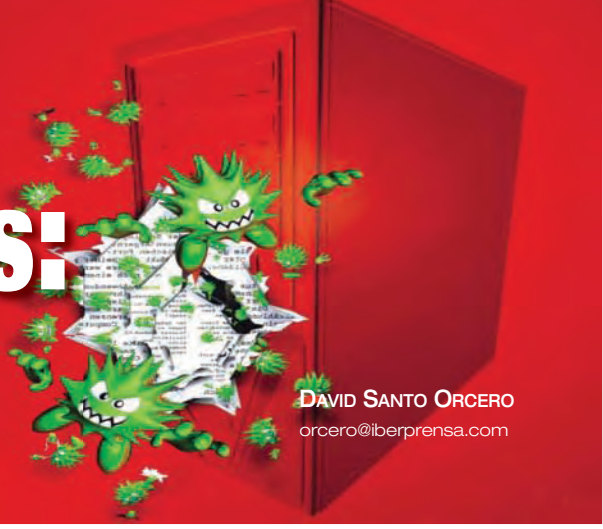
# Mitos y realidades: Linux y los virus



David Santo Orcero
orcero@iberprensa.com

**El debate sobre Linux y los virus no es algo nuevo. Cada cierto tiempo vemos un correo en alguna lista preguntando si existen virus para Linux; y automáticamente alguien responde afirmativamente y alega que si no son más populares es porque Linux no está tan extendido como Windows. También son frecuentes las notas de prensa de desarrolladores de antivirus diciendo que sacan versiones contra los virus de Linux.**

Personalmente he tenido alguna que otra discusión con distintas personas por correo, o por lista de distribución, respecto al tema de si existen o no los virus en Linux. Se trata de un mito, pero es complejo derribar un mito o, mejor dicho, un bulo, especialmente si está causado por interés económico. A alguien le interesa transmitir la idea de que si Linux no tiene este tipo de problemas, es porque muy poca gente lo utiliza.

A la hora de publicar este reportaje me hubiese gustado elaborar un texto definitivo sobre la existencia de virus en Linux. Desgraciadamente, cuando la superstición y el interés económico campan a sus anchas, es difícil construir algo definitivo. No obstante, intentaremos hacer aquí un argumentario razonablemente completo para desarmar los ataques de cualquiera que quiera discutirlo.

## QUÉ ES UN VIRUS

Lo primero, vamos a comenzar definiendo qué es un *virus*. Se trata de un programa que se copia y se ejecuta automáticamente, y que tiene por objeto alterar el normal funcionamiento de un ordenador, sin el permiso o el conocimiento del usuario. Para ello, los virus reemplazan archivos ejecutables por otros infectados con su código. La definición es estándar, y es un resumen de una línea de la entrada sobre virus que aparece en la Wikipedia.

La parte más importante de esta definición, y la que diferencia el virus del resto del *malware*, es que un virus se instala solo, sin el permiso o conocimiento del usuario. Si no se instala solo, no es un virus: podría ser un ser un *rootkit*, o un *troyano*.

Un *rootkit* es un parche al kernel que permite ocultar determinados procesos a las utilidades de área de usuario. Dicho de otra forma, es una modificación del código fuente del kernel que tiene como objeto que las utilidades que permiten ver qué se está ejecutando en cada momento no visualicen un determinado proceso, o un determinado usuario.

Un *troyano* es análogo: es una modificación al código fuente de un servicio concreto para ocultar determinada actividad fraudulenta. En ambos casos es necesario obtener el código fuente de la versión exacta instalada en la máquina Linux, parchear el código, recompilarlo, obtener privilegios de administrador, instalar el ejecutable parcheado, e inicializar el servicio –en el caso del troyano– o el sistema operativo completo –en el caso del rootkit–. El proceso, como vemos, no es trivial, y nadie puede hacer todo esto "por error". Tanto unos como otros exigen en su instalación que alguien con privilegios de administrador, de forma consciente, ejecute una serie de pasos tomando decisiones de índole técnica.

Lo cual no es un matiz semántico sin importancia: para que un virus se instale, basta con que ejecutemos un programa infectado como usuario común. Por otro lado, para la instalación de un rootkit o de un troyano es imprescindible que un humano malicioso entre personalmente en la cuenta de root de una máquina, y de forma no automatizada realice una serie de pasos que son potencialmente detectables. Un virus se propaga con rapidez y eficiencia; un rootkit o un troyano necesitan que vayan específicamente a por nosotros.

