Virus en Linux

Un nuevo campo de batalla

Pablo Garaizar Sagarminaga
txipinet@txipinet.com

El mundo de los virus ha acompañado a la microinformática desde sus inicios. Si bien las plataformas más atacadas han sido las de Microsoft, ha habido incursiones en otros sistemas por parte de los escritores de virus. Linux ha permanecido unos años al margen, con unos pocos virus que eran más una prueba de concepto que una amenaza real... quizá esto no haya hecho más que empezar.

Breve introducción a Linux

Linux es un SO similar a UNIX, con una breve pero intensa historia. No quiero hablaros de Linus Torvalds y de toda la poesía que rodea a Linux. Para ello visitad www.linux.org ó www.fsf.org para tener un enfoque más global en cuanto al Software Libre. Centrándonos en el tema técnico, diremos que Linux es compatible con la mayoría de software desarrollado para UNIX comerciales, tiene un kernel monolítico con módulos, y utiliza mayoritariamente ejecutables de tipo ELF o scripts de shell, Perl, etc. Actualmente es una alternativa seria a los productos de Microsoft o Sun, tanto en entornos de escritorio como en servidores de producción, aunque quedan muchas cosas por mejorar.

Pero bueno, esto sólo pretendía situar en el tema a quienes no tuvieráis muy claro que es Linux, así que vamos a entrar en materia...

¿Qué es un virus?

Un virus informático es:

Computer Viruses

Consider the set of programs which produce one or more programs as output. For any pair of programs **p** and **q**, **p** *eventually produces* **q** if and only if **p** produces **q** either directly or through a series of steps (the "eventually produces" relation is the transitive closure of the "produces" relation.) A *viral set* is a maximal set of programs **V** such that for every pair of programs **p** and **q** in **V**, **p** eventually produces **q**, and **q** eventually produces **p**. ("Maximal" here means that there is no program **r** not in the set that could be added to the set and have the set still satisfy the conditions.) For the purposes of this paper, a *computer virus* is a viral set; a program **p** is said to be an instance of, or to be infected with, a virus **V** precisely when **p** is a member of the viral set **V**. A program is said to be *infected* simpliciter when there is some viral set **V** of which it is a member. A program which is an instance of some virus is said to *spread* whenever it produces another instance of that virus. The simplest virus is a viral set that contains exactly one program, where that program simply produces itself. Larger sets represent polymorphic viruses, which have a number of different possible forms, all of which eventually produce all the others.

Figura 1. Definición formal de virus informático.

Bien, esta es la definición formal basada en los estudios de Fred Cohen, el inventor del término "virus informático", que hace las delicias de los matemáticos pero que sólo está aquí para asustar. Afortunadamente Cohen da otra definición más asequible de virus informático: "un virus es un programa que es capaz de infectar otros programas modificándolos para que incluyan una copia, quizá evolucionada, de él mismo". Es decir, un virus lo que hace es tratar de añadir una copia de su propio código en otros programas.

Si os fijáis nadie ha dicho nada de discos duros destrozados, accesos ilícitos a ordenadores o ficheros secretos, etc. Más adelante veremos lo que es el "payload" de un virus, pero no tiene relación con la esencia de los virus en sí.

¿Qué NO es un virus?

Muchas veces oímos por la calle:

- "Jo, le voy a meter un virus a mi novia para espiarle lo que hace por Internet"
- "¿Si? Yo necesito un virus para que el día de la entrega de notas estalle el ordenador central de la Universidad"
- "Lo malo es que nunca seremos capaces de programar un virus tan potente como el Melissa o el ILoveYou..."

Existe una creencia popular que identifica "virus" con "TODO lo que hace daño al ordenador o a su seguridad". Eso es totalmente falso.

En el primero de los casos, lo que nuestro desconfiado amigo quiere es un programa espía, que normalmente suele presentarse en el formato de troyano o "caballo de Troya". Un troyano es un programa que hace otra cosa distinta a la que se supone que hace. Si nosotros troyanizamos un programa que muestra una felicitación de cumpleaños y le incluímos un programa espía que nos informará de todo lo que hace, cuando mandemos esa felicitación y su receptor la abra, se ejecutará la felicitación y el programa espía, de manera análoga a lo que sucedió con el regalo del caballo de Troya.

En el segundo caso estamos hablando de una "bomba lógica". En este tipo de programas hay una condición de activación ("fecha de la entrega de notas") y un efecto ("estallar el ordenador central de la Universidad"). No es necesario infectar ficheros. Sólo conseguir "instalar la bomba lógica".

El tercer caso habla de "afamados virus de ordenador" de los que se han hablado en los medios de comunicación. En ambos casos se trataba de "gusanos" o "worms" que en lugar de infectar ficheros, lo que infectaban era ordenadores. Es decir, un gusano es similar a un virus pero en lugar de infectar ficheros se propaga de ordenador en ordenador. En cuanto a la calidad de los citados gusanos... es como si a alguien todavía le extraña por qué un músico de conservatorio nunca venderá tantos discos en su vida como David Bisbal, así es este mundo :-D

Muchas veces virus, troyanos, bombas lógicas y gusanos utilizan técnicas de unos y otros entremezcladas, por lo que es difícil establecer fronteras fijas.

Bacterias, programas con vida

El término "virus informático" fue acuñado por Fred Cohen a comienzos de los 80. En mi opinión, no es el nombre más adecuado para este tipo de programas. Un virus biológico destruye las células por las que pasa, monopolizando todo su trabajo vital para su provecho, hasta que termina por matar al organismo. Un virus de computadora puede "convivir" durante años con un sistema de producción, y los buenos virus hacen todo lo posible para no interrumpir la funcionalidad de sus programas "huéspedes".

Yo prefiero llamarlos bacterias, o programas con vida. Quizá esta denominación deje clara mi postura ante los virus de ordenador, y quizá alguno de vosotros estará tirándose de los pelos diciendo: ¿o sea que ahora me dices que los virus de ordenador no son malos? ¿estás loco? Hay un debate abierto sobre la posibilidad de crear virus benévolos y no me gustaría alargarme en este tema, pero si alguien tiene curiosidad sobre ello, ya proporcionaré bibliografía y referencias acerca de ello.

Repaso de términos relacionados con los virus

Payload

El payload es el efecto de un virus en el sistema. Los virus normalmente se hacen famosos por sus payloads más que por sus métodos de infección. Mucha gente cree que un virus es más o menos potente en función de su efecto destructivo. Esto es absolutamente falso. Un virus mal programado, con un montón de bugs en su código y un payload absurdamente destructivo podrá salir en los titulares del telediario, pero desde un punto de vista académico o técnico no es más que un aporreo de teclas por parte de algún programador frustrado.

Marca de infección

Los virus, por lo general, necesitan saber si un fichero ha sido infectado ya para no infectar una y otra vez. Aquí puede residir el punto flaco de un virus en cuanto a su supervivencia: si se define una marca de infección muy específica, esa será la llave para que un antivirus lo detecte con precisión.

Existen virus famosos que no tenían marca de infección, como el Jerusalem, que infectaba una y otra vez los ficheros y provocaba el colapso en los discos duros.

Actualmente se suelen utilizar marcas de infección muy sutiles, que puedan cumplir ficheros que no estén realmente infectados, para provocar falsos positivos en los antivirus. Si, por ejemplo, definimos como marca de infección que el tamaño del fichero sea multiplo de 144, no infectaremos a todos los ficheros del sistema, pero un antivirus no podrá detectar el virus usando esa marca de infección, porque habrá muchos ficheros sin infectar que tengan un tamaño múltiplo de 144.

Cavity

Los ficheros y las páginas o segmentos de memoria se guardan en bloques que suelen ser múltiplos de una cantidad fija de bytes (típicamente 4 KB). Cuando la información no ocupa exactamente un múltiplo de ese tamaño de bloque, se hace un rellenado a ceros o "padding" y ese espacio se marca como inútil.

Un virus podrá hacer uso de este espacio sin alterar el tamaño en disco del fichero en el que se aloja. Si cabe en ese "recoveco" cumplirá dos objetivos: pasar desapercibido y meterse dentro del fichero sin corromper ningún dato original.

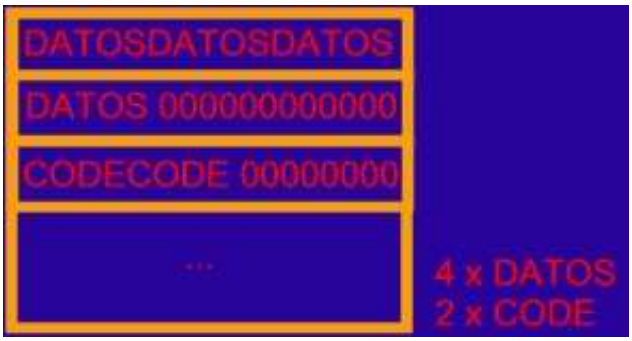

Figura 2. Padding en los segmentos de memoria.

Técnicas

Overwrite (sobreescritura)

El método de sobreescritura es el más obvio y el más sencillo de realizar, pero también el más burdo. Consiste simplemente en sobreescribir el host con el código del virus.

Como habréis adivinado, el huesped queda inutilizado, por lo que ocultar la presencia del virus es casi imposible. Además el método de infección se convierte en su payload. Un "buen" virus (o mal virus, según se mire) permite que el huesped siga funcionando. Este tipo de infección sólo la utiliza gente muy vaga o con pocos conocimientos.

Un símil en shell script podría ser el siguiente:

```
$> cat virus > host
```

Figura 3. Un virus de sobreescritura.

Prepending (copia al principio)

El siguiente paso lógico es que el virus se copie justo antes que el código del huesped, para asegurarse que el código vírico se va a ejecutar y no interferir en el funcionamiento del huesped.

Este método tiene un inconveniente: es costoso en cuanto a tiempo, y además, variable. Imaginémonos que el virus quiere instalarse en un huesped que es un ejecutable de 500 K's de tamaño: para ello deberá almacenar el equivalente al tamaño del virus en un buffer, luego copiarse al principio, e ir repitiendo este proceso de desplazamiento del código huesped hasta el final. También se podría crear un fichero temporal

en el que se copiaría el código del virus y luego concatenar el huesped, para, por último, mover el fichero temporal sobre el fichero que contenía al huesped... Ambas alternativas son costosas en cuanto a tiempo y Entrada/Salida, por lo que esta técnica tampoco es muy eficiente.

Siguiendo con los ejemplos en shell script, el método de copia al principio (prepending) podría entenderse de la siguiente manera:

```
$> cat virus > tmp
$> cat host >> tmp
$> mv tmp host
```

Figura 4. Proceso de copia al principio.

Postpending (copia al final)

Una vez vistas las limitaciones del método anterior, vamos a ver cómo se las ingenian los virus para que el efecto sea el mismo pero reduciendo el tiempo y la Entrada/Salida. La idea es relativamente sencilla: el virus se copia al del huesped y se pones un salto desde el principio del huesped, hasta el principio del código vírico, y cuando termine el virus, otro salto hasta el comienzo del huesped... ¿me seguís? Veamos:

- Situacion inicial:

```
Virus, Huesped (separados)
```

- Situacion final:

```
[saltar a Virus]Huesped[Fin]Virus[saltar a Huesped]
```

La ejecución sería:

1. Saltamos a Virus
2. Virus
3. Saltamos a Huesped
4. Huesped
5. Final

Figura 5. Proceso de copia al final.

Companion

Un companion virus, como su propio nombre indica, lo que hace es acompañar al fichero "huesped", sin modificarlo. Lo que se suele hacer es mover el fichero original a otro (generalmente oculto) y escribir el código vírico en un fichero con el nombre del original.

Este tipo de infección tiene el inconveniente de que si alguien intenta mover o copiar el fichero original, todo el efecto se perderá, ya que se rompera la "compañía" y el código vírico difícilmente encontrará al "huesped" original.

Multipartite (multiplataforma)

Los virus multiplataforma tienen como peculiaridad la posibilidad de infectar diferentes plataformas. Los casos más espectaculares de este tipo de virus pueden infectar diferentes microprocesadores, haciendo verdaderas virguerías en ensamblador. Normalmente suele hablarse de un virus multipartite cuando infecta ficheros y el sector de arranque, por ejemplo.

Técnicas anti-bait

Muchos antivirus crean ficheros "cebo" ("bait" o "goat files") para ver si hay un virus en el sistema y comprobar cómo ha cambiado ese fichero. Las técnicas anti-bait pretenden detectar esos ficheros y no infectarlos, saltarse la trampa.

Resindencia (+ per-process)

Para poder extenderse por el sistema, un virus puede infectar fichero a fichero o directorio a directorio, pero este método es bastante lento. Una alternativa a este método de diseminación es permanecer residente e infectar los ficheros a los que se vaya accediendo.

Esto en MSDOS o las primeras versiones de Mac era algo muy fácil de hacer (TSRs), pero conforme los sistemas se han ido complicando, las protecciones han sido cada vez más serias y ahora es bastante difícil conseguir una residencia en RING-0 (el nivel más alto de privilegios dentro del procesador) y lo que se utiliza bastante es la residencia en RING-3 (nivel de usuario) "per-process". Esta técnica la inventó Jacky Qwerty, ex-29a, y consiste en parchear una determinada API (por ejemplo CreateProcess) y asi suplantar todas las llamadas a esa API por parte de los procesos hijos que se crean a partir del ejecutable infectado. Este metodo de infeccion es bastante ingenioso, ya que con pocos privilegios se pueden infectar muchos ficheros.

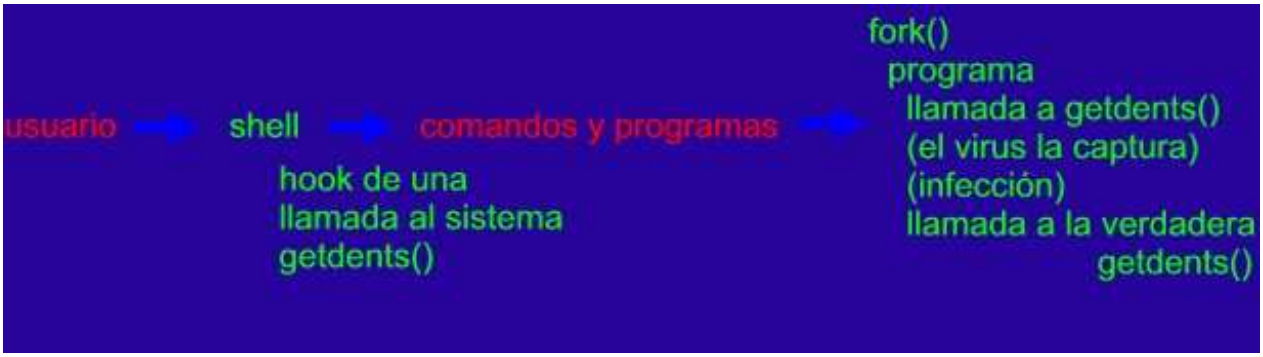

Figura 6. Residencia "per-process".

Encriptación (cifrado)

Los primeros antivirus fueron analizadores de cadenas de bytes, los virus tenían siempre el mismo código, y aislando la parte caracterísitca del código del virus, podía detectarse fácilmante con una comprobación (por ejemplo: "Si el fichero contiene 'I Love You, by megah4x0r' entonces ILoveYouDetectado;"). Para tratar de ocultar el código vírico los escritores de virus dividieron el código en dos: una pequeña rutina de desencriptado, y el resto del código vírico encriptado con ese mismo algoritmo.

Figura 7. Virus encriptado.

Poly (polimorfismo)

La anterior técnica oculta el contenido real del virus en cuanto a código, pero sigue generando virus idénticos generación tras generación. Si se convina con el polimorfismo, el virus puede convertirse en una auténtica pesadilla para las casas de antivirus. La idea es que la rutina de encriptado y desencriptado será diferente cada vez, y la clave para encriptar el código vírico será variable. Así, con muchas rutinas de encriptado e infinitas claves posibles, el código del virus va mutando asombrosamente en cada infección, y detectarlo es mucho más difícil.

Para detectar este tipo de virus, en lugar del análisis de cadenas de bytes, se suelen emplear simuladores, que tratan de simular la ejecución del virus para ver si, una vez desencriptado, se trata realmente de un virus.

EPO (Entry Point Obscuring)

Cuando un virus infecta un ejecutable, es bastante normal que cambie el Entry Point o puntero a la dirección de memoria que contiene el comienzo del código ejecutable. Antes, al explicar la infección postpending o de copia al final, ocurría esto: el virus cambia el puntero que apunta al comienzo del código por un puntero que apunta al comienzo de SU código, para asegurarse que será lo primero que se ejecute.

Este método es sencillo, pero muy fácilmente detectable por un antivirus. GriYo, de 29a, ideó una técnica para hacer esto menos transparente que consiste en ocultar el salto al código vírico dentro del código del fichero infectado. Es decir, el virus deja que el fichero infectado se ejecute normalmente durante unas instrucciones para que el antivirus no lo detecte, y poco después, lanza su código.

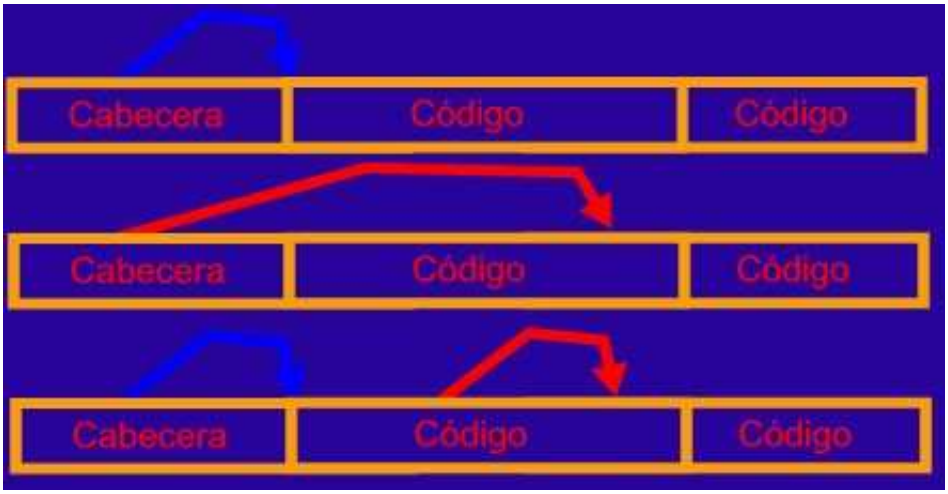

Figura 8. Entry Point Obscuring.

En Linux no hay virus... ¡Falso! Objetivos y técnicas:

Scripts: sh, Perl

Los lenguajes interpretados han sido una constante en todo sistema UNIX. Actualmente los más utilizados son los scripts de shell y Perl. Programar un virus en estos lenguajes es un juego de niños, aquí tenemos un ejemplo de un virus de shell sencillo:

```
#!/bin/sh
for FICHERO in *
do
  tail -4 $0 >> $FICHERO
done
```

¿Qué hace? Va copiando sus 4 últimas líneas (tail -4 $0) al final de cada fichero en este directorio (">>" es append, o añadir al final). Como podemos ver, es un virus bastante tonto, infecta tanto ejecutables como ficheros de datos, y puede dejarlos inutilizados, pero con unas pocas decenas de líneas más, podría hacerse algo más presentable.

La sencillez de estos virus es su ventaja para sus programadores, pero también su debilidad: son tremendamente fáciles de detectar a simple vista, engordan el tamaño del fichero infectado considerablemente y realentizan su ejecución más allá de lo inperceptible por un usuario normal.

Binarios

a.out

Es un formato realmente simple, casi tanto como los COM de DOS. Actualmente este tipo de ejecutables está en desuso, pero todavía quedan sistemas con a.out's (a pesar de que el compilador genere un fichero llamado "a.out", eso no implica que tenga este formato, casi con seguridad se tratará de un ELF).

Estructura de un ejecutable a.out:

Figura 9. Formato de un a.out.

Infeccion de ejecutables a.out:

Se puede optar por aumentar el tamaño de la sección de código (.text) y desplazar el resto del archivo, o tratar de encontrar una cavidad (cavity) para instalar el virus allí. El virus, además, deberá modificar la cabecera para reflejar los cambios (diferente tamaño de secciones, diferente entry point...).

ELF

El formato ELF es el más utilizado hoy en día en los ejecutables para UNIX. Es un formato muy flexible y bastante bien diseñado.

Estructura de un ELF:

Figura 10. Formato de un ELF.

Infeccion de ejecutables ELF:

Las investigaciones más serias en este campo vienen de la mano de Silvio Cesare. Ha publicado ya numerosos artículos acerca de este tema, y todos sus virus han sido programados en C, para poder ser compilados en cualquier sistema UNIX. El método que utiliza es el siguiente:

- Incrementar un campo en la cabecera del ELF (p_shoff) que indica el desplazamiento u offset donde se encuentra la tabla de cabecera de secciones (Section header table).
- Hallar la cabecera de programa del segmento de codigo y:
  - Incrementar la variable que indica el tamaño que ocupa el código físicamente (p_filesz).
  - Incrementar la variable que indica el tamaño que ocupa el código cuando se carga en memoria (p_memsz).
- Para cada cabecera de programa cuyo segmento esta despues del de codigo (que es donde hemos introducido el virus):
  - Incrementar el offset del segmento en el fichero (p_offset).
- Para cada cabecera de seccion cuya seccion este despues de nuestra insercion:
  - Incrementar sh_offset, para tener en cuenta el nuevo codigo
- Insertar el virus en si en el fichero

Esto puede parecer un lío para más de uno, en simples palabras lo que se trata es de hacer el tamaño del segmento de código más grande para hacer espacio para el virus. Luego hay que actualizar todos los valores para que el código nuevo se cargue, y cambiar el entry point para que apunte al virus.

Figura 11. Infección de un ELF por el método de Silvio Cesare.

Este método de infección funciona perfectamente, pero no es el único. Wintermute presento en el hackmeeting de 2000 un nuevo virus para Linux, el Lotek, que realizaba una infección aprovechando una cavidad (cavity) en la sección ".note".

Recientemente el ex-VXer Bumblebee ha publicado un virus para Linux con residencia per-process en RING-3 y unas cuantas técnicas aprendidas en entornos win32. Muchas de las estructuras de win32 tienen su paralelismo en Linux, por lo que gran cantidad de técnicas pueden portarse fácilmente a los virus de Linux.

Ficheros de fuentes

Hay algunos intentos de infectar ficheros fuente en lugar de binarios. Se puede realizar un enfoque desde el punto de vista de ensamblador "inline" o embebido dentro del código, o bien un ejecutable que genere fuente como salida.

Packages: .deb, .rpm, .mdk

Un punto todavía poco explotado es el de los paquetes de software de las diferentes distribuciones de Linux. Mucha gente utiliza paquetes para instalar programas de manera sencilla y ordenada, y en ocasiones esos paquetes son descargados por un usuario sin privilegios desde un navegador, para ser instalados posteriormente por "root". En ese intervalo de tiempo en el que permanecen en el directorio del usuario sin privilegios podrían ser infectados y luego, al ser instalados por "root", acceder a todo el sistema.

Un paquete generalmente tiene comprobaciones mediante MD5, pero pueden recalcularse, por lo que este puede ser un punto flaco importante en nuestras distribuciones Linux.

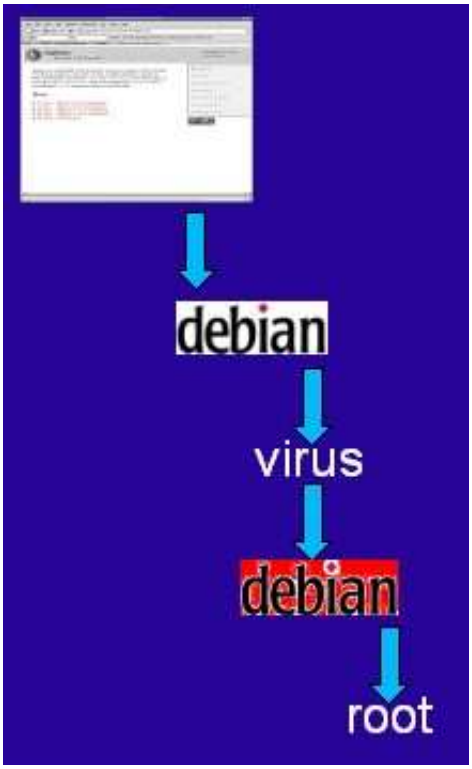

Figura 12. Infección de un paquete .deb tras ser descargado por un usuario desde su navegador.

Ya, pero Linux es seguro, ¿no?

Linux es seguro

Sí, Linux es bastante seguro. De hecho un virus deberá utilizar algun despiste de configuración en el sistema para poder colarse hasta tener acceso a todas sus partes.

Está claro que actualmente usar Linux es el mejor antivirus que existe. No he visto a nadie que haya sufrido un virus en Linux y eso que conozco a mucha gente que usa Linux masivamente. Es posible que con el tiempo esta situación vaya cambiando y Linux sea otro escenario donde se libren las batallas entre programadores de virus y de antivirus. Por el momento, salvo experimentos de laboratorio, estamos a salvo.

¿Cómo podemos hacer una escalada de privilegios?

Exploits

Un exploit es un programa que aprovecha un fallo en el sistema para conseguir algo no permitido de él. Si un virus incluye ese código dentro del suyo, podría conseguir acceder a zonas no permitidas y hacerse con el control del sistema.

Este enfoque ha sido utilizado en varios virus para Linux, como el staog por Quantum/VLAD, pero implica la extinción del virus en cuanto el fallo que explota el exploit sea subsanado. Algunos virus intentan aprovecharse del exploit, y si no tiene éxito, eliminan el código del exploit del resto de infecciones.

Este enfoque es más propio de entornos menos dinámicos en cuanto a correcciones de fallos en programas, como Windows (poca gente actualiza periódicamente su navegador o su editor de textos). En entornos de desarrollo open source los fallos suelen ser detectados y subsanados más dinámicamente.

LKMs (Loadable Kernel Modules)

Hemos dicho en la introducción que el núcleo de Linux es monolítico, pero tiene un sistema de carga y descarga de módulos que permiten un uso más eficiente de los controladores de dispositivos (drivers).

Un módulo del kernel (o LKM) se ejecuta en RING-0, dentro del espacio reservado para el kernel, es decir, tiene un poder total sobre la máquina. Es posible programar LKMs que tengan mas poder o que engañen a "root", por lo que si un virus lograse cargar un módulo dentro del módulo, podría ser una pesadilla para el administrador de la máquina, y el único límite de acción serían las limitaciones físicas de los dispositivos.

Con un LKM se puede hacer de todo: ocultar procesos, modificar tamaños de archivos, etc. por lo que puede que los futuros virus de Linux incluyan LKMs para sus propósitos.

Windows/Linux

Muchos de los ordenadores personales que utilizan los usuarios de Linux, aunque a veces cueste reconocerlo, tienen una partición con Windows. Existen varias herramientas para acceder a particiones Linux desde Windows, explore2fs quizá sea la más conocida.

Si un virus atacase un sistema Windows, consiguiese los privilegios suficientes como para acceder al disco duro y buscar un fichero clave dentro de la partición Linux, como pueda ser "init" (el proceso inicial del que se crean todos los demás procesos) o la shell que use "root", todas las protecciones de Linux como tal habrían sido inútiles. Aunque suene un poco fuerte: la inseguridad inherente de Windows actuaría como "Caballo de Troya" contra el sistema Linux.

Existe otra herramienta bastante utilizada, VMWare, que permite tener varias máquinas virtuales corriendo Sistemas Operativos diferentes. Es también muy común tener Windows y Linux funcionando al mismo tiempo con VMWare. En lugar de tener que esperar a que el sistema rearranque con Linux como en el caso anterior, la infección podría hacerse directamente, ya que VMWare es fácilmente detectable (utiliza un RING que no es ni 0 ni 3).

fork() y crack

Un virus desde una cuenta de usuario podría armarse de paciencia y crear un proceso con muy baja prioridad (para no interferir en el rendimiento normal del sistema) que intentase crackear las contraseñas por fuerza bruta.

Imaginemos un sistema automatizado, en el que el administrador entra sólo cada semana a retocar 4 cosas, pero no hay una supervisión real. Un virus podría colarse desde una cuenta sin privilegios, y estar un par de semanas intentando crackear las contraseñas. Una vez conseguido esto, sólo queda dar el salto a "root" y de ahí a donde quiera (kernel, otros ordenadores...).

Entonces... ¿por qué no hay (casi) virus para Linux?

Perfil del usuario medio

Actualmente el usuario medio de Linux tiene poco que ver con el usuario medio de Windows o Macintosh. Quizá mucha gente cayó con lo de "Enanito si, pero que pedazo de coj...", pero esto tendría poco éxito en un entorno de usuarios de Linux.

La gente acostumbra a conocer el origen de sus programas, y examina su código fuente. No quiero decir que todos los usuarios de Linux lo hagan, pero hay un grupo importante de gente que lo hace y lo comenta al resto.

Las llamadas "técnicas de ingeniería social", es decir, hacer uso de la candidez del usuario que recibe un virus o gusano, tienen muchas más dificultades con usuarios de Linux.

Filosofía del software

Como acabo de comentar, que Linux sea de código abierto y haya una mentalidad clara en cuanto a ese tema, dificulta ocultar código en los programas.

Ken Thompson dijo que ningún software creado por otro podía ser confiable, especialmente si había creado por él. Tal y como explicó en una conferencia en mitad de la década de los 80, Thompson había introducido un sistema de autoreplicación y "troyanización" en todo compilador C para UNIX que le permitió hasta entonces poder entrar como "root" en todo sistema hasta la fecha. Si alguien quiere saber cómo se las ingenió el bueno de Thompson, lo explicaré por email gustosamente, o bien esperáis al turno de preguntas O:-)

Pocos VXers linuxeros

Todavía hay pocos escritores de virus que usan Linux habitualmente. Algunos han instalado Linux en una partición para hacer sus pruebas y conocen bastantes cosas de él, pero no tienen en absoluto la soltura que han conseguido durante años de uso de DOS y Windows.

Supongo que esto cambiará con el tiempo, y pronto los VXers usarán Linux a diario. Cuando esto ocurra, yo creo que habrá una nueva hornada de virus para Linux programados desde la experiencia, no como un experimento de laboratorio.

¿Y qué puede pasar en el futuro?

- + Usuarios novatos
- + Empresas usan Linux -> - Open Source
- + Configuraciones "click&play" -> - Robustez del sistema
- + VXers linuxeros

Conclusión

Solución: una buena "salud" informática

Es decir:

- actualizar las versiones de nuestros programas para evitar bugs.
- conseguir los programas de fuentes fidedignas.
- utilizar siempre que sea posible la versión en código fuente de los programas.
- no ejecutar todo lo que nos llegue por Internet
- ...

Todo ese tipo de cosas que, como espero que haya quedado claro ;-), utilizan los virus para colarse en nuestros sistemas

Para saber más...

- http://www.coderz.net/29a: Página de 29a, el grupo de virus de más nivel de la viruscene actual.
- http://pagina.de/wintermute: Página de wintermute, gran VXer y amigo ;-)
- www.int80.org: Página de ensamblador en UNIX y BSD
- http://linuxassembly.org: Página de ensamblador en Linux

Este documento ha sido escrito por un miembro 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).