Soporte de red en rootkits de linux

Mikel Conde, Gorka Rodriguez, Jon Ander Ortiz

22 de enero de 2007

Resumen

Los sistemas de rootkits, tradicionalmente se han caracterizado por la ocultación de información al usuario, o la elevación de privilegios en los sistemas comprometidos. Estas acciones aunque apreciadas por los usuarios de los rootkits, no aprovechan el potencial de un sistema comprometido hasta dicho nivel. Siguiendo este concepto, se han realizado algún estudio sobre la comunicación de un sistema comprometido con otros sistemas ajenos.

El siguiente artículo, estudia la capacidad de un rootkit de linux, de acceder, modificar e inyectar tráfico de red sin ser descubierto por los sistemas habituales de detección de intrusiones. De esta manera, se utilizaría el potencial real de una infección a nivel de núcleo de sistema operativo.

1. Introducción rootkits

Un rootkit es un conjunto de herramientas software que permiten a un intruso conseguir acceso privilegiado a un sistema comprometido[17], habitualmente este tipo de herramientas es instalado en el sistema huésped tras un ataque que ha logrado su objetivo, es decir, ha logrado privilegios de administrador[1]. Habitualmente este tipo de software, realiza acciones sobre el sistema para que el atacante pueda controlar la máquina huésped, borre las huellas del ataque o esconda información, aunque las tareas que pueden llevarse a cabo en este tipo de sistemas son tan amplios como lo permita la plataforma para la que están implementados.

Los rootkits, son divididos en dos grandes familias: los que se ubican en el núcleo del sistema operativo, y los que se encuentran en zona de usuario. Los rootkits que funcionan como aplicaciones en la zona de usuario, habitualmente realizan suplantación de archivos troyanizados[2][15], o bien modificando el comportamiento de los programas inyectando código a los mismos[6]. El otro tipo de rootkits, suele utilizarse añadiendo código a la zona de kernel bien mediante

[1] En los sistemas UNIX-Like, habitualmente se conoce como root al administrador del sistema. Esta característica ha propiciado el nombre *"rootkit"*.

[2] Archivos modificados para que realicen acciones para las que no fueron programados.

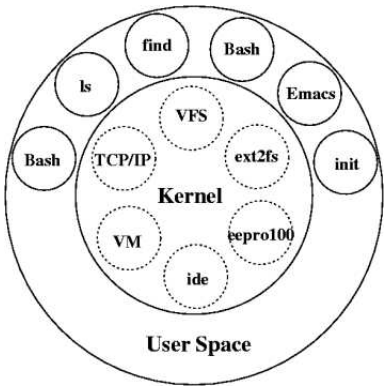

Figura 1: Ejemplo de la distribución de los sistemas UNIX-like (en concreto linux) en dos grandes partes: Kernel y zona de usuario

un driver[3], o bien mediante un módulo[4].

Este tipo de software se da en todos los sistemas operativos, tanto Windows[4, 7], como derivados de sistemas UNIX (También conocidos como sistemas Unix-Like), el presente documento, se centrará en los sistemas Unix-Like, concretamente en los rootkits desarrollados para los sistemas linux a nivel de Kernel.

2. Rootkits a nivel de kernel en Linux

Los sistemas basados en kernels Linux, se han caracterizado por la fiabilidad y la seguridad de los sistemas que componen, pero se daba la situación de que si un ataque lograba su objetivo, era factible, inyectar módulos subversivos al kernel, que permitieran al atacante ser indetectable y llegar a tener un control absoluto sobre la máquina atacada[14]. Por ello, en la rama de desarrollo del kernel 2.5 se decidió modificar ciertos aspectos para que el desarrollo de este tipo de software fuese al menos, un poco mas complejo[10].