## LA TRANSMISIÓN DE LOS VIRUS EN LINUX

El mecanismo de transmisión de un virus, por lo tanto, es lo que realmente lo define como tal, y es la base de la existencia de los mismos. Un sistema operativo es más sensible a los virus cuanto más fácil sea desarrollar un mecanismo eficiente y automatizado de transmisión de estos.

Supongamos que tenemos un virus que quiere transmitirse solo. Supongamos que ha sido lanzado por un usuario normal, de forma inocente, al lanzar un programa. Dicho virus tiene exclusivamente dos mecanismos de transmisión:

- Replicarse tocando la memoria de otros procesos, anclándose a ellos en tiempo de ejecución.
- Abriendo los ejecutables del sistema de ficheros, y añadiendo su código –*payload*– al ejecutable.

Todos los virus que podemos considerar como tales tienen al menos uno de estos dos mecanismos de transmisión. O los dos. No hay más mecanismos.

Respecto al primer mecanismo, recordemos la arquitectura de memoria virtual de Linux y cómo funcionan los procesadores Intel. Estos poseen cuatro anillos, numerados de 0 a 3; a menor número, mayores los privilegios que tiene el código que se ejecute en dicho anillo. Estos anillos corresponden con estados del procesador, y, por lo tanto, con lo que se puede hacer con un sistema estando en un anillo concreto. Linux hace uso del anillo 0 para el kernel, y del anillo 3 para los procesos. No hay código de proceso que se ejecute en anillo 0, y no hay código de kernel que se ejecute en anillo 3. Solo hay un único punto de entrada al kernel desde el anillo 3: la interrupción 80h, que permite saltar del área donde está el código de usuario al área donde está el código de kernel.

**La arquitectura de Unix en general y de Linux en particular no hace factible la dispersión de un virus**

El kernel mediante el uso de la memoria virtual hace creer a cada proceso que tiene toda la memoria para él solo. Un proceso –que trabaja en anillo 3– solo puede ver la memoria virtual que le han configurado, por el anillo en el que opera. No es que la memoria de los otros procesos esté protegida; es que para un proceso la memoria de los otros está fuera del espacio de direcciones. Si un proceso diese una batida a todas las direcciones de memoria, no sería capaz ni de referenciar una dirección de memoria de otro proceso.

¿Por qué esto no se puede trampear? Para modificar lo comentado –por ejemplo, generar puntos de entrada en anillo 0, modificar los vectores de interrupciones, modificar la memoria virtual, modificar la LGDT...– solo es posible desde el anillo 0. Es decir, para que un proceso pudiese tocar la memoria de otros procesos o del kernel, debería ser el propio kernel. Y el hecho de que haya un único punto de entrada y que los parámetros se pasen por registros complica la trampa –de hecho, se pasa por registro hasta lo que se debe hacer, que se implementa luego como un *case* en la rutina de atención a la interrupción 80h–.

Otro escenario es el caso de sistemas operativos con cientos de llamadas no documentadas al anillo 0, donde esto sí es posible –siempre puede quedar una llamada olvidada mal implementada sobre la que se pueda desarrollar una trampa–, pero en caso de un sistema operativo con un mecanismo de paso tan simple, no lo es.

Por ello, la arquitectura de memoria virtual impide este mecanismo de transmisión; ningún proceso –ni siquiera los que tienen privilegios de root– tienen forma de acceder a la memoria de otros. Podríamos argumentar que un proceso puede ver el kernel; lo tiene mapeado a partir de su dirección de memoria lógica 0xC0000000. Pero, por el anillo del procesador en el que se ejecuta, no puede modificarlo; generaría un trap, ya que son zonas de memoria que pertenecen a otro anillo.

