



Universidad de Córdoba

Escuela Politécnica Superior

Seguridad Informática
Trabajo de la asignatura

Bugs y exploits en GNU/Linux

Luis Díaz Más

2º Ingeniería Informática (2º Ciclo)

Córdoba - 3 de abril de 2008

Esta obra está bajo una licencia Reconocimiento-No comercial 2.5 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc/2.5/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Usted es libre de:



copiar, distribuir y comunicar públicamente la obra



hacer obras derivadas

Bajo las siguientes condiciones:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



No comercial. No puede utilizar esta obra para fines comerciales.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Índice

1. Introducción	1
2. El kernel de Linux y su modelo de desarrollo	1
2.1. Introducción	1
2.1.1. Controversia del nombre GNU/Linux	2
2.2. Las primeras versiones 2.x	2
2.3. Los kernels 2.6.x	3
2.3.1. Los Kernels 2.6.x.y	5
2.4. El ciclo de revisión	7
2.5. Los kernels -mm	7
2.6. Los kernels -git	8
3. Bugs y Exploits	9
3.1. ¿Qué clases de exploits hay?	9
3.2. El bug CVE-2008-0600 y su exploit	11
3.3. Velocidad de respuesta ante fallos de seguridad en Linux	13
3.4. GCC 4.3.0 expone un bug del kernel	15
4. Conclusiones	16
Referencias	17

1. Introducción

La seguridad es una cualidad que es difícil de encontrar en cualquier aspecto de la vida cotidiana al 100 % y en los sistemas informáticos sucede exactamente lo mismo. El documento que se presenta a continuación surge de la reciente noticia del descubrimiento de una vulnerabilidad en el kernel de Linux que permite a un usuario con acceso local obtener los privilegios de root ejecutando un simple exploit [2].

Este documento pretende además dar una ligera idea al lector de los diferentes aspectos de seguridad que se suelen tener en cuenta dentro de los sistemas operativos libres GNU/Linux [18], y describir algunos de los términos clave que aparecen en dicho contexto. En los sistemas basados en GNU/Linux se suelen tener en cuenta aspectos relacionados con:

- Seguridad de la red.
- Seguridad del host.
- Seguridad en cuentas de usuario.
- Seguridad física.
- Detección de intrusiones.
- Seguridad en sistemas de archivos.
- Cortafuegos.
- Encriptación, criptografía y autenticación.
- Seguridad del kernel.
- Exploits.

En este documento nos centraremos en las cuestiones referentes a **Seguridad del kernel** y **exploits**, y además para entrar completamente en materia, la siguiente sección hará una introducción sobre el kernel de GNU/Linux y como se encuentra el estado del arte en cuanto al proceso de desarrollo del kernel 2.6[7].

2. El kernel de Linux y su modelo de desarrollo

2.1. Introducción

Los comienzos acerca del kernel de GNU/Linux [17] giran en torno a dos grandes figuras dentro del movimiento del software libre en la actualidad, Richard Stallman y Linus Torvalds. El primero de ellos en 1983 inició el proyecto GNU con el objetivo de crear un sistema operativo parecido a UNIX (UNIX-like) y compatible con POSIX. Como parte de su trabajo escribió la licencia pública general GNU (GPL). A principios de 1990 se tenía lo suficiente para crear un sistema operativo

completo, pero el kernel GNU, llamado Hurd, había fallado en la captación de desarrolladores y por tanto en su velocidad de desarrollo.

Linus Torvalds por su parte comenzó en 1991, en Helsinki, un proyecto que más tarde se convertiría en el Kernel de Linux. Inicialmente fue un emulador de terminal que Torvalds usaba para acceder a grandes servidores de la universidad, y finalmente culminó en Mimix usando el compilador GNU de C, siendo esta la principal elección hoy día para compilar Linux.

El primer nombre que adoptó la invención de Linus Torvalds fue “Freax”, una palabra compuesta por “freak”, “free” y “x” en referencia a Unix. El proyecto permaneció con este nombre sobre medio año, pero después consideró el nombre de “Linux” que inicialmente no había sido considerado por parecer demasiado egoísta (por la similitud con su nombre).

La primera publicación del Kernel de Linux - entonces conocido exclusivamente como Linux - fue bajo su propia licencia, que era esencialmente, una licencia de código compartido con una restricción en la actividad comercial. En 1992 Torvalds sugirió el cambio a la licencia pública general de GNU. Primero anunció este cambio en las notas de lanzamiento de la versión 0.12 y a mediados de diciembre de 1992 publicó la versión 0.99 haciendo uso de GNU GPL. Los desarrolladores de Linux y GNU trabajaron para integrar los componentes GNU con Linux para hacer un sistema operativo funcionalmente potente y libre. Por entonces, Torvalds indicó: “Hacer Linux GPL podría ser definitivamente la mejor cosa que nunca he hecho”.