Este cambio implicaba que los módulos que estaban compilados para kernels 2.4 o inferiores dejaban de funcionar para kernels 2.6.[5], pero rápidamente nuevas técnicas fueron desarrolladas para burlar las trabas puestas por los desarrolladores del kernel[14, 10], y en la actualidad podemos encontrar multitud de rootkits para los mas modernos kernels:

- Adore: Se trata de un módulo del kernel, que realiza, la ocultación de

[3]Un driver es un pedazo de código que se encarga del tratamiento de un HW concreto a bajo nivel. En los sistemas Windows los rootkits se implementan como drivers[4], por ello, tras muchos problemas de seguridad, Microsoft utilizó la técnica de firmar digitalmente los drivers para que los mismos no comprometieran la seguridad del sistema operativo.

[4]En los sistemas Linux, existen los LKM (Loadable Kernel Module), que son binarios que se adjuntan a el grueso del kernel.

[5]En concreto, se suprimió la exportación del símbolo del kernel sys_call_table, que es el vector de direcciones de las llamadas al sistema.

sí mismo, de procesos, ficheros mediante la capa VFS[6]. Desarrollado por teso-team[16].

- SuckIT: Módulo del kernel de linux, que reemplaza las syscalls, obtiene la dirección del vector de syscalls desde zona de usuario, esconde conexiones y procesos, así como abre una puerta trasera en el sistema[3].
- eNYeLKM: Modulo que implementa una técnica nueva de interceptar las syscalls sin modificar el vector de sys_call_table, implementa auto-ocultación y obtención de shell de root [10].
- Knark: Se trata de un módulo implementado que modifica ciertas llamadas al sistema y oculta conexiones ficheros y procesos [9], el rootkit incluye una colección de exploits [2].

Muchos de estos rootkits implementan la ocultación de conexiones TCP/IP, de procesos o bien de ficheros, pero hasta hoy en día, las funcionalidades de los rootkits para con las funciones de red, se han limitado a pruebas de concepto [5, 13, 1], en las cuales, se enumeran las técnicas existentes para manejar el tráfico de red LKM´s. Estas técnicas, aunque han sido descritas y publicadas en los ámbitos de este tipo de software aún no han sido aplicadas en los rootkits más comunes como los que han sido comentados anteriormente.

3. Acceso a la red desde el kernel

El acceso a los sistemas de red en el kernel del linux se realiza desde API´s o funciones a bajo nivel, es decir, se acceden a recursos a un bajo nivel, pudiendo tener un acceso mas complicado, pero más potente que en otro tipo de sistemas. A continuación se detallarán las técnicas y los métodos utilizados actualmente para acceder al tráfico de red desde la zona de kernel.

3.1. Acceso al tráfico del sistema

Como tráfico del sistema, consideramos el tráfico de red que se genera desde la zona de usuario o bien desde las pilas de protocolos del sistema operativo. Existen diferentes maneras de acceder a este tráfico, e incluso de tener acceso a modificar dicho tráfico antes incluso de que lleguen a la zona de usuario o que salgan del sistema hacia otros lugares.

En el siguiente apartado se muestran las dos técnicas mas elegantes existentes[13] para capturar dicho tráfico de red[7].

[6] Virtual File System

[7] Podrían existir otros métodos sobreescribiendo funciones internas del kernel de linux (no todas ellas se encuentran exportadas), o bien sobreescribiendo punteros a función (como por ejemplo la función poll del dispositivo de red)

3.1.1. Netfilter

Este método utiliza el sistema de gestión de tráfico Netfilter, este sistema, se encarga de gestionar el enrutamiento, filtrado y manipulación de los paquetes de red que llegan hasta el sistema. El sistema IPTABLES, es un cliente de Netfilter, que se encuentra implementado mediante una arquitectura cliente / servidor[11], en el que el sistema Netfilter, ofrece una arquitectura que permite que diferentes sistemas[8], puedan acceder al tráfico del sistema.

La arquitectura de Netfilter, está pensada para que diferentes sistemas puedan enlazarse en varios puntos del tratamiento de tráfico de red en el núcleo del sistema. Por lo tanto, es un punto en el que los atacantes pueden acceder al tráfico del sistema.