La "solución" sería un programa que modificara el código del kernel cuando es un fichero. Pero el hecho de que estos se recompilen, lo hace imposible. No se puede de parchear el binario, ya que hay millones de kernels binarios distintos en el mundo. Simplemente con que al recompilarlo le hubiesen puesto o quitado algo al ejecutable del kernel, o le hubiesen cambiado el tamaño de alguna de las etiquetas que identifican la versión de compilación –algo que se hace incluso involuntariamente– el parche binario no se podría aplicar. La alternativa sería descargar el código fuente de Internet, parchearlo, configurarlo para el hardware apropiado, compilarlo, instalarlo y reiniciar la máquina. Todo esto lo debería hacer un programa, de forma automática. Todo un reto para el campo de la Inteligencia Artificial.

Como vemos, ni siquiera un virus como root puede saltar esta barrera. La única solución que queda es la transmisión entre ficheros ejecutables. Lo que tampoco funciona como veremos a continuación.

## Mi experiencia como administrador ▶

**En más de diez años que llevo administrando Linux, con instalaciones en cientos de máquinas de centros de cálculo, laboratorios de alumnos, empresas, etc.**

- **Nunca me ha "entrado" un virus**
- **Nunca he conocido a alguien que le haya ocurrido**
- **Nunca he conocido a alguien que haya conocido a alguien que le haya ocurrido**

**Conozco a más gente que ha visto al monstruo del Lago Ness a que haya visto virus para Linux.**

**Personalmente, reconozco que he sido un temerario, y he lanzado varios programas que los autoprocramados "especialistas" denominan "virus para Linux" -en adelante, los denominaré virus, para no hacer pedante el texto-, desde mi cuenta habitual contra mi máquina, para ver si es posible un virus: tanto el virus bash que circula por ahí -y que, por cierto, no me infectó ningún fichero-, como un virus que se hizo muy famoso, y salió en la prensa. Intenté instalarmelo; y después de veinte minutos de trabajo, me rendí cuando vi que entre sus exigencias estaba tener el directorio *tmp* en una partición del tipo *MSDOS*. Personalmente, no conozco a nadie que cree una partición específica para *tmp* y la formatee en FAT.**

**De hecho, algunos supuestos virus que he probado para Linux necesitan un nivel de conocimientos altos y la clave de root para ser instalados. Podríamos calificar, cuanto menos, de "cutre" un virus si necesita nuestra intervención activa para que nos infecte la máquina. Además, en algún caso requieren amplios conocimientos de UNIX y la clave de root; lo que está bastante lejos de la instalación automática que se le supone.**

## INFECTANDO EJECUTABLES EN LINUX

En Linux, un proceso puede hacer simplemente lo que le permita su usuario efectivo y su grupo efectivo. Es cierto que existen mecanismos para intercambiar el usuario real con el efectivo, pero poco más. Si nos fijamos donde están los ejecutables, veremos que solamente root tiene privilegios de escritura tanto en dichos directorios, como en los ficheros contenidos. Dicho de otro modo, solamente root puede modificar dichos archivos. Esto es así en Unix desde los 70, en Linux desde sus orígenes, y en un sistema de ficheros que soporte privilegios, aún no ha aparecido ningún error que permita otro comportamiento. La estructura de los ficheros ejecutables ELF es conocida y está bien documentada, por lo que es técnicamente posible que un fichero de este tipo cargue el *payload* en otro fichero ELF... siempre que el usuario efectivo del primero o el grupo efectivo del primero tengan privilegios de lectura, escritura y ejecución sobre el segundo fichero. ¿Cuántos ejecutables del sistema de ficheros podría infectar como usuario común?

Esta pregunta tiene una respuesta simple, si queremos saber a cuántos ficheros podríamos "infectar", lanzamos el comando:

```
$ find / -type f -perm -o=rwx -o
 \( -perm -g=rwx -group `id -g`
\) -o \( -perm -u=rwx -user `id
-u` \) -print  2> /dev/null |
grep -v /proc
```

Excluimos el directorio */proc* porque es un sistema de ficheros virtual que muestra información sobre cómo funciona el sistema operativo. Los archivos de tipo fichero y con privilegios de ejecución que encontraremos no suponen un problema, ya que con frecuencia son enlaces virtuales que constan como que pueden leerse, escribirse y ejecutarse, y si un usuario lo intenta, nunca funciona. También descar-

tamos los errores, abundantes –ya que, especialmente en */proc* y en */home*, hay muchos directorios donde un usuario común no puede entrar–.

Este script tarda bastante. En nuestro caso particular, en una máquina donde trabajamos cuatro personas, la respuesta fue:

```
/tmp/.ICE-unix/dcop5265
-1205225188
/tmp/.ICE-unix/5279
/home/irbis/kradview-1.2/src
/kradview
```

La salida muestra tres ficheros que podrían infectarse si se ejecutase un hipotético virus. Los dos primeros son ficheros de tipo Unix socket que se borran en arranque –y que no pueden verse afectados por un virus–, y el tercero es un fichero de un programa en desarrollo, que cada vez que se recompila se borra. El virus, desde el punto de vista práctico, no se propagaría.

Por lo que vemos, la única forma de propagar el *payload* es siendo root. En ese caso para que un virus funcione es necesario que los usuarios tengan siempre privilegios de administrador. En ese caso sí puede infectar archivos. Pero aquí viene la trampa: para transmitir la infección, necesita tomar otro ejecutable, mandarlo por correo a otro usuario que solamente emplee la máquina como root, y que repita el proceso.

En sistemas operativos donde es necesario ser administrador para tareas comunes o para ejecutar muchas aplicaciones diarias, esto sí se puede dar. Pero en Unix es necesario ser administrador para configurar la máquina y modificar los archivos de configuración, así que es escaso el número de usuarios que emplea como cuenta diaria la de root. Es más; algunas distribuciones de Linux ni siquiera tienen habilitada la cuenta de root. En casi todas, si accedes como tal al entorno gráfico, el fondo se cambia a rojo intenso, y se repiten mensajes constantes que recuerdan que no se debe emplear esta cuenta. Finalmente, todo lo que se debe hacer como root es posible hacerlo mediante un comando *sudo* sin riesgo.

Por ello, en Linux un ejecutable no puede infectar a otros siempre que no estemos usando la cuenta de root como cuenta de uso común; y aunque las compañías antivirus se empeñan en decir que hay virus para Linux, realmente lo más parecido que se puede crear en Linux es un troyano en área de usuario. La única forma de que estos troyanos puedan afectar algo del sistema es ejecutándolo como root y con lo privilegios necesarios. Si solemos emplear la máquina como usuarios de a pie, no es posible que un proceso lanzado por un usuario común infecte al sistema.

## MITOS Y MENTIRAS

Encontramos una gran cantidad de mitos, bulos y simplemente mentiras sobre los virus en Linux. Hagamos una relación de los mismos basándonos en una discusión acontecida hace ya tiempo con un representante de un fabricante de antivirus para Linux que se ofendió mucho por un artículo publicado en esta misma revista. Aquella discusión es un buen ejemplo de referencia, ya que toca todos los aspectos sobre los virus en Linux. Vamos a repasar todos estos mitos uno a uno según se comentaron en aquella discusión concreta, pero que tantas veces se ha repetido en otros foros.

### Mito 1:

"No todos los programas malignos, particularmente los virus, necesitan privilegios de root para infectar, sobre todo en el caso particular de los virus ejecutables (formato ELF) que infectan otros ejecutables".

### Respuesta:

Quien realice semejante afirmación desconoce cómo funciona el sistema de privilegios de Unix. Para poder afectar a un fichero, un virus necesita el privilegio de lectura –hay que leerlo para modificarlo–, y de escritura –hay que escribirlo para que la modificación sea válida– sobre el fichero del ejecutable que quiere ejecutar. Esto es así siempre, sin excepciones. Y en todas y cada una de las distribuciones, los usuarios que no son root no disponen de estos privilegios. Luego simplemente con no ser root, la infección no es posible. Prueba empírica: En la sección anterior vimos un simple script para comprobar el rango de ficheros que pueden ser afectados por una infección. Si lo lanzamos en nuestra máquina, veremos como es ínfimo, y respecto a ficheros de sistema, nulo. Además, a diferencia de operativos como Windows, no es necesario tener privilegios de administrador para realizar tareas comunes con programas que emplean comúnmente usuarios normales.

### Mito 2:

"Tampoco necesitan ser root para entrar en el sistema por vía remota, es el caso de *Slapper* un gusano que explotando una vulnerabilidad en el SSL de Apache (los certificados que permiten comunicación segura), creó su propia red de máquinas zombie en septiembre de 2002".

