" en http://www.bujarra.com/Brutus.html, 05/10/2009.
[85] Tomada de http://www.bujarra.com/Brutus.html
[86] Van Hauser, "THC Hydra" en http://freeworld.thc.org/thc-hydra/, 31/10/2009.

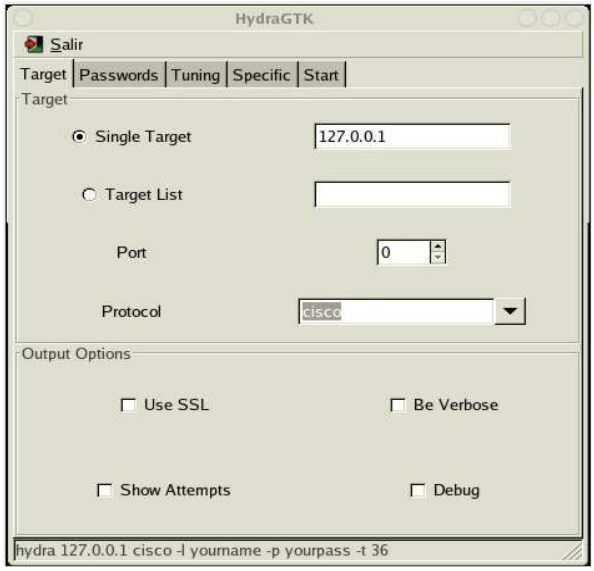

Figura 23. Pantalla de THC-Hydra [87]

2.5 Ataques a redes inalámbricas

Aunque las redes inalámbricas proveen gran facilidad de comunicación y ofrecen una gran movilidad entre los usuarios, representan también un riesgo de seguridad. Los atacantes pueden detectar y obtener acceso a redes corporativas aun cuando existen métodos para implementar seguridad, debido a que la mayoría de éstos son muy débiles y por tanto, fáciles de romper.

2.5.1 Denegación de servicio

Es un ataque en el que uno o más equipos intentan prevenir que una máquina objetivo pueda realizar un trabajo específico. La víctima puede ser un *servidor*, un *router*, un *hipervínculo*, una red completa, un usuario de internet, un proveedor de servicio de internet, un país o una combinación de varios de los casos anteriores. [88]

En el caso de las redes inalámbricas, este ataque puede implicar, entre otras cosas, la imposibilidad de un usuario para conectarse a una red o la pérdida de la conexión a ésta.

Un ejemplo de lo anterior es el llamado ataque de des-autenticación, en el cual el atacante envía una trama de des-autenticación al punto de acceso, provocando una des-autenticación del usuario en el mismo y, consecuentemente, la pérdida de la conexión. (Véase figura 24.)

[87] Tomada de http://www.mexicoextremo.com.mx/content/view/482/62
[88] Andrew Whitaker, Daniel Newman, *Penetration testing and network defense*, Estados Unidos, Cisco Press, 2006, p. 358.

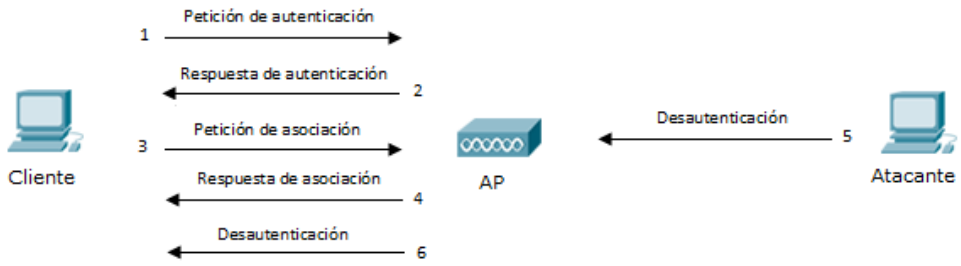

Figura 24. Ataque de des-autenticación.

Otro ataque similar es el *flooding* de autenticación, en donde se inunda el punto de acceso con peticiones que previenen que las genuinas sean atendidas.

Ataques de este tipo pueden ser detectados fácilmente utilizando herramientas inalámbricas *IDS* como Airdefense de Motorola o Airespace.

2.5.2 ARP poisoning

El protocolo *ARP (Address Resolution Protocol)* tiene como finalidad encontrar la dirección MAC que corresponde a una determinada dirección IP. Para ello se envía un paquete *ARP* request a la dirección *broadcast* de la red que contiene la dirección IP que se busca y, posteriormente, se espera a que dicha dirección IP responda con un *ARP* reply que incluye la dirección física que corresponde.[89] (Véase figura 25.)

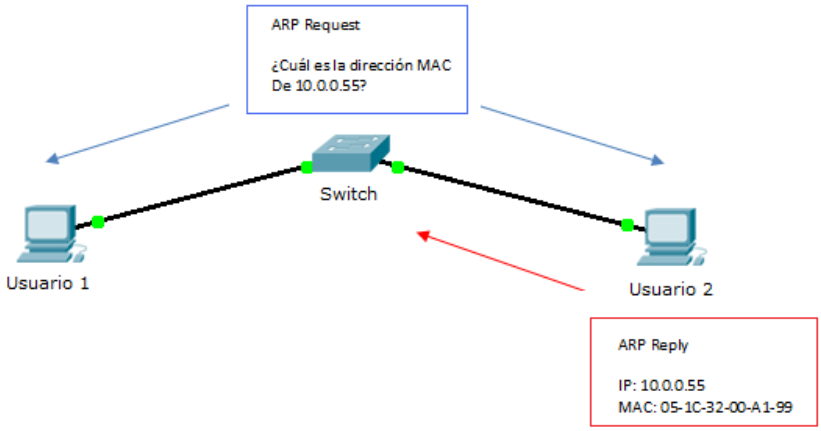

Figura 25. ARP reply.

El principio del *ARP* poisoning es enviar mensajes *ARP* falsos a la red. Normalmente la finalidad es asociar la dirección MAC del atacante con la dirección IP de otro

[89] *Idem*, p. 335, 336.

equipo (la víctima), como por ejemplo la puerta de enlace predeterminada. (Véase figura 26.) Cualquier tráfico dirigido a la dirección IP de esa máquina será erróneamente enviado al atacante, en lugar de a su destino real. El atacante puede entonces elegir entre reenviar el tráfico a la puerta de enlace predeterminada real (ataque pasivo) o modificar los datos antes de reenviarlos (ataque activo). El atacante puede incluso provocar una denegación de servicio contra una víctima, asociando una dirección MAC inexistente con la dirección IP de la puerta de enlace predeterminada de la víctima. [90]

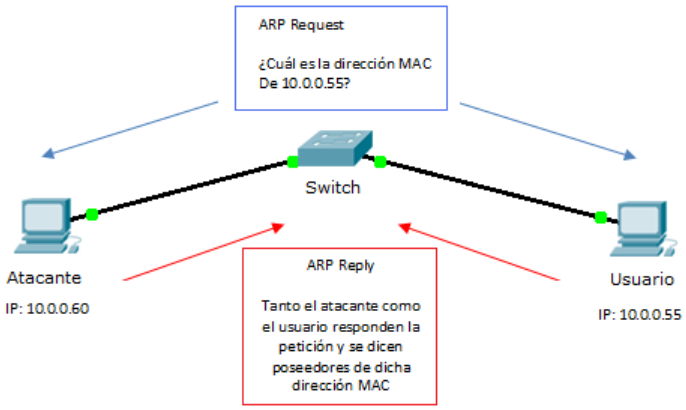

Figura 26. ARP poisoning.

2.5.3 Man in the middle

Es un ataque en el que se establecen conexiones independientes entre dos víctimas y se envían mensajes a ellas y/o entre ellas, haciéndoles creer que están efectuando una comunicación directa a través de una conexión privada cuando de hecho toda la conversación es controlada y manipulada por el atacante. Éste debe tener la posibilidad de interceptar todos los mensajes de la comunicación y hasta de inyectar nuevos en la misma. [91]

Este tipo de ataque es muy atractivo porque el *host* ya se encuentra autenticado con la víctima, siendo así, el atacante no necesita efectuar ningún tipo de *cracking*. No importa qué tan seguro sea un determinado proceso de autenticación porque la mayoría de los sistemas envían texto en claro una vez que tal proceso ha sido realizado con éxito. Esto hace que la mayoría de las computadoras sean vulnerables a este ataque.

Existen dos tipos:

- Activo. Es cuando el atacante se apodera de una sesión activa y compromete un objetivo específico.

[90] Gopi Nath, Shefalika Ghosh, "Different Flavours of Man-In-The-Middle Attack, Consequences and Feasible Solutions" en http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5563900, 30/04/2011.
[91] *Idem*.

- Pasivo. Es cuando el atacante se apodera de una sesión activa e intercepta todo el tráfico entre el *host* y el objetivo sin comprometer a ninguno de los dos. El tipo activo siempre inicia con uno pasivo.

Session replay y *Session hijacking* son considerados también ataques *Man in the middle*. [92]

A. Session replay

El atacante captura paquetes y modifica los datos antes de enviarlos al objetivo. (Véase figura 27.) [93]

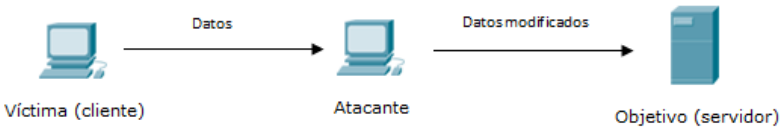

Figura 27. Session replay.

B. Session hijacking

El atacante se apodera de una sesión IP suplantando la fuente o el destino. Frecuentemente se realiza una denegación de servicio hacia el *host* mientras se suplanta a éste dentro de la red. (Véase figura 28.) [94]

Figura 28. Session hijacking.

2.5.4 Cracking WEP

WEP (Wired Equivalent Privacy) forma parte de los estándares 802.11 para conexiones inalámbricas. Utiliza una llave secreta que es compartida entre un cliente y un punto de acceso y es usada en conjunto con el algoritmo RC4 para cifrar todas las comunicaciones entre los mismos.

Puede operar con 40 o 104 *bits* de cifrado. Mientras más fuerte sea tal cifrado, más segura será la red.

---

[92] Andrew Whitaker, Daniel Newman, *Penetration testing and network defense*, Estados Unidos, Cisco Press, 2006, pp. 127-130.
[93] *Ídem*.
[94] *Ídem*.

El problema con *WEP* es el valor tan pequeño de su vector de inicialización que lo hace fácil de averiguar. El vector crea los primeros 24 *bits* de la clave *WEP*. Muchas implementaciones inician utilizando valores del vector iguales a cero y lo incrementan en uno por cada paquete que envían. 24 *bits* es igual a 16, 777, 216 valores, así que después de que se han enviado 16 millones de paquetes, el vector regresa a un valor de cero. Este comportamiento tan predecible de los primeros 24 *bits* de la clave *WEP* hace que ésta sea muy fácil de romper.

Además de lo anterior, en muchos casos no se modifica la clave *WEP* regularmente, provocando que los atacantes mantengan el acceso fácilmente.

Algunos ejemplos de herramientas que son comúnmente utilizadas para romper las claves *WEP* son:

- AirSnort. Es una utilidad de Linux que puede romper claves *WEP*. Requiere que el adaptador de red inalámbrico esté en modo de monitoreo. Captura paquetes de manera pasiva y luego intenta romper la llave cifrada. Con 5 o 10 millones de paquetes capturados, AirSnort puede romper la clave en menos de un segundo.

- WEPCrack. Es similar a AirSnort, pero no es tan popular. Está basado en el lenguaje Pearl.

- WifiSlax. Es una herramienta con utilidad de LiveCD basada en la distribución Slax de Linux, muy similar a las anteriores. [95]

2.6 Eliminación de evidencias

Casi al mismo tiempo que se desarrollara la técnica informática forense, se comenzaron a originar contramedidas para prevenir que los piratas informáticos fueran descubiertos, así, la eliminación de evidencias se hizo práctica común entre aquellas personas expertas en informática que no deseaban que sus acciones ilegales pudieran ser descubiertas.

A medida que se explora e investiga más sobre las técnicas anti-forenses, se han generado varias clasificaciones, como son:

- Destrucción de la evidencia.

- Ocultamiento de la evidencia.

- Eliminación de las fuentes de evidencia.

- Falsificación de la evidencia. [96]

[95] *Ídem*, pp. 352-357.
[96] Armando Botero, Iván Camero, Jeimy Cano, "Técnicas anti-forense en informática: ingeniería reversa aplicada a TimeStomp", Colombia, Pontificia Universidad Javeriana en http://www.fing.edu.uy/inco/eventos/cibsi09/docs/Papers/CIBSI-Dia3-Sesion6%283%29.pdf, 27/01/2010.

2.6.1. Destrucción de la evidencia

El principal objetivo de esta técnica es evitar que la evidencia sea encontrada por los investigadores y en caso de que esto no sea posible, disminuir radicalmente el uso que se le podría dar. Este método no busca que la evidencia sea inaccesible, sino irrecuperable.

Esto implica que se deben destruir, desmantelar o modificar todas las pruebas útiles para una investigación.

Existen dos niveles de destrucción de la evidencia:

- Nivel Físico: A través de campos magnéticos (discos duros, externos, por mencionar algunos).

- Nivel Lógico: Busca cambiar la posición de los datos, sobreescribirlos, eliminar la referencia a los mismos, entre otros.

Existe una variedad de herramientas para la destrucción de evidencia de las cuáles se pueden valer los intrusos para realizar este método. Un ejemplo de estas son: Wipe, Shred, PGP Secure Delete, Evidence Eliminator y Swap.

2.6.2 Ocultamiento de la información

Este método tiene como principal objetivo el hacer inaccesible la evidencia para el investigador. No busca manipular, destruir o modificar la evidencia, sino hacerla lo menos visible posible.

Esta técnica puede llegar a ser muy eficiente de ser bien ejecutada, pero conlleva muchos riesgos para el atacante, puesto que al no modificar la evidencia, de ser encontrada puede ser válida en una investigación formal y por lo tanto, servir para la incriminación e identificación del autor de dicho ataque.

Una de las herramientas utilizadas por los atacantes es la esteganografía, la cual trata de técnicas que permiten la ocultación de mensajes u objetos dentro de otros, llamados portadores, de modo que no se perciba su existencia. (Véanse figuras 29 y 30.) En el mercado se pueden encontrar muchas herramientas fáciles de usar tales como StegoArchive.

Figuras 29 y 30. Ejemplo de esteganografía. En este cuadro titulado "Los embajadores" de Holbein, llama la atención una mancha bastante rara entre los protagonistas en la imagen de la izquierda. Mirando el cuadro desde el ángulo correcto, aparece una calavera (imagen derecha). Este es un ejemplo de esteganogtrafía visual. [97]

2.6.3 Eliminación de las fuentes de evidencia

Este método tiene como principal objetivo neutralizar la fuente de la evidencia, por lo que no es necesario destruir las pruebas puesto que no han llegado a ser creadas. Por ejemplo, en el mundo real cuando un criminal utiliza guantes de látex para manipular el arma, lo que está haciendo es evitar dejar huellas dactilares.

Una de las acciones que los atacantes pueden llevar a cabo es la desactivación de los *logs* del sistema que se está atacando y la edición de la bitácora del mismo. [98]

2.6.4 Falsificación de la evidencia

Este método busca engañar y crear falsas pruebas para los investigadores forenses logrando así cubrir a el verdadero autor, incriminando a terceros y por consiguiente, desviando la investigación con lo cual sería imposible resolverla de manera correcta.

El ejercicio de este método se vale de una edición selectiva de las pruebas creando evidencias incorrectas y falsas que corrompen la validez de las pruebas de investigación forense, por lo cual no podrán ser tomadas en cuenta como evidencia. [99]

[97] Tomadas de http://danteslab.blogspot.com/
[98] "Armando Botero, Iván Camero, Jeimy Cano, "Técnicas anti-forense en informática: ingeniería reversa aplicada a TimeStomp", Colombia, Pontificia Universidad Javeriana en http://www.fing.edu.uy/inco/eventos/cibsi09/docs/Papers/CIBSI-Dia3-Sesion6%283%29.pdf, 27/01/2010.
[99] *Ídem*.

2.6.5 Herramientas utilizadas en las técnicas anti-forense

- TimeStomp. Es una herramienta anti-forense que puede leer y escribir los registros de tiempo o *time-stamps* de los archivos *NTFS (New Technology File System)*. Los registros de tiempo sirven para almacenar información de cuándo fue modificado, accedido, creado o modificado un archivo en el sistema *NTFS*. Según la clasificación anteriormente ofrecida, TimeStomp es una herramienta que destruye y falsifica información.

  Con este programa se consigue evitar que una analista forense obtenga una línea de tiempo de sucesos en un sistema. (Véase figura 31.)

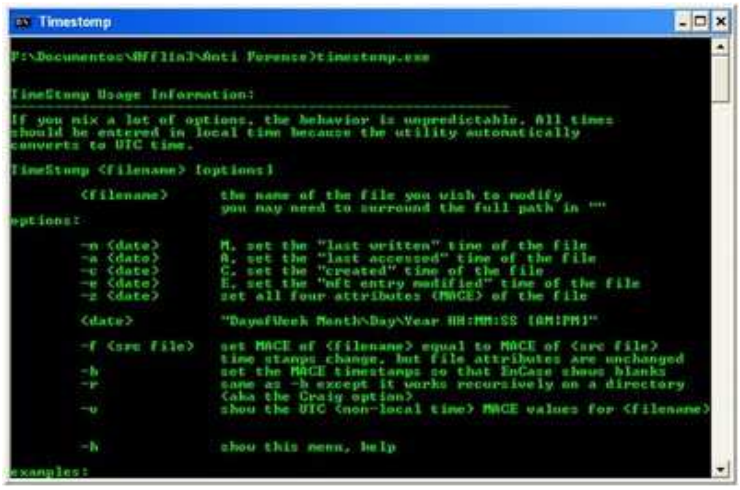

Figura 31. Pantalla principal de TimeStomp. [100]

- Shred. Es una utilidad disponible para sistemas Linux que sobreescribe los archivos especificados un número determinado de veces, 25 por defecto, con varios patrones de texto, convirtiendo así, el contenido del archivo original en otro totalmente distinto y llenándolo de información sin sentido, haciendo que la recuperación del mismo sea prácticamente imposible. (Véase figura 32.)

  Esta es una utilidad que destruye la información.

[100] Tomada de http://www.0ffl1n3.com/2009/12/tecnicas-anti-forenses.html

Figura 32. Vista de las opciones de la utilidad shred. [101]

- Evidence eliminator. Programa orientado al sistema operativo Microsoft Windows que permite eliminar información de forma casi permanente del disco duro. Sobre escribe la información a ser eliminada con el fin de que su recuperación sea prácticamente inviable. Esta aplicación destruye la información. [102] (Véase figura 33.)

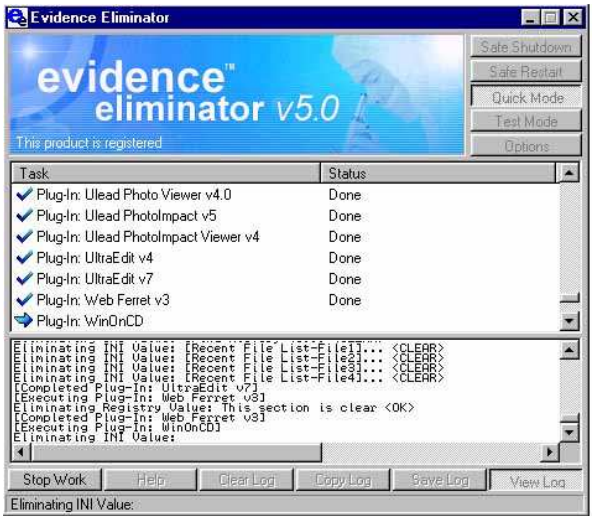

Figura 33. Pantalla de funcionamiento de Evidence Eliminator. [103]

101 Tomada de www.forensicswiki.org/wiki/Timestomp
102 Armando Botero, Iván Camero, Jeimy Cano, "Técnicas anti-forense en informática: ingeniería reversa aplicada a TimeStomp", Colombia, Pontificia Universidad Javeriana en http://www.fing.edu.uy/inco/eventos/cibsi09/docs/Papers/CIBSI-Dia3-Sesion6%283%29.pdf, 27/01/2010.
103 Tomada de http://www.pctrackscleaner.com/images/evidence-eliminator-screen-12.jpg