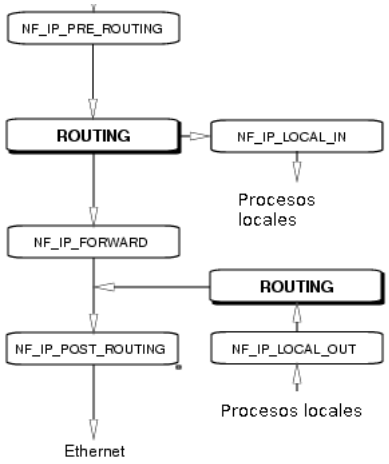

Figura 2: Arquitectura de Netfilter y diferentes puntos de acceso al tráfico de red.

La interacción que esta API ofrece no solamente se basa en la lectura de los paquetes que se reciben o envían, sino que es posible modificar e incluso eliminar paquetes[8] de cualquiera de estos puntos. Este método, es el utilizado por herramientas no-subversivas como el Snort-Inline[9].

3.1.2. Ptype Handlers

Esta técnica, descrita en varias investigaciones[12, 5], permite, igual que el método explicado anteriormente, acceder a el tráfico de red del sistema desde un módulo del kernel de linux, este acceso se realiza en el kernel a un nivel más bajo que los filtros Netfilter explicados anteriormente. El método consiste en utilizar el sistema de protocolos en nuestro favor, cuando un paquete sale del sistema, hay que asociar a dicho paquete a una pila de protocolo para que se

[8] Como por ejemplo el sistema de NAT (Network Address Translation)
[9] Implementación en modo prevención del Snort, el IDS más extendido.

encargue de gestionar la misma, el sistema de gestión de protocolos de linux, permite registrar nuevos tipos de protocolos en tiempo de ejecución por lo que el sistema se basa en registrar un nuevo protocolo para todo tipo de paquetes. Por lo tanto, todo tráfico hacia fuera o hacia dentro del sistema, es recibido por nuestro nuevo protocolo (además de por el resto de protocolos), con la posibilidad de modificar los parámetros de los mismos[5].

Cabe destacar que este proceso es más complejo de detectar que el sistema anterior, dado que para comprobar si este punto sensible del kernel ha sido manipulado habría que comprobar *ptype_base* y *ptype_all*, que son las estructuras de memoria que contienen los protocolos existentes, pero por desgracia, estas estructuras no están exportadas en el kernel de linux, lo que hace mas compleja la localización de este tipo de hooks, en el otro sistema, en cambio la estructura de memoria que almacena los hooks sobre las diferentes etapas de netfilter si que esta disponible[10]

3.2. Inyección de nuevo tráfico

Esta característica, no ha sido utilizada en ningún rootkit, ni en ningún sistema subversivo conocido, dado que, la inyección de tráfico nuevo tiene un gran problema: ocultar dicho tráfico al sistema que lo está emitiendo, para evitar estos problemas, se explican a continuación metodologías para evitar que dicha información transcienda.

3.2.1. Ocultando el tráfico inyectado a la zona de usuario

Los administradores de sistemas suelen utilizar herramientas de detección de intrusiones o de inspección del tráfico de red que recibe cada ordenador (Sniffers o IDS[11]), estas herramientas tradicionalmente funcionan mediante una conocida librería: libpcap[12]. Por lo tanto, para que la inyección de tráfico no fuese rápidamente detectada, se propone una técnica de ocultación publicada anteriormente[1], que permite controlar, mediante la intercepción de las llamadas al kernel de la implementación del paso de paquetes a la zona de usuario, los paquetes que llegan a dicha librería.

3.2.2. Inyección de tráfico mediante *hard_start_xmit*

La inyección de tráfico, es una de las posibles técnicas para interactuar desde el kernel de linux para con el mundo. Hasta el momento, se han publicado maneras encubiertas de enviar información desde el kernel[12], hacia el resto de

[10] En concreto se encuentra en */include/linux/netfilter.h* (*extern struct list_head nf_hooks...*)y en *net/netfilter/core.c*, se puede observar como se inserta dicho hook en la lista exportada para todo el kernel.

[11] Intrusion Detection System, Sistemas de Detección de intrusiones.