### Respuesta:

Ese ejemplo no alude a un virus, sino un gusano. La diferencia es muy importante: un gusano es un programa que explota un servicio de cara a Internet para transmitirse. No afecta a programas locales. Por ello, solamente afecta a los servidores; no a máquinas particulares.

Los gusanos han sido siempre muy pocos y de incidencia ínfima. Los tres realmente importantes nacieron en los 80, una época en la que Internet era inocente, y todo el mundo se fiaba de todo el mundo. Recordemos que eran los que afectaban a sendmail, fingerd y rexec.

Hoy en día la cosa es más complicada. Aunque no podemos negar que sigue habiéndolos y que, si no se controlan, son extremadamente peligrosos. Pero ahora, los tiempos de reacción ante los gusanos son muy cortos. Es el caso del *Slapper*: un gusano creado sobre una vulnerabilidad descubierta –y parcheada– dos meses antes de la aparición del propio gusano. Aun suponiendo que todo el mundo que usara Linux tuviese Apache instalado y funcionando siempre, simplemente con actualizar mensualmente los paquetes hubiese sido más que suficiente para que nunca se corriera ningún riesgo.

Es cierto que el fallo de SSL que originó *Slapper* era crítico –de hecho, el mayor fallo encontrado en toda la historia de SSL2 y SSL3–, y como tal fue solucionado en pocas horas. Que dos meses más tarde de que se encontrara y se solucionara dicho problema, alguien hiciera un gusano sobre un error ya corregido, y que ese

### Antivirus para Linux ▶

**Es cierto que existen antivirus para Linux buenos. El problema es que no hacen lo que los defensores de los antivirus argumentan. Su función es filtrar el correo que pasa de malware y virus para Windows, así como verificar la existencia de virus de Windows en carpetas exportadas vía SAMBA; con lo que si empleamos nuestra máquina como gateway de correo o como NAS para máquinas Windows, podemos protegerlas.**

sea el ejemplo más potente que se puede dar como vulnerabilidad, cuando menos tranquiliza.

Como regla general, la solución a los gusanos no es comprar un antivirus, instalarlo y desperdiciar tiempo de cómputo manteniéndolo residente. La solución es hacer uso del sistema de actualizaciones de seguridad de nuestra distribución: teniendo la distribución actualizada, no habrá problemas. Ejecutar solamente los servicios que necesitamos es también una buena idea por dos razones: mejoramos el aprovechamiento de recursos, y evitamos problemas de seguridad.

**Mito 3:**

"No creo que el núcleo sea invulnerable. De hecho existe un grupo de programas malignos denominados con las siglas LRK (Linux Rootkits Kernel), que se basan precisamente en que explotan vulnerabilidades de módulos del kernel y sustituyen los binarios del sistema".

**Respuesta:**

Un rootkit es básicamente un parche al kernel que permite ocultar la existencia de determinados usuarios y procesos ante las herramientas habituales, gracias a que no aparecerán en el directorio */proc*. Lo normal es que lo utilicen al final de un ataque, en primer lugar, van a explotar una vulnerabilidad remota para tener acceso a nuestra máquina. Después emprenderán una secuencia de ataques, para hacer escalado de privilegios hasta llegar a tener la cuenta de root. El problema cuando lo consiguen es cómo instalar un servicio en nuestra máquina sin ser detectados: ahí entra el rootkit. Se crea un usuario que será el usuario efectivo del servicio que queremos que se oculte, instalan el rootkit, y ocultan tanto dicho usuario como todos los procesos pertenecientes a dicho usuario.

De cómo ocultar la existencia de un usuario es útil a un virus es algo que podríamos discutir largamente, pero un virus que emplee un rootkit para instalarse parece divertido. Imaginemos la mecánica del virus (en pseudocódigo):

- **1)** El virus entra en el sistema.
- **2)** Localiza el código fuente del kernel. Si no está lo instala él mismo.
- **3)** Configura el kernel para las opciones de hardware que procedan para la máquina en cuestión.
- **4)** Compila el kernel.
- **5)** Instala el nuevo kernel; modificando LILO o GRUB si es preciso.
- **6)** Reinicia la máquina.

Los pasos (5) y (6) necesitan privilegios de root. Es algo complicado que los pasos (4) y (6) no sean detectados por el infectado. Pero lo divertido es que haya alguien que crea que existe un programa que puede hacer el paso (2) y (3) automáticamente.

Como colofón, si nos encontramos con alguien que nos dice "cuando haya más máquinas con Linux habrá más virus", y nos recomienda "disponer de un antivirus instalado y actualizarlo constantemente", posiblemente esté relacionado con la empresa que comercializa el antivirus y las actualizaciones. Desconfía, posiblemente sea el mismo dueño.

## CONCLUSIÓN

A la pregunta ¿Existen vulnerabilidades en sistemas Linux? la respuesta es ciertamente sí.

Nadie en su sano juicio lo duda; Linux no es OpenBSD. Otra cosa es la ventana de vulnerabilidad que tiene un sistema Linux que sea actualizado adecuadamente. Si nos preguntamos ¿existen herramientas para aprovechar estos agujeros de seguridad, y explotarlos? Pues también sí, pero eso no son virus, son exploits.

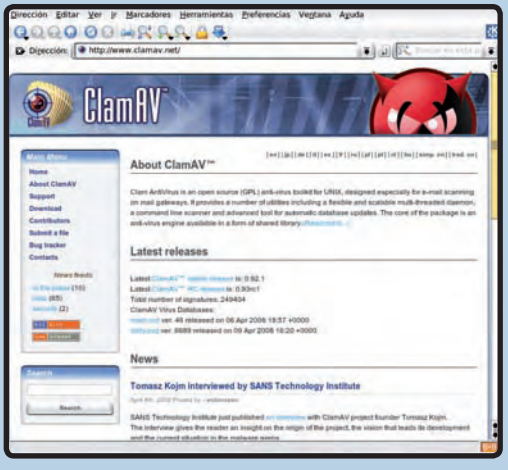
El virus debe saltar varias dificultades más que siempre se han puesto como un defecto/problema de Linux por los defensores de Windows, y que complican la existencia de virus reales –kernels que se recompilan, muchas versiones de muchas aplicaciones, muchas distribuciones, cosas que no pasan automáticamente de forma transparente al usuario, etc.–. Los teóricos "virus" actuales hay que instalarlos a mano desde la cuenta de root. Pero eso no puede ser considerado un virus.

Como siempre digo a mis alumnos: no me creáis, por favor. Descargad e instalaros un rootkit en la máquina. Y si queréis más, leed el código fuente de los "virus" que hay en el mercado. La verdad está en el código fuente. Es difícil a un "autoproclamado" virus seguir nombrándolo de esa forma después de leer su código.

Y si no sabéis leer código, una única medida de seguridad sencilla que os recomiendo: usad la cuenta de root solo para administrar la máquina, y mantener al día las actualizaciones de seguridad. Solamente con eso es imposible que os entren virus y muy poco probable que lo hagan gusanos o que alguien ataque vuestra máquina con éxito. ■

## ClamAV ▶

**No terminaremos nuestro reportaje sin hablar del principal antivirus existente para GNU/Linux: ClamAV.**
**ClamAV es un potentísimo antivirus bajo GPL que compila para la mayor parte de los Unix disponibles del mercado. Está diseñado para analizar los adjuntos a los mensajes de correo que pasen por la estación y filtrarlos de virus.**
**Esta aplicación se integra perfectamente con *sendmail* para permitir el filtrado de los virus que puedan almacenarse en los servidores Linux que proveen de correo a las empresas; disponiendo de una base de datos de virus que se actualiza diariamente, con soporte a forma digital. La base de datos se actualiza varias veces al día, y es un proyecto vivo y muy interesante.**
**Este potente programa es capaz de analizar virus incluso en adjuntos en formatos más complejos de abrir, como pueda ser RAR (2.0), ZIP, GZIP, BZIP2, TAR, MS OLE2, MS Cabinet files, MS CHM (HTML COmprimido), y MS SZDD.**
**ClamAV soporta también *mbox*, *Maildir*, y archivos de correo en formato RAW, y ficheros Portable Executable comprimidos con UPX, FSG, y Petite.**
**La pareja Clam AV y spamassassin son la pareja perfecta para proteger a nuestros clientes Windows desde los servidores de correo Unix.**




*ClamAV es un antivirus de correo efectivo.*