

Rootkits basados en hardware (casi) indetectables

ROOTKITS DE CUARTA GENERACIÓN

Revisamos la historia de los rootkits, incluyendo su encarnación más novedosa, el DR RootKit.

POR KURT SEIFRIED

Originalmente intenté escribir un artículo sobre el estado actual de los rootkits y de las herramientas que podrían usarse para detectarlos. Pero tropecé con un pequeño problema: los rootkits más modernos suelen ser realmente buenos evitando la detección. Por realmente buenos quiero decir que es poco probable detectarlos a menos que tomemos alguna medida, como puede ser un análisis detallado del volcado de memoria del sistema, por ejemplo, comparando la imagen del kernel actual con la esperada.

Lección de Historia

Los rootkits tradicionales eran programas relativamente simples, que se ejecutaban a menudo como un demonio independiente proporcionando acceso a una puerta trasera. Generalmente eran fáciles de detectar mediante la búsqueda de nuevos procesos o de software recién instalado, lo que le permitía a los atacantes comenzar a subvertir siste-

mas binarios. Sucesivamente, esto permitió a los atacantes instalar sistemas binarios modificados, como versiones hackeadas de OpenSSH que tienen un nombre de usuario administrativo programado y contraseña para obtener acceso a nivel de root. Con la llegada de herramientas como Tripwire y el incremento del uso común de administradores de paquetes que pueden verificar la integridad de los ficheros instalados, tales como RPM y dpkg, fue posible detectarlos fácilmente [1].

Rootkits Basados en Kernel

Los atacantes pronto se dieron cuenta que se necesitaban más métodos de subversión y de ocultación sofisticados, lo cual condujo a los rootkits basados en kernel. Modificando la tabla de llamada del sistema, un atacante puede evitar fácilmente la detección porque, dicho de manera simple, controlan lo que estamos viendo y cómo se están ejecutando nuestros programas.

Habitualmente los atacantes usan un par de métodos para modificar el kernel del sistema: o cargando un módulo de kernel malicioso (por ejemplo, *heroin*), o parcheando la memoria del kernel escribiendo al dispositivo especial `/dev/kmem` (por ejemplo, *SucKIT*). Como estos ataques residen solamente en la memoria, su desventaja es que no sobreviven habitualmente a un reinicio.

Aunque son difíciles de detectar, estos rootkits pueden encontrarse comparando la tabla de llamada del sistema actual con la esperada (por ejemplo, examinando el fichero *System.map*). Los volcados de memoria del sistema pueden tomarse y usarse para verificar que el kernel en la memoria es correcto.

Por tanto, ¿Qué es lo que hacen los atacantes? Siga leyendo.

Rootkits Basados en Hardware y SO Virtualizados

Lanzado en el 2006 en Black Hat en Vegas, el primer rootkit basado en hardware hecho público se llamó "Blue Pill" [2]. Las CPUs modernas, desde AMD a Intel, incluyen un buen número de funcionalidades que soportan virtualización de sistemas operativos. Como ya no necesitan modificar el sistema operativo para funcionar, estos rootkits son más difíciles de detectar, así que chequear nuestra tabla de llamada del sistema operativo ya no funciona. Sin embargo, reemplazan el Interrupt Descriptor Table (IDT), guardado dentro de un registro de la CPU (el IDTR) [3].

Como ahora existen dos IDTRs (el real y otro que se presenta al sistema operativo comprometido), el que se presenta al sistema operativo comprometido se situará en una posición de memoria distinta a la habitual. Afortunadamente, la instrucción privilegiada Store Interrupt Descriptor Table (SIDT) puede ejecu-

tarse desde el espacio de usuario y devolverá formalmente los contenidos del IDTR al presentarse al sistema operativo (lo cual no es muy útil porque ha estado comprometido) y, lo que es más importante, la posición de memoria (que no estará en su posición normal).

Esto parece ser un empate: Los atacantes han creado nuevos métodos para ocultar rootkits, y los defensores han encontrado formas de detectarlos.

Nueva Generación de Rootkits

Lanzado en Septiembre de 2008 por Immunity Inc., DR RootKit [4] implementa la conexión de la llamada al sistema dentro del kernel Linux 2.6 sin modificar la tabla de llamada del sistema o la tabla descriptiva del interruptor. Para hacerlo, coloca un interruptor hardware en un gestor de llamada del sistema. Esta trampa sitúa un observador de memoria en la entrada `__NR_syscall` de `syscall_table`, lo cual se usa para exportar números de llamada del sistema. Básicamente, el rootkit se comporta como una herramienta de depuración, esperando que se ejecuten llamadas específicas del sistema, que en el momento que las ve, las modifica al vuelo. Actualmente, conecta las llamadas del sistema listadas en la Tabla 1.

DR RootKit incluye habilidades tales como ocultar procesos y evitar que finalicen procesos ocultos (lo que se traduce en que el atacante puede ejecutar software que está oculto y que no podemos matar ni incluso aunque consigamos adivinar el ID del proceso). Usando los ejemplos que vienen en el paquete, es relativamente fácil extender y crear llamadas del sistema modificadas adicionales. Por ejemplo, desearemos modificar el `capset` de manera que podamos configurar las habilidades del proceso a voluntad. El rootkit en sí mismo es un módulo del kernel que puede cargarse, lo que hace que sea fácil

de insertar una vez que hemos comprometido al sistema, aunque como otros rootkits basados en memoria, al reiniciar, el sistema lo eliminará de la memoria.

Si deseamos ampliar el rootkit deberemos insertar nuestras llamadas del sistema personalizadas. Como por ejemplo al reemplazar la salida de la llamada del sistema. Dicho de manera simple, el proceso consiste en declarar nuestro propio conector para sustituir un `syscall` del sistema y escribir a continuación una implementación de la llamada del sistema personalizada, algo realmente simple. El mejor sitio para comenzar es la fuente del kernel, para ser exactos, el subdirectorio `/kernel`, donde se definen la mayoría de las llamadas comunes del sistema. Por ejemplo, si tenemos un sistema en el que se usan las capacidades para restringir lo que pueden y lo que no pueden hacer los programas, podemos modificar la llamada del sistema `do_sys_capset_other_tasks` a otra modificada `cap_set_all`, la cual siempre devuelve todas las capacidades para un ID de proceso específico, como:

```
@@ -237,6 +237,9 @@
if (!capable(CAP_SETPCAP))
return -EPERM;
+ if (pid == 12345)
/* magic process number*/
+ return cap_set_all_evil
(effective, inheritable,
permitted);
```

Como puede verse, incluso una pequeña modificación puede tener un efecto significativo. De repente, el proceso con el ID 12345 dispondrá siempre de todas las capacidades, permitiéndole hacer todo cuanto desee. Con una sola llamada del sistema, un atacante puede crear una puerta trasera eficaz.

A todos los efectos, la única manera de detectar este rootkit es mediante la medida de la sincronización o condiciones de carrera introducidas por él. Si un rootkit está presente, el sistema debe ejecutarse un poco más lentamente de lo habitual, aunque medirlo de manera fidedigna no es una tarea fácil, especialmente en sistemas de producción. Además, el software es bastante simple y puede expandirse fácilmente para ocultarse mejor, haciendo que la detección sea incluso más difícil.

Método Alternativo

Evidentemente, podemos comprometer un sistema y conservar el acceso por otras vías mientras permanece oculto. Otra empresa de

prueba de participación de software llamada Core Security [5] ha adoptado el método de inyectar código hostil en el proceso que ha sido atacado. Por ejemplo, si explotamos un servidor http Apache, podemos inyectar código al proceso que nos permitirá tener acceso remoto. Esta técnica es un tanto limitada comparada con el rootkit basado en hardware o kernel completo, y es menos probable que afecte al sistema entero, haciéndolo más sigiloso. La primera desventaja de esta técnica es que mecanismos de protección a nivel del sistema operativo, tales como SELinux, aún podrán reforzar la política de seguridad. Sin embargo, para atacantes específicos, esto es a menudo un problema poco serio, ya que pueden usar exploits locales para comprometer más el sistema o permanecer dentro de las predisposiciones de una política SELinux y extraer información o usar el sistema para ejecutar otros ataques maliciosos.

Conclusión

Lo bueno es que a nivel de hardware los atacantes (conceptualmente) se quedan sin sitio adonde ir. Lo malo es que pueden usarse tantos trucos hardware como se deseen para mantener el control de un sistema comprometido. Por ejemplo, las tarjetas gráficas modernas tienen típicamente acceso de memoria directo (lo que significa que pueden hacer todo cuanto deseen a la memoria del sistema sin que el sistema operativo tenga mucho que decir en el proceso), su propia memoria de a bordo, y una gran cantidad de potencia de procesamiento (hasta el punto que la gente los está usando en clusters de ordenadores baratos). Las tarjetas más nuevas tienen memoria flash que pueden actualizarse desde el software, y no me queda duda de que algún día la gente averiguará cómo usar su tarjeta de vídeo para mantener el control de un sistema comprometido. ■

Tabla 1: Llamadas del Sistema

<code>getdents64</code>	Lee entradas al directorio
<code>getdents</code>	Lee entradas al directorio
<code>chdir</code>	Cambia el directorio de trabajo
<code>open</code>	Abre un archivo o dispositivo
<code>execve</code>	Ejecuta un programa
<code>socketcall</code>	Enlace a llamadas del sistema
<code>fork</code>	Crea un proceso hijo
<code>exit</code>	Finaliza el proceso actual
<code>kill</code>	Envía señal a un proceso
<code>getpriority</code>	Consigue prioridad del scheduling del programa

RECURSOS

- [1] "Pasadizos Secretos", de Amir Alsbih. Linux Magazine edición en Castellano, nº 29: <http://www.linux-magazine.es/issues/29>
- [2] Blue Pill: <http://bluepillproject.org/>
- [3] Red Pill: <http://www.invisiblethings.org/papers/redpill.html>
- [4] DR RootKit: <http://www.immunityinc.com/resources-freesoftware.shtml>
- [5] Core Security Technologies: <http://www.coresecurity.com/>