[12] Esta librería utiliza un sistema que se utiliza para realizar pilas de protocolo en zona de usuario, por lo que tanto todos los paquetes que se envían como los que se reciben son remitidos a la zona de usuario por si pertenecen a alguno de estos protocolos del nivel de usuario.

ordenadores, o de interactuar con el entorno de red del sistema, mediante la alteración de los números de secuencia aleatorios de las conexiones TCP. Este método, implica la posesión del control de una máquina intermedia para rescatar dicha información de la red.

En el presente artículo, se plantea una nueva técnica, que se trata de utilizar la instrucción *hard_start_xmit* para enviar paquetes hacia el exterior del sistema, y utilizar la técnica anterior (3.2.1) para ocultar dicho evento a la zona de usuario del sistema comprometido.

Los dispositivos de red en linux, definen una serie de operaciones que permiten la interacción estandarizada con los mismos, entre ellas una de las más importantes se trata de la instrucción *hard_start_xmit*, que pasando como parámetro, el dispositivo mediante el cuál deseamos inyectar el tráfico, y el *skb*[13] con el paquete que queramos inyectar a la red. Esta instrucción es la de más bajo nivel posible, dado que es una llamada directa a los dispositivos de red, y para utilizarla hay que ensamblar y cálculos los checksums[14] de los paquetes a enviar.

Por lo tanto, este camino permite la inyección de tráfico, y utilizando las técnicas de ocultación de datos a la librería libpcap, enviar datos hacia el exterior del sistema, sin tener que poseer el control de sistemas intermedios.

4. Conclusiones

La investigación subyacente a esta publicación remarca sobremanera la poca utilización de los recursos disponibles en el kernel de linux a la hora de la utilización en los métodos subversibos y de troyanización de el núcleo del sistema operativo. La técnica explicada en el presente paper combina la ocultación de las actividades de red ilícitas con el acceso, manipulación e inyección de nuevo tráfico de red, permitiendo otro nuevo aspecto no visto hasta ahora en este tipo de técnicas.

Referencias

[1] bioforge. Hacking the linux kernel network stack. *Phrack N61*, 2003.

[2] Jonathan P. Clemens. An email interview with creed, the author of knark.

[3] devik ¡devik@cdi.cz¿. Linux on-the-fly kernel patching without lkm. *Phrack Magazine, N.58 0x07*, 2001.

[4] Greg Hoglund and Jamie Butler. *Subverting the Windows Kernel*. Adison Wesley, 2005.

[13] Socket Kernel Buffer, estructura de memoria de el kernel de linux, que transmite un paquete de red.

[14] Códigos de control de errores utilizados en los paquetes de datos para verificar si ha sufrido alteraciones en el transcurso de la transmisión.

[5] kossak. Building into the linux network layer. *Phrack N55*, 1999. Disponible en: http://www.phrack.org/archives/55/P55-12.

[6] R. Kuster. Three ways to inject your code into another process. 2003.

[7] Jose Ignacio Sáchez Martín. Ténicas de modelado de llamadas al sistema como aproximación a la detección de intrusiones en sistemas gnu/linux, microsoft windows 2000/xp y microsoft windows mobile. 2006.

[8] Jon Ander Ortiz. Intercepción de tráfico en sistemas gnu/linux. *Semana ESIDE*, 2006.

[9] plagez. Weakening the linux kernel. *Phrack, 8(52)*, 1998.

[10] RaiSe. Lkm rootkits en linux x86 v2.6. *enye-sec*, 2005.

[11] Rusty Russell. Linux netfilter hacking howto. 2003.

[12] Joanna Rutkowska. The implementation of passive covert channels in the linux kernel. *Chaos Communication Congress*, 2004.

[13] Joanna Rutkowska. Linux kernel backdoors and their detection. *ITUnderground Conference*, 2004. Disponible en: http://invisiblethings.org/papers/.

[14] Pablo Garaizar Sagarminaga. Linux 2.6 kernel hacking. *hackAndalus*, 2004.

[15] Querty Sandman. Pe infection under w32. *29a ezine*, 1996.

[16] stealth. Kernel rootkit experiences. *Phrack Magazine, N.61 0x03*, 2003.

[17] Dino Dai Zovi. Kernel rootkits. 2001.