2.1.1. Controversia del nombre GNU/Linux

La designación “Linux” fue inicialmente usada por Torvalds solo para el kernel de Linux. El kernel, sin embargo, era frecuentemente usado junto otro software, especialmente software perteneciente al proyecto GNU. Esto rápidamente tomo la adopción popular de GNU Software. En Junio de 1994 en un boletín de GNU, Linux fue referido como un “clon libre de Unix”, y el proyecto Debian comenzó llamando a su producto Debian GNU/Linux. En mayo de 1996, Richard Stallman publicó el editor Emacs, en el que el tipo de sistema fue renombrado de Linux a Lignux. Esto intentaba hacer referencia a la combinación de GNU y Linux, pero esta idea se abandonó pronto en favor de “GNU/Linux”.

2.2. Las primeras versiones 2.x

Hay muchos otros conocidos que mantienen el kernel de Linux junto a Torvalds como Alan Cox y Marcelo Tosatti. Cox mantuvo la versión 2.2 del Kernel hasta que esta fue interrumpida a finales de 2003. Asimismo, Tosatti mantuvo la versión 2.4 del Kernel hasta mediados de 2006. Andrew Morton dirige el desarrollo y administración del kernel 2.6, que fue lanzado el 18 de Diciembre de 2003 en su primera encarnación estable y es en la actualidad el Kernel que se usa en la mayoría de distribuciones GNU/Linux.

La versión del núcleo de Linux consta actualmente de cuatro números. Por ejemplo, actualmente la versión actual del kernel de la distribución GNU/Linux Ubuntu 7.10 es la 2.6.22-14. Asumamos que el número de la versión está compuesta de esta forma: A.B.C.D (ej.: 2.2.1, 2.4.13, 2.6.12.3 o 2.6.22-14).

- El **número A** denota la versión del núcleo. Es el que cambia con menor frecuencia y solo lo hace cuando se produce un gran cambio en el código o en el concepto del núcleo. Históricamente solo ha sido modificado dos veces: en 1994 (versión 1.0) y en 1996 (versión 2.0).
- El **número B** denota la mayor revisión del núcleo.
Antes de la serie de Linux 2.6.x, los números pares indicaban la versión “estable” lanzada. Por ejemplo una para uso de fabricación, como el 1.2, 2.4 ó 2.6. Los números impares, en cambio, como la serie 2.5.x, son versiones de desarrollo, es decir que no son consideradas de producción.
Comenzando con la serie Linux 2.6.x, no hay gran diferencia entre los números pares o impares con respecto a las nuevas herramientas desarrolladas en la misma serie del núcleo. Linus Torvalds dictaminó que este será el modelo en el futuro.
- El **número C** indica una revisión menor en el núcleo. En la forma anterior de versiones con tres números, esto fue cambiado cuando se implementaron en el núcleo los parches de seguridad, bugfixes, nuevas características o drivers. Con la nueva política, solo es cambiado cuando se introducen nuevos drivers o características; cambios menores se reflejan en el número D.
- El **número D** se produjo cuando un grave error, que requiere de un arreglo inmediato, se encontró en el código NFS de la versión 2.6.8. Sin embargo, no habían otros cambios como para lanzar una nueva revisión (la cual hubiera sido 2.6.9). Entonces se lanzó la versión 2.6.8.1, con el error arreglado como único cambio. Con 2.6.11, esto fue adoptado como la nueva política de versiones. Bug-fixes y parches de seguridad son actualmente manejados por el cuarto número dejando los cambios mayores para el número C.

En ocasiones tras los los dígitos de las versiones puede haber algunas letras como “rc1” o “mm2”. El “rc” se refiere a “release candidate” e indica un lanzamiento no oficial. Otras letras usualmente (pero no siempre) hacen referencia a las iniciales de la persona. Esto indica una bifurcación en el desarrollo del núcleo realizado por esa persona, por ejemplo “ck” se refiere a Con Kolivas, “ac” a Alan Cox, mientras que “mm” se refiere a Andrew Morton. Estos, junto a otro grupo de personas son los usuales desarrollares que mantienen el núcleo de Linux.

2.3. Los kernels 2.6.x

El modelo de desarrollo para Linux 2.6 fue un cambio significativo desde el modelo de desarrollo de Linux 2.5. Previamente existía una rama estable (2.4) donde se habían producido cambios menores y seguros, y una rama inestable (2.5) donde estaban permitidos cambios mayores. Esto significó que los usuarios siempre tenían una versión 2.4 a prueba de fallos y con lo último en seguridad y casi libre

de pulgas, aunque tuvieran que esperar por las características de la rama 2.5. La rama 2.5 fue eventualmente declarada estable y renombrada como 2.6. Pero en vez de abrir una rama 2.7 inestable, los desarrolladores de núcleos eligieron continuar agregando los cambios en la rama “estable” 2.6. De esta forma no había que seguir manteniendo una rama vieja pero estable y se podía hacer que las nuevas características estuvieran rápidamente disponibles y se pudieran realizar más test con el último código.

