Rootkits para el Kernel Linux 2.6

El presente artículo fue desarrollado por Pablo Fernández.

Pablo es un desarrollador argentino de 21 años con casi 6 años de experiencia en GNU/Linux y 4 años en el campo de la seguridad.

Pablo ha contribuído con el Open Source, es autor del cliente de correo Cronos II de GNOME, de proxychain y ha contribuído en proyectos tales como Nmap (siendo el creador del método más reciente de exploración cautelosa y completa de proxys), entre otros.

Pablo puede ser visitado en su sitio web http://www.littleq.net/

***Traducción:** Lic. Cristian Borghello para www.segu-info.com.ar*

Conocer el funcionamiento de los rootkits tiene un enorme valor desde varios puntos de vista; un atacante no se apropia realmente de un sistema hasta que no se ha ocupado, de una manera conveniente, de controlarlo por completo. Un administrador de sistema necesita saber como trabajan estos para poder evaluar si un sistema ha sido comprometido.

Este artículo describirá las técnicas más importantes utilizadas por un rootkit LKM que se utiliza en la vida real, llamado **SIDE**, y que se construyó para la serie 2.6 actual del Kernel de Linux.

Ocultando el módulo

Ya que el rootkit se ejecutará dentro del sistema como un módulo del kernel, hay que tener mucho cuidado para que no sea descubierto con comandos tales como “lsmod” o a través de “/proc/modules”. El código en la **Listado 1** se ocupa de ello. Para comprender correctamente cómo funciona es importante entender como se organizan los módulos dentro del kernel y la teoría que hay detrás de esta técnica de ocultación.

Listado 1. Ocultando el módulo

```
      lock_kernel(); /* Held the kernel lock to prevent faulting in SMP
systems */
      __this_module.list.prev->next = __this_module.list.next;
      __this_module.list.next->prev = __this_module.list.prev;
      __this_module.list.prev = LIST_POISON1; /* A common practice in
kernel development */
      __this_module.list.next = LIST_POISON2; /* to invalidate a list that
shouldn't be used */
      unlock_kernel();
```

Básicamente, a través de este código el módulo se separa de los de módulos cargados de la lista interna circular doblemente enlazada del kernel.

Ocultando procesos

La capacidad de ocultar los procesos a cada usuario del sistema (incluyendo al root) es una de las características fundamentales que un rootkit debe implementar.

Las herramientas de espacio de usuario (tales como ps(1) o top(1)) conocen las tareas leyendo el directorio “/proc”. Cada tarea que se ejecuta en el sistema crea una entrada de la form “/proc/<PID>“, donde se puede obtener la información útil sobre ese proceso. Lo que hacen las herramientas de espacio de usuario es abrir el directorio “/proc” y consultar la existencia de “/proc/<n>“, donde 1 <= n <= pid_max. Si el directorio no existe se asume que el PID está libre, mientras que si existe se puede recopilar información del mismo.

Con esta afirmación en mente y el conocimiento sobre las particularidades del VFS es posible hacer creer a las herramientas del espacio de usuario que los PIDs existentes están realmente libres.

Esto se logra interrumpiendo la llamada “readdir” en la capa VFS. Para lograr esto debe modificarse la tabla que contiene la dirección a la llamada “readdir” con la dirección del nuevo segmento de código que reimplementa esta función.

Esto tiene como propósito que el argumento “filldir” pueda ser modificado para que apunte hacia una implementación diferente del mismo, lo que descartará a aquellos directorios que identifican a los PIDs ocultos.

Listado 2. Fragmento de reimplementación filldir de “/proc”

```
      if (!(process = _atoi(name, &process))); /* If this isn't a PID just
call the original filldir */
      else if (!process_is_authed(current) && process_is_hidden(process))
/* If process is hidden */
            return 0; /* don't show it (unless current is superroot) */

      if (p_proc_filldir)
            return p_proc_filldir(buf, name, nlen, off, ino, x);
```

Los detectores de rootkits

Los detectores del rootkit utilizan una técnica para encontrar procesos ocultos que consiste en enviar una señal SIGCONT a todos los procesos posibles.

Estas señales se envían hacia los procesos a través de la llamada del sistema “kill(2)”. Cuando se envía una señal a un proceso que no existe, “kill(2)” devuelve el valor -1 y el “errno” se establece a “ESRCH”, mientras que si un proceso existe, “kill(2)” devuelve 0.

De este modo, los detectores del rootkit detectan si los procesos existen o no, sin utilizar los datos de “/proc”. Después de esto, la lista que se crea es comparada con la lista de procesos que muestran los “/proc”. Si se encuentra una diferencia entre ambas listas significa que hay un proceso que está oculto para el espacio de usuario.

Desde el punto de vista del rootkit, engañar a los detectores que utilizan esta técnica es cuestión de prolongar el código. Todo lo que se necesita es que el rootkit intercepte la llamada del sistema a “kill(2)” y si alguien envía una señal que no sea el usuario “superroot” hacia un proceso que está en estado oculto, la nueva función de rellamada debe devolver “ESRCH”, pero si esto sucede debe llamarse a la función kill(2) original y que devuelva su valor de retorno.

Listado 3. Reemplazo de sys_kill()

```
asmlinkage int new_kill(pid_t pid, int sig) {
      struct siginfo info = { .si_signo = sig, .si_errno = 0,
.si_code = SI_USER, .si_pid = current->tgid, .si_uid = current->uid
};
      if (!process_is_hidden(pid) || process_is_authed(current))
            return kill_proc(pid, sig, &info);
      return -ESRCH;
}
```

Aftermath: técnica de detección de rootkits

Hay muchas maneras en las que un detector de rootkit reconoce a los rootkits. Ya que se puede engañar fácilmente a las herramientas del espacio de usuario, no hay necesidad de mantener detectores de rootkit en ellas. Es tan fácil como comprobar las modificaciones de la "sys_call_table" una vez están en el espacio del kernel. También seria fácil comprobar los procesos ocultos, ya que estos no pueden desprenderse de la lista de procesos tan fácilmente como los módulos; su presencia en dicha lista es la única garantía de que tendrán fragmentos en el tiempo de ejecución.

Mientras que algunos detectores de rootkit llevan a cabo algunas de estas técnicas, la mayoría permanecen en el espacio de usuario. La lección para los administradores paranoicos es no depender ciegamente de los detectores de rootkit.

Ocultando las conexiones de red

Los programas y aplicaciones del espacio de usuario conocen el tráfico de red a través de las entradas "/proc/net", dentro de esta ubicación hay muchos elementos (que tienen la apariencia de archivos, pero son realmente estructuras "proc_dir_entry"), como el "tcp" y el "tcp6" (si está activada la "CONFIG_IPV6"). Estas entradas contienen la información del tráfico de red existente en el sistema.

También pueden interceptarse las llamadas "read" utilizando un método diferente, así la información devuelta puede modificarse en tránsito. Mientras la conexión de red esté aún activa (o el socket esté en estado LISTEN) "netstat" y herramientas similares no podrán verla.

SIDE provee una excelente manera de ocultar las conexiones de red, muy parecida (aunque ni remotamente tan poderosa) a la del Netfilter. Se define una lista de condiciones y comandos durante el tiempo de ejecución (ver la Tabla 1) y cuando se necesite la información sobre los sockets la lista se hace corresponder con cada socket, si se utiliza alguna regla se ejecuta el comando asociado a la condición.

El comando puede utilizarse ya sea para mostrar u ocultar el socket en el espacio de usuario. Esta lista es muy potente. Las acciones predeterminadas pueden definirse con la condición "all" que está al final de la lista, que puede ser completamente manipulada durante la ejecución (e incluso puede ejecutar comandos predeterminados cuando se carga el módulo).

Tabla 1. Lista de commandos

Command	Example	Description
net.hide.src=[IP]	net.hide.src=192.168.0.10	Oculta las conexiones de red en las que en la dirección local es [IP]
net.show.dstport=[PORT]	net.show.dstport=22	Muestra todas las conexiones de red en las que el puerto remoto es [PORT]
sys.superroot=[KEY]	sys.superroot=dSi2d_q@d	Obtiene permisos de superroot si la clave [KEY] es correcta
sys.hide=[PID]	sys.hide=1	Oculta los procesos con PID [PID]
sys.show=[PID]	sys.show=5982	Muestra el proceso ocultos con PID [PID]
sys.guid=[UID],[GID]	sys.guid=1000,1000	Elimina superroot, establece UID con [UID] y GID con [GID]
fs.hide=[FILENAME]	fs.hide=dfdfdf-nc	Oculta archivos con nombre [FILENAME]
fs.show=[FILENAME]	fs.show=dfdfdfdf-arpspoof	Muestra archivos ocultos con nombre [FILENAME]

El método empleado en la ocultación de conexiones de red consiste en ponerle un indicador a la “proc_dir_entry” del protocolo que se necesita interceptar como el TCP.

“proc_dir_en” que pertenecen al espacio de “/proc” pueden encontrarse a lo largo de la lista circular doblemente enlazada en el “proc_net->subdir”. Repitiéndolo por completo y comprobando que el nombre en el miembro “node->name” sea correcto debería funcionar.

Cuando se ha marcado esta estructura, el indicador “seq_show” necesita ser reescrito con una nueva implementación de esta función. La implementación de SIDE proporciona los datos correctos (utilizando la función original) y cada línea de datos se corresponde con las reglas cargadas, que aplican acciones específicas en las líneas correspondientes.

El problema

Debido a la naturaleza del tráfico de red es imposible ocultar realmente la actividad a los ojos de un administrador sensato. Con frecuencia se aprecian varios saltos entre el sistema comprometido y el otro extremo de la comunicación oculta. Estos saltos mostrarán claramente el tráfico oculto y no hay nada que hacer al respecto.

De hecho, cuando aún no está establecida una conexión, si el socket en estado LISTEN está oculto, el nmap revelará que hay un puerto abierto que netstat no muestra. Por supuesto que se puede hacer algo para evitar este tema utilizando Port Knocking, pero una vez que la conexión se abre y el tráfico tiene lugar será fácil detectar este último.

Un truco ingenioso sería no establecer conexión alguna, el intercambio de paquetes ICMP o UDP es una forma interesante de controlar un sistema y obtener información sobre su estado. D este modo, el atacante puede controlar el sistema comprometido sin dejar huellas, utilizando paquetes UDP o ICMP falsificados.

Ocultando archivos

A menudo un sistema es comprometido para ser utilizado como una plataforma segura desde la cual se lanzan ataques [D]DoS, o para ser empleados como salto entre el atacante y otro sistema comprometido. La mayor parte del tiempo el atacante necesitará descargar algunos archivos del sistema para llevar a cabo estos ataques. Por supuesto, si el

administrador encuentra tales herramientas surgirán algunas preguntas. Es así que un rootkit siempre debe ser capaz de ocultarle archivos a cualquier usuario, incluso al root.

Vamos a utilizar una vez más los conocimientos de VFS para realizar estas acciones, empleando exactamente el mismo método que se usó para ocultar los procesos.

Esta vez el objeto “fs” almacenará una lista de los nombres de los archivos ocultos. Estos nombres carecen de una ruta, de modo que cualquier archivo oculto en un directorio implicará la ocultación de cada archivo con el mismo nombre en todos los directorios. El propósito de esta función es forzar al usuario superroot a utilizar nombres no standares pues aún existen otros métodos que no están siendo manipulados, por ejemplo, aunque los archivos ocultos no serán enumerados en los directorios todavía se podrá acceder a ellos a través de las llamadas del sistema tales como la “open(2)”, “stat(2)”, etc.

Actualmente SIDE no protege de estos métodos a los archivos ocultos, aunque todo lo que se necesita es sustituir esas llamadas del sistema (y algunas otras como “rename(2)”).

Sugerencia: sería un buen ejercicio desarrollar estas funciones cuando termines de leer este artículo.

Observa que la metodología empleada depende del sistema de archivos. Si se desea ocultar archivos en diferentes puntos de montaje tiene que modificarse SIDE en el archivo “vfs.c” para que se oculte de estos puntos de montaje. Es totalmente seguro utilizar la misma “readdir” y “filldir” del sistema de archivos que emplea el root.

El problema

Otra vez, la acción de ocultar archivos tiene un problema intrínseco propio, similar al problema con las conexiones de red.

Los archivos se guardan en disco y los mismos se pueden acceder de distintas maneras como con un CD-ROM de rescate y montando las partición root. Ya que el rootkit no está cargado en el kernel ni en ejecución, los archivos ocultos ya no lo están tampoco. Hay diferentes formas de hacer que estos archivos sean más difíciles de encontrar. La protección más fácil y frecuente es la seguridad por oscuridad, almacenando archivos en forma no estándar con nombres poco descriptivos y confusos.

Un mejor enfoque es el de almacenar todos los archivos del sistema en sistema de archivos de lazo cerrado (loopback filesystem), claro que con esta metodología debe tenerse cuidado para que este sistema de archivos no se vea con “mount(8)” o “/proc/mounts”. Esta metodología también permite el cifrado más fácil del sistema de archivos, y por más que los administradores sospechen, no podrán ver su interior.

Usuarios normales con permisos de root

El usuario superroot es identificado por un UID y GID distinto de cero, y de este modo este usuario carece de permisos de root, por supuesto esto es absolutamente inaceptable.

SIDE implementa un mecanismo para establecer un UID de 0 a casa proceso ejecutado por superrot.

Esto también es hecho en la llamada interrumpida de “lookup()”, en el directorio “/proc”. Cada vez que un proceso autenticado accede a cualquier cosa en él, su UID y los otros valores relacionados se colocan a 0 (root) y algunas capacidades son activadas por completo (*cap_effective*, *cap_inheritable* y *cap_permitted*). Si el proceso no accede a nada que este en el “/proc”, se ejecutara con la UID del superrot. Es por eso que algunos programas

extremadamente pequeños y simples no se identificarán como root. Tal es el caso de “whoami”.

Usabilidad en tiempo de ejecución

SIDE provee una interfaz muy cómada en tiempo de ejecución. En el “vfs.c” el rottkit intercepta las llamadas a “lookup()” en el “/proc”. De esta forma el usuario superrot puede interactuar con el rootkit, para hacer que este por ejemplo, oculte un proceso o conceda permisos de root (o superrot) a algún usuario. Es muy fácil enviar comandos al rootkit. Todo lo que tiene que hacer el usuario es intentar acceder al archivo en el sistema de archivos de “/proc”. El nombre del archivo será interpretado como el comando.

SIDE organiza los commandos por objetos, así que, dependiendo de lo que el usuario quiera hacer, se ejecutan los comandos contra los objetos específicos.

Actualmente, SIDE reconoce tres objetos diferentes: *net*, para la manipulación de la red; *sys* para la administración de procesos y las propiedades relacionadas al usuario; y *fs* para la manipulación de la lista de archivos ocultos.

Hay varios comandos para estos objetos. Alguno de ellos aparecen en la Tabla 1. El resto de ellos se puede encontrar en el archivo COMMANDS.txt dentro del paquete.

Los comandos deben ser ejecutados de la siguiente forma:

```
echo > /proc/[object].[command[=args]]
```

Módulos

Los módulos cargados se almacenan dentro del kernel en una lista circular doblemente enlazada, donde cada nodo de la lista representa un struct module (definida en “include/module.h”). Los módulos con frecuencia se reúnen leyendo la entrada “/proc/modules”.La lista de módulos es creada por “m_show”, en el “kemel/module.c”, pasando por la lista enlazada mencionada previamente.

El hecho de que esta lista sea accesible desde cada módulo permite su manipulación y modificación. La técnica empleada para ocultar el módulo es desprender el nodo de este de la lista enlazada, conectándole directamente el valor previo y el siguiente dentro de la lista.

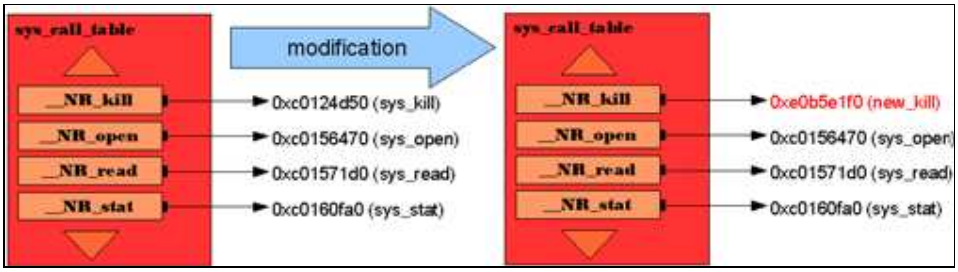
Llamadas de sistema

Las llamadas del sistema son la interfaz que reside en el kernel, que comunica al espacio de usuario con este.

Cada cosa que vaya desde el kernel hacia el espacio de usuario o viceversa tiene que pasar por una llamada del sistema. Esta es la razón principal por la cual los rootkits siempre se han interesado en interceptarlas, pues controlarlas se traduce en verificar lo que el usuario puede ver y hacer.

Hay muchas llamadas del sistema como la “open(2)”, la “read(2)”, etc. Todas estas funciones son señaladas por punteros en una matriz llamada Tabla de Llamadas del Sistema conocida como “sys_call_table”.

Históricamente, el modificar la “sys_call_table” era sólo cuestión de reescribir el puntero deseado a una nueva dirección de memoria con la nueva función, de esta forma cualquier llamada del sistema (excepto la “execve(2)” que necesita ser manipulada con más cuidado) podía ser interceptada fácilmente.



(Des)afortunadamente Linux 2.5.41 ya no exporta el símbolo de la “sys_call_table” y mientras que este aún exista, la dirección de la memoria no está disponible para los módulos.

Para encontrar la dirección correcta hay una técnica que se puede utilizar: la “/usrlsrc/linux/include/asm/unistd.h” que enumera el orden de la “sys_call_table” con constantes __ NR que define el offset en la matriz en la que encuentra cada llamada del sistema y luego que se conoce la dirección de cada llamada (cada una de ellas se exporta), se puede escanear la memoria buscando en la “sys_call_table”.

El código que realiza esto sólo declara a un puntero que se inicia en una dirección baja de memoria (habitualmente se utiliza la dirección de “loops_per_jiffy”) y hace un bucle hasta un offset __NR del indicador se corresponda con con la dirección correcta de la misma llamada del sistema.

Si se alcanza una dirección alta de memoria (como “boot_cpu_data”), algo malo ha sucedido (quizás un módulo ya este interceptando la llamada del sistema que se esta utilizando para observar la “sys_call_table”), lo que significa que la tabla de llamadas del sistema no se pudo encontrar. Si la tabla de llamadas del sistema no se encuentra, las interrupciones de las llamadas del sistema no serían posibles. Observar que esto es completamente independiente de las interrupciones VFS.

Listado 4. Buscando sys_call_table

```
unsigned long ptr;
extern int loops_per_jiffy;
for (ptr = (unsigned long) &loops_per_jiffy; ptr < (unsigned
long)&boot_cpu_data; ptr += sizeof(void *)) {
      unsigned long *p;
      p = (unsigned long *)ptr;
      if (p[__NR_close] == (u32) sys_close) /* When this condition is
met p points to sys_call_table */
      return (u32 **) p;
}
```

Cuando se ha encontrado la “sys_call_table”, cambiarla es tan simple como lo era en el pasado. Ver en la Listado 3 un ejemplo de cómo sustituir la llamada del sistema “open(2)”.

Listado 5. Interceptando una llamada del sistema

```
u32 **sys_call_table;
asmlinkage int (*old_open)(const char *, int, mode_t);

if ((sys_call_table = find_sys_call_table())) {
      old_open = (void*) sys_call_table[__NR_OPEN];
      sys_call_table[__NR_OPEN] = (u32*) new_open;
}
```

Es importante tener en cuenta que las llamadas del sistema son muy críticas para el mismo. Si se intercepta una llamada del sistema y la nueva función de rellamada (en el

Listado 3, new_open) no se comporta correctamente, el sistema tampoco se comportará del mejor modo, y lo más probable es que se vuelva completamente inestable. Llamar a la llamada original del sistema desde la nueva función de rellamada es una práctica común en la que la nueva función de rellamada decide permitir su ejecución. Esa es la razón exacta por la cual el código en el Listado 3 guarda un indicador a la función original, y también, por la que cuando el módulo se va a descargar la "sys_call_table" debe modificarse para que señale a la localización original.

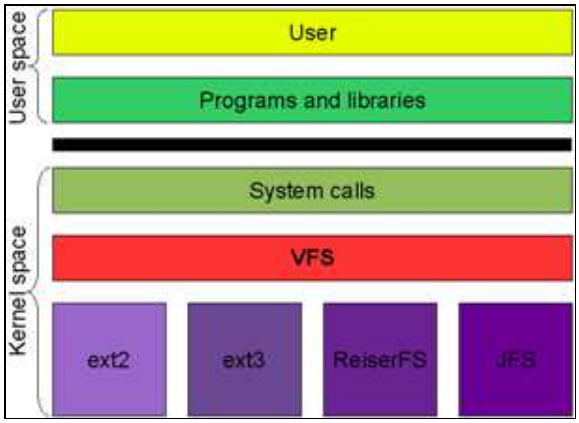
El VFS Internamente

El Sistema Virtual de Archivos o el Interruptor del Sistema de Archivos Virtual es una capa entre las llamadas del sistema relacionadas con los archivos (como la "open(2)") y los sistemas de archivos actuales (como las ext2, ext3, reiserfs, jfs, etc.). Este proporciona una interfaz común para facilitar el trabajo de los que utilizan el sistema de archivos.

Las implementaciones del sistema de archivos tienen que definir un conjunto de funciones predefinidas y métodos, y notificar a la capa VFS de esos métodos, los cuales son invocados por ellos como funciones de rellamada a través de punteros de función. El VFS tiene esencialmente una estructura que cada sistema de archivos tiene que completar y registrar en la capa VFS. En esa estructura reside la información necesaria para encontrar las direcciones en las que esas funciones de rellamada tienen que encontrarse.

Durante el desarrollo de un rootkit, la llamada más interesante será sin duda la "readdir". Esta función de rellamada proporciona el algoritmo para llamar al parámetro de la función "filldir", la cual será llamada por cada archivo o directorio read por "readdir". El valor devuelto se utiliza para preparar la información sobre el directorio read.

A lo largo de este artículo se utiliza mucho una técnica que emplea el valor 0 devuelto en la función "filldir". Este valor de retorno hace que la "readdir" descarte la información del asunto leído.



Superroot

Si algún usuario del sistema fuese capaz de controlar un rootkit que tuviese la capacidad de someter a un sistema, este no sería muy bueno. Antes de que SIDE ejecute cualquier comando, los usuarios deben autenticarse con el rootkit, esto se hace con el comando "sys.superroot". Para que este comando autentifique con éxito al usuario, la clave tiene que especificarse como un parámetro del comando.

La clave es (usualmente) una cadena aleatoria que identifica la instalación. SIDE selecciona la clave para cada instalación cuando se ejecuta el script “configure”.

Cuando se introduce la clave correcta la UID y la GID del usuario se cambian por aquellas que identifican al usuario superroot (seleccionado también en “configure”).

Se necesita del usuario superroot para permitir la ejecución de los comandos y evitar que se oculte información sobre ese usuario en particular que está oculto para otros usuarios.

Conclusión

Diversas técnicas y enfoques han sido analizados a lo largo de este artículo en cuanto al desarrollo del rootkit en la serie 2.6 del Kernel de Linux. Se revisó la metodología para ocultar las conexiones de red, los procesos, los módulos y los archivos y también las contramedidas que utilizan los detectores de rootkit, así como las nuevas contramedidas que debería ponerse en práctica por los detectores de rootkit y administradores.

El desarrollo dentro del Kernel Linux trae consigo un nuevo mundo de oportunidades de todo tipo. Esto es sólo la punta del iceberg, la madriguera del conejo va mucho más profunda.

.

En la Web

http://www.littleQ.net/SIDE/ – SIDE homepage.

Glosario

LKM – Linux Kernel Module
VFS – Virtual File System or Virtual Filesystem Switch