Sin embargo, el modelo de desarrollo del nuevo 2.6 también significó que no había una rama estable para aquellos que esperaban seguridad y bug fixes sin necesitar las últimas características. Los arreglos solo estaban en la última versión, así que si un usuario quería una versión con todos los bug fixed conocidos también tendría las últimas características, las cuales no habían sido bien testeadas. Una solución parcial para esto fue la versión ya mencionada de cuatro números (2.6.x.y), la cual significaba lanzamientos puntuales creados por el equipo estable (Greg Kroah-Hartman, Chris Wright, y quizás otros). El equipo estable solo lanzaba updates para el núcleo más reciente, sin embargo esto no solucionó el problema de la falta de una serie estable de núcleo. Los vendedores de distribuciones GNU/Linux como Red Hat y Debian, mantienen los núcleos que salen con sus lanzamientos, de forma que una solución segura para la mayoría de personas es seguir el núcleo de una distribución.

Recientemente en la lista de desarrollo del kernel LKML (<http://lkml.org/>) se ha intentado documentar dicho modelo, para plasmar sus puntos fundamentales. A continuación se detallan algunos de estos puntos fundamentales en el desarrollo actual del kernel de Linux, los cuales han sido extraídos del documento elaborado Paolo Ciarrocchi[7], un integrante activo de la LKML.

Los kernels 2.6.x (el tercer dígito es el que varía) son las versiones estables liberadas por Linus Torvalds, en donde la versión con más alta numeración es la más reciente. Si se encuentran **defectos** o **vulnerabilidades** se realiza un parche “-stable” con las correcciones sobre esta base. Para controlar los cambios realizados, una vez que un nuevo kernel base 2.6.x es realizado, se hace disponible un parche con las diferencias entre el kernel 2.6.x previo y el nuevo para que pueda ser aplicado fácilmente con el comando **patch**.

El desarrollo de los kernels 2.6.x es como sigue: Tan pronto como un nuevo kernel es liberado, se abre una “ventana” de tiempo de dos semanas. Durante este período los encargados del mantenimiento pueden emitir grandes “diffs” (impresión de las líneas diferentes entre dos archivos) a Linus, y habitualmente estos parches son colocados en los kernels -mm (La bifurcación del Kernel de Andrew Morton) durante unas semanas. La manera preferida para emitir grandes cambios es usando GIT, un sistema de control de versiones distribuido orientado a la velocidad, efectividad y utilidad en grandes proyectos (más información sobre GIT en <http://git.or.cz/> y <http://www.kernel.org/pub/software/scm/git/docs/tutorial.html>).

Después de dos semanas un kernel -rc1 (Release candidate 1) se libera, a partir de entonces, sólo es posible la emisión de parches que no incluyan nuevas funcionalidades que pudieran afectar a la estabilidad del kernel al completo. Hay que advertir que un nuevo driver completo, sistema de archivos, o cualquier otra características de gran envergadura podría ser admitido después del -rc1, pues no hay riesgo de causar conflictos con este tipo de cambios. Un nuevo -rc es realizado cuando Linus considera que el árbol -git actual está en un estado razonablemente sano para ser testeado. La meta en un principio era liberar un nuevo kernel -rc cada semana, pero a medida que pasa el tiempo los tiempos se van haciendo mayores debido a que el Kernel está ya en un estado muy avanzado.

El proceso continua hasta que el kernel es considerado “listo”, debiendo durar dicho proceso alrededor de 6 semanas (se liberarán unos 6 kernels por año) y habitualmente incluyendo unas 4 o 5 versiones -rc.

Hay un número de herramientas usadas por la comunidad para medir la calidad y rendimiento del kernel. Un par de ejemplos son:

- <http://kernel-perf.sourceforge.net/>. Parafraseando el contenido del sitio: “Somos un grupo de ingenieros del kernel Linux especializados en el desafío de testear el kernel Linux. Con la intención de seguir la pista del rendimiento, estamos ejecutando un largo conjunto de benchmarks que cubren los componentes centrales del kernel Linux (administración de la memoria virtual, subsistema de entrada/salida, planificador de procesos, sistema de archivos, red, drivers de dispositivos, etc). Los benchmarks son ejecutados en una gran variedad de plataformas cada semana, testeando los últimos snapshot del árbol -git de Linus (Las últimas instantáneas o versiones del árbol). Datos comprensibles del rendimiento de nuestros tests son almacenados aquí para su fácil acceso”.
- <http://bugzilla.kernel.org/>. Es la instancia oficial on-line de seguimiento de errores del kernel. Los usuarios son invitados a informar sobre todos los errores que encuentren usando esta herramienta.

Merece la pena mencionar que Andrew Morton escribió en LKML: “Nadie sabe cuando un kernel será liberado, pues está en relación con la percepción del estado de los errores, no en relación a tiempos límite”.

2.3.1. Los Kernels 2.6.x.y

Los kernel con versiones de 4 dígitos son los kernel -stable. Contienen pequeñas correcciones críticas para problemas de seguridad o problemas significativos de defectos o vulnerabilidades descubiertos en un kernel 2.6.x dado. Es la rama recomendada para usuarios que quieren el kernel más estable reciente y no están interesados en ayudar a testear las versiones de desarrollo/experimentación. Para entenderlo mejor, estas son las versiones que aparecen normalmente en las principales distribuciones GNU/Linux orientadas a los usuarios normales.

Si no hay kernel 2.6.x.y disponible, entonces la versión más alta de 2.6.x es el kernel estable actual. Los 2.6.x.y son mantenidos por el equipo “estable” (en kernel.org), e intentan ser realizados cada semana. Las reglas sobre qué clase de parches son aceptados en la rama “-stable” son (Estos puntos pueden ser ampliados en el artículo “Contributing to the Linux Kernel” [10]):

- Debe obviamente corregir algún problema de seguridad o vulnerabilidad y estar apropiadamente probado.
- No debe ser más grande de 100 líneas, incluyendo el contexto del parche (Las líneas extra que aparecen antes y después de los bloques “diff” en el parche).
- Debe arreglar sólo una cosa.
- Debe arreglar un error real que afecta a la gente (no algo del tipo “esto podría ser un error”).
- Debe arreglar un problema que causa un error de construcción del kernel (no para cosas marcadas como CONFIG_BROKEN), un “oops”, un cuelgue, una corrupción de datos, una cuestión real de seguridad, o una cuestión del tipo “oh, esto no es bueno”. En resumen, algo crítico.
- No emitir condiciones de carrera teóricas, a menos que exista una explicación de cómo la condición de carrera puede ser explotada.
- No puede tener ningún tipo de arreglos triviales (cambios ortográficos, limpieza de espacios, etc.).
- Debe ser aceptado por el mantenedor del subsistema involucrado.
- Debe seguir las reglas expresadas en Documentation/SubmittingPatches existentes en cada versión del Kernel [9].

El procedimiento seguido para emitir estos parches al árbol -stable son los siguientes:

- Enviar el parche, después de verificar que sigue las reglas anteriores, a stable@kernel.org.
- El emisor recibirá un acuse de recibo afirmativo cuando el parche sea aceptado en la cola de espera, o un acuse de recibo negativo si el parche es rechazado. Esta respuesta podría tardar varios días, de acuerdo con la planificación del desarrollador.
- Si es aceptado, el parche será añadido a la cola -stable, para la revisión de otros desarrolladores.
- Los parches de seguridad no deberían enviarse a la dirección anterior, sino a security@kernel.org.

2.4. El ciclo de revisión

Cuando los mantenedores de la rama `-stable` se deciden por un ciclo de revisión, el parche es enviado al comité de revisión y al mantenedor del área afectada por él (a menos que sea el propio mantenedor el que envía el parche) con copia a la lista de correo `linux-kernel`. El comité de revisión tiene 48 horas para aceptar o rechazar el parche.

Si el parche es rechazado por uno de los miembros del comité, o miembros de la lista `linux-kernel` tienen objeciones sobre el parche sacando a colación cuestiones que el mantenedor y/o los miembros del comité no advirtieron, el parche será eliminado de la cola.

Al final del ciclo de revisión, los parches aceptados serán añadidos a la última versión `-stable`, y una nueva versión `-stable` tendrá lugar. Los parches de seguridad serán aceptados directamente en el árbol `-stable` desde el equipo de seguridad del kernel y no a través del ciclo normal de revisión. Para más detalles sobre esta cuestión, se aconseja contactar con el equipo de seguridad del kernel.

Comité de revisión: Está compuesto por un número de desarrolladores que voluntariamente realizan esta tarea, y unos cuantos no voluntariamente.

2.5. Los kernels `-mm`

Son kernels experimentales liberados por Andrew Morton. El árbol `-mm` sirve como una forma de proporcionar terreno para nuevas características y otros parches experimentales. Una vez que un parche ha probado su valía en `-mm` durante un tiempo, Andrew lo proporciona a Linus Torvalds para su inclusión en la línea principal. Aunque se anima a que los parches fluyan a linus a través de la rama `-mm`, no siempre se obliga a ello. Mantenedores de subsistemas (o anónimos) a veces envían sus parches directamente a Linus, aunque ellos han sido mezclados y testeados en `-mm` (o a veces, incluso sin testeo previo en la rama `-mm`).

Esta rama está en constante cambio y contiene muchas características experimentales, muchos parches para depuración no apropiados para la línea principal, etc. Es la rama más experimental de las descritas en este documento. Estos kernels no son apropiados para el uso en sistemas que se suponen tienen que ser estables, su uso es más arriesgado que el de otras ramas (Hay que asegurarse de mantener actualizadas las copias de seguridad para el uso de cualquier kernel experimental, e incluso más para los kernels `-mm`).

Estos kernels, además de todos los parches experimentales contienen también cualquier cambio en la línea principal `-git` disponible durante su realización. Los kernels `-mm` no son liberados en base a una planificación fija, pero habitualmente unos cuantos kernels `-mm` son realizados entre cada kernel `-rc` (de uno a tres es lo común).

2.6. Los kernels -git

Estos kernels son instantáneas (snapshots) diarias del árbol de Linus (manejado en un repositorio git, de ahí el nombre).

Se realizan parches habitualmente a diario (incluso varios al día) que representan el estado actual del árbol de Linus. Son más experimentales que los kernels -rc, puesto que se generan automáticamente incluso sin un vistazo superficial para determinar si están en un estado "sano". Para ver como funciona este aspecto puede visitar <http://git.kernel.org/>.

3. Bugs y Exploits

Un **exploit** es un programa o técnica que aprovecha una vulnerabilidad o deficiencia de otro programa (bug). Los exploits dependen de los sistemas operativos y sus configuraciones, de las configuraciones de los programas que se están ejecutando en un ordenador y de la red de área local donde se encuentran. El fin del exploit puede ser la destrucción o inhabilitación del sistema atacado, aunque normalmente se trata de violar las medidas de seguridad para poder acceder al mismo de forma no autorizada y emplearlo en beneficio propio o como origen de otros ataques a terceros.

Pero ¿qué es una vulnerabilidad o bug? Una **vulnerabilidad** o **bug** es un aspecto o error no tenido en cuenta de un sistema informático que evitará que se pueda usar correctamente, o que permitirá que lo controlen personas no autorizadas. Hay muchos tipos de vulnerabilidades, existiendo posibles errores en la configuración de servicios o bien errores en la programación de dichos servicio. Los exploits que afectan a estas vulnerabilidades pueden estar creados en cualquier lenguaje, por ejemplo los exploit de Unix o GNU/Linux pueden estar hechos en perl, aunque lo más común es que estén desarrollados en C [1].

La utilización de exploits sin la supervisión/aprobación del administrador del sistema puede ser considerado (y lo es) un delito y está penado con fuertes multas e incluso cárcel. A pesar de ello, también son utilizados por los testers y hackers de seguridad informática para mejorar la seguridad de las compañías.

3.1. ¿Qué clases de exploits hay?

Se puede diferenciar a los exploit en locales y remotos. Los exploits locales actúan en la máquina en la que están. Por tanto si deseo atacar otro ordenador primero tendré que subir ese código y luego conseguir ejecutarlo para que funcione allí. Si lo ejecuto aquí actuará aquí.

Los exploit remotos se ejecutan desde una máquina, pero actúan en otra máquina víctima de manera que utilizan algún servicio conectado a la red que tenga una vulnerabilidad para poder obtener un tipo de beneficio.

Según las categorías de vulnerabilidades utilizadas, los exploits se pueden clasificar de la siguiente forma [16]:

- **De desbordamiento de buffer:** error de software que se produce cuando se copia una cantidad de datos sobre un área que no es lo suficientemente grande para contenerlos, sobrescribiendo de esta manera otras zonas de memoria. Como consecuencia, se producirá una excepción del acceso a memoria seguido de la terminación del programa o, si se trata de un usuario obrando con malas intenciones y un programa con privilegios especiales, la explotación de una vulnerabilidad o agujero de seguridad.

- **De condición de carrera (race condition):** error que se produce en programas cuando no han sido diseñados adecuadamente para su ejecución simultánea con otros programas. Un ejemplo típico es el interbloqueo que se produce cuando dos procesos están esperando a que el otro realice una acción. Como los dos están esperando, ninguno llega a realizar la acción que el otro espera.
- **De error de formato de cadena (format string bugs):** clase de vulnerabilidad software descubierta sobre 1999 y que anteriormente se pensaba que era inofensiva. Los ataques de formato de cadenas pueden ser usados para quebrar programas o ejecutar código malicioso. El problema proviene del uso de entradas de usuario que no hacen uso de filtros como los parámetros de formato de cadenas en C en funciones como `printf()`. Un usuario malicioso podría usar los tokens de formato `%s` o `%x`, entre otros, para mostrar datos de la pila o otras localizaciones de memoria. Los exploits típicos usan una combinación de dichas técnicas para forzar a un programa a sobrescribir la dirección de una función de una librería o la dirección de retorno de la pila con un puntero a algún código malicioso.
- **De Cross Site Scripting (XSS):** es un ataque basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado. Su nombre original “Cross Site Scripting”, y renombrado XSS para que no sea confundido con las hojas de estilo en cascada (CSS), originalmente abarcaba cualquier ataque que permitiera ejecutar código de “scripting”, como VBScript o javascript, en el contexto de otro dominio. Recientemente se acostumbra a llamar a los ataques de XSS “HTML Injection”, sin embargo el término correcto es XSS. Estos errores se pueden encontrar en cualquier aplicación HTML, no se limita a sitios web, ya que puede haber aplicaciones locales vulnerables a XSS, o incluso el navegador en sí. El problema está en que normalmente no se validan correctamente los datos de entrada que son usados en cierta aplicación.
- **De Inyección SQL:** es una vulnerabilidad informática en el nivel de la validación de las entradas a la base de datos de una aplicación. El origen es el filtrado incorrecto de las variables utilizadas en las partes del programa con código SQL. Es, de hecho, de los errores de clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o de script que esté incrustado dentro de otro. Una inyección SQL sucede cuando se inserta o “inyecta” un código SQL “invasor” dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código “invasor” en la base de datos. La inyección SQL es un problema de seguridad informática que debe ser tomado en cuenta por el programador para prevenirlo. Un programa hecho con descuido, displicencia, o con ignorancia sobre el problema, podrá ser vulnerable y la seguridad del sistema puede quedar ciertamente comprometida.
- **De Inyección de Caracteres (CRLF):** se refiere a la combinación de dos códigos de control: CR (retorno de carro) y LF (salto de línea). No es habitual encontrar este tipo de vulnerabilidades en los programas, aunque el

conocido sistema de comunicación por foros de código libre Yabb sufrió en el 2007 ataques de este tipo que fueron rápidamente solventados [6].

- **De denegación del servicio:** también llamado ataque DoS (de las siglas en inglés Denial of Service), es un ataque a un sistema de ordenadores o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima. Se genera mediante la saturación de los puertos con flujo de información, haciendo que el servidor se sobrecargue y no pueda seguir prestando servicios, por eso se le dice “denegación”, pues hace que el servidor no de abasto a la cantidad de usuarios. Quizás sea el tipo de ataque más común, ya que es de los más sencillos de utilizar y está al alcance de cualquier persona que se proponga hacer un daño. Sin ir más lejos en Agosto de 2007, el conocido programa de comunicación Skype sufrió unos días con problemas en su servicio que al parecer venían provocados por un ataque de este tipo que hacía que se sufrieran desconexiones constantes [5]. El código que ocasionaba este problema podría ser tan simple como el siguiente [14]:

```
#!/usr/bin/perl
# Simle Code by Maranax Porex ;D
# Ya Skaypeg!!

for ($i=256; $i>xCCCC; $i=$i+256){
$eot='AAAA' x $i;
call_sp();
}
exit;

sub call_sp(){
$str=""C:\\Program Files\\Skype\\Phone\\Skype.exe\"
  \"/uri:$eot\"";
}
```

- **De Inyección múltiple HTML (Multiple HTML Injection):** mediante este tipo de ataques un usuario con malas intenciones podría ejecutar código script arbitrario en el navegador de un usuario confiado en el contexto del sitio afectado. Esto puede permitir a los atacantes roban credenciales de autenticación basadas en cookies y lanzar otros ataques. Son también muy comunes y ocurren a diario en muchos sitios y servicios web como: Drupal[15], phpBB[13], Jimzora[8], etc.

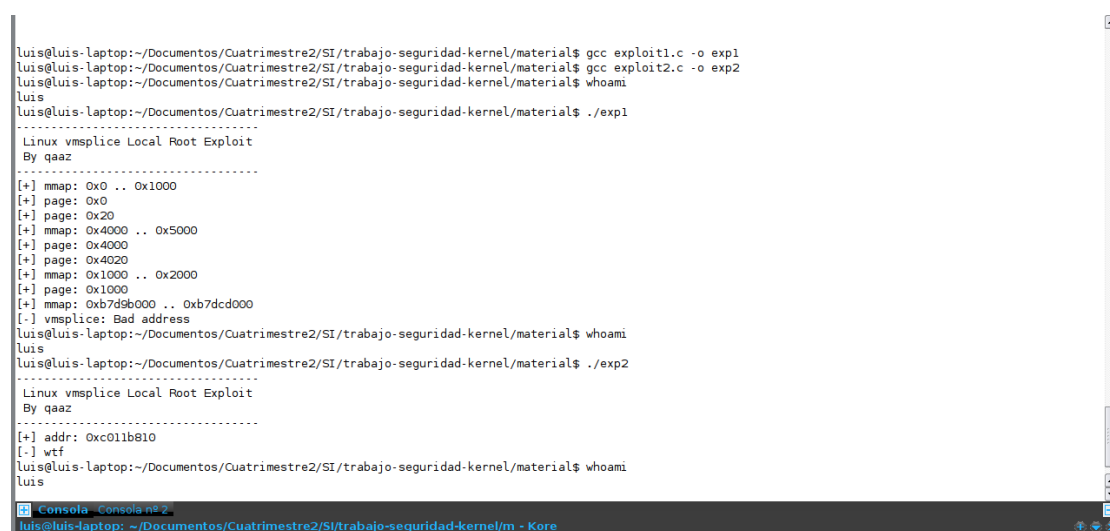
3.2. El bug CVE-2008-0600 y su exploit

El bug **CVE-2008-0600** ha sido descubierto el 8 de febrero en la llamada de sistema `vmsplice()` que fue introducida en el kernel 2.6 de Linux, concretamente

en el 2.6.17. Este es el tercero de una serie de exploits que permiten obtener privilegios de root alrededor de la misma llamada al sistema, los dos anteriores fueron CVE-2008-0009 y CVE-2008-0010. Estos exploits permiten a un usuario local no-root obtener los privilegios de root [3],[4].

Esta noticia es el origen del desarrollo de este trabajo. Al mantener actualizados diariamente mis equipos personales esta vulnerabilidad ha sido corregida por las actualizaciones que me brinda el soporte de mi distribución GNU/Linux Ubuntu 7.10. En la figura 1 se aprecia como dos versiones diferentes del exploit fallan al intentar explotar la vulnerabilidad del sistema y conseguir los privilegios de root. Estos dos códigos se pueden encontrar en las siguientes direcciones Web:

- <http://www.milw0rm.com/exploits/5092>
- <http://www.milw0rm.com/exploits/5093>

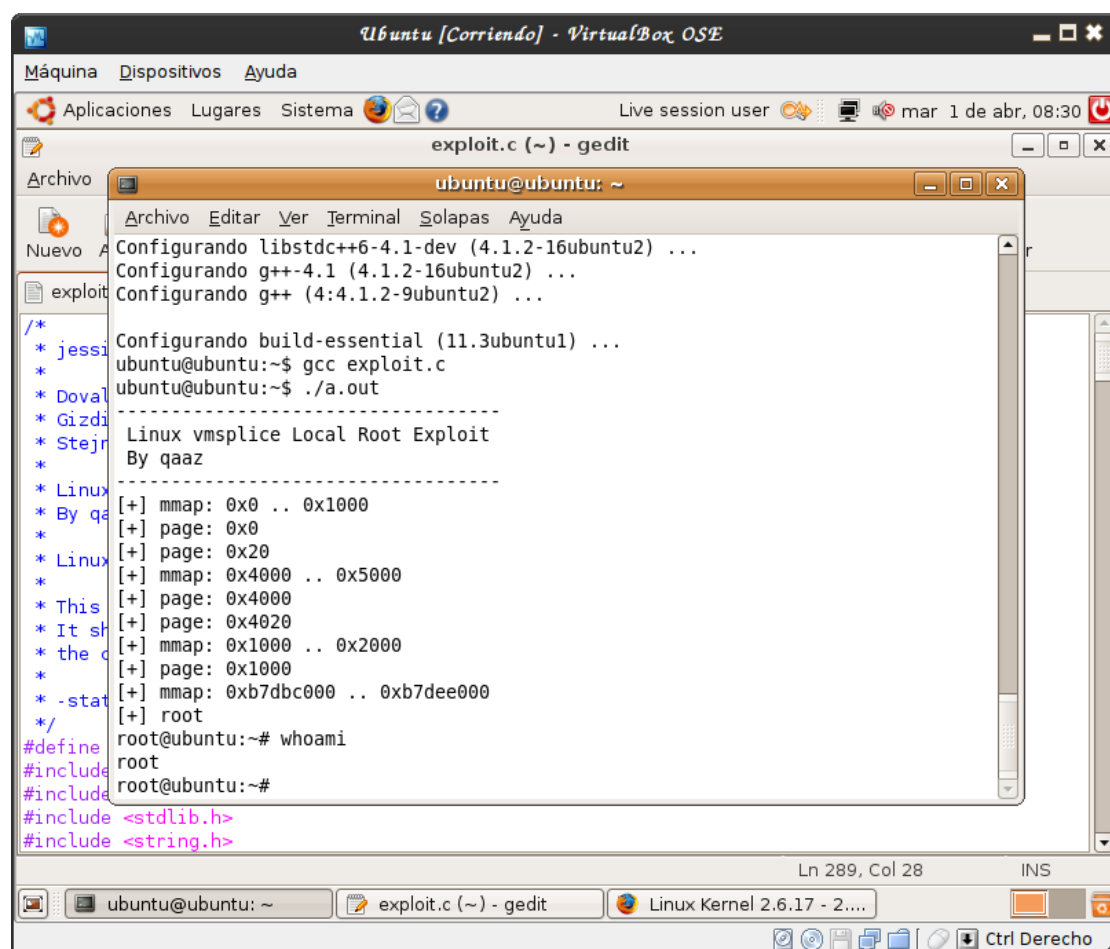


```
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ gcc exploit1.c -o exp1
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ gcc exploit2.c -o exp2
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ whoami
luis
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ ./exp1
Linux vmsplICE Local Root Exploit
By qaaz
-----
[+] mmap: 0x0 .. 0x1000
[+] page: 0x0
[+] page: 0x20
[+] mmap: 0x4000 .. 0x5000
[+] page: 0x4000
[+] page: 0x4020
[+] mmap: 0x1000 .. 0x2000
[+] page: 0x1000
[+] mmap: 0xb7d9b000 .. 0xb7dcd000
[.] vmsplICE: Bad address
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ whoami
luis
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ ./exp2
Linux vmsplICE Local Root Exploit
By qaaz
-----
[+] addr: 0xc011b810
[.] wtf
luis@luis-Laptop:~/Documentos/Cuatrimstre2/SI/trabajo-seguridad-kernel/material$ whoami
luis
```

Figura 1: Fallo de ejecución del exploit en una distribución Ubuntu 7.10 actualizada

Sin embargo en la figura 2 se aprecia un claro ejemplo de como se ejecuta el exploit y se aprovecha la vulnerabilidad que explicamos en esta sección. Para ello se ha usado la misma distribución Ubuntu 7.10 siendo arrancada desde un LiveCD en una máquina virtual mediante el software de virtualización **VirtuaBox**. En esta ocasión el exploit ha realizado su cometido debido a que el sistema no ha sido actualizado y la vulnerabilidad permanecía aún intacta en el kernel del sistema. Este exploit utiliza un cóctel de diferentes tipos de exploits: desbordamiento de buffer, desbordamiento de enteros y punteros NULL [12].

En la dirección <http://lkml.org/lkml/2008/2/11/25> Daniel Phillips aporta el parche que corrige la vulnerabilidad en los kernels actuales. Además a partir de la versión 2.6.24.2 del kernel de Linux este problema queda resuelto de forma definitiva.



```
Ubuntu [Corriendo] - VirtualBox OSE
Máquina Dispositivos Ayuda
Aplicaciones Lugares Sistema Live session user mar 1 de abr, 08:30
exploit.c (~) - gedit
Archivo Editar Ver Terminal Solapas Ayuda
ubuntu@ubuntu: ~
Archivo Editar Ver Terminal Solapas Ayuda
Configurando libstdc++6-4.1-dev (4.1.2-16ubuntu2) ...
Configurando g++-4.1 (4.1.2-16ubuntu2) ...
Configurando g++ (4:4.1.2-9ubuntu2) ...
Configurando build-essential (11.3ubuntu1) ...
ubuntu@ubuntu:~$ gcc exploit.c
ubuntu@ubuntu:~$ ./a.out
-----
Linux vmsplICE Local Root Exploit
By qaaz
-----
[+] mmap: 0x0 .. 0x1000
[+] page: 0x0
[+] page: 0x20
[+] mmap: 0x4000 .. 0x5000
[+] page: 0x4000
[+] page: 0x4020
[+] mmap: 0x1000 .. 0x2000
[+] page: 0x1000
[+] mmap: 0xb7dbc000 .. 0xb7dee000
[+] root
root@ubuntu:~# whoami
root
root@ubuntu:~#
Ln 289, Col 28 INS
ubuntu@ubuntu: ~ exploit.c (~) - gedit Linux Kernel 2.6.17 - 2.... Ctrl Derecho
```

Figura 2: Compilación y ejecución del exploit que aprovecha el bug CVE-2008-0600

También es importante resaltar que a pesar de que el bug se descubrió el 8 de febrero no fue hasta el día 11 del mismo mes donde los principales sitios de internet relacionados con este tipo de noticias no se hicieron eco del asunto. Es importante que este tipo de noticias se hagan visibles para los usuarios finales lo antes posible para evitar problemas de seguridad, y es por ello por lo que muchas distribuciones emiten noticias oficiales de correcciones de este tipo de vulnerabilidades.

3.3. Velocidad de respuesta ante fallos de seguridad en Linux

Ante los diversos bugs que se descubren dentro del kernel de Linux, hay una rápida respuesta por las personas que se dedican al mantenimiento y desarrollo del mismo. Existe un gran control en internet acerca de este tipo de fallos, y normalmente la mayoría de los bugs que se descubren son reportados rápidamente para intentar subsanarlos lo antes posible. Normalmente cada una de las distribuciones Linux existentes en el panorama tiene un grupo de personas encargadas de este tipo de cuestiones.

Pongamos como ejemplo el bug analizado en la sección anterior. Las principales

Distribución	Fecha/Hora	Retraso	Referencia
Debian GNU/Linux	2008-02-11 13:58	+0 horas	DSA 1494-1
Fedora	2008-02-11 22:39	+8 horas	FEDORA-2008-1423
Slackware Linux	2008-02-12 02:00	+12 horas	SSA:2008-042-01
Mandriva Linux	2008-02-12 07:06	+19 horas	MDVSA-2008:043
Frugalware Linux	2008-02-12 11:17	+21 horas	FSA-369
openSUSE	2008-02-12 12:43	+23 horas	SUSE-SA:2008:007
rPath Linux	2008-02-12 16:28	+26 horas	rPSA-2008-0052-1
Red Hat Enterprise Linux	2008-02-12 16:54	+27 horas	RHSA-2008:0129-01
Ubuntu	2008-02-12 17:23	+27 horas	USN-577-1
CentOS	2008-02-13 03:27	+37 horas	CESA-2008:0129

Tabla 1: Tiempo en corrección del bug CVE-2008-0600 de las principales distribuciones

distribuciones de Linux empezaron a sacar parches el día 11 de Febrero, el mismo día en que la noticia se hizo pública ampliamente. ¿Pero cual fue más rápida?. Naturalmente, la mayoría de distribuciones siempre necesitan algo de tiempo para evaluar el mejor acercamiento posible y comprobar las actualizaciones resultantes. Se depende del número de kernels y productos que necesitan ser parcheados y probados, la disponibilidad de expertos de seguridad en la distribución, coordinación de trabajo y del nivel de burocracia en cada organización. Sin embargo, desde el punto de vista del usuario, la actualización que es lanzada de forma más inmediata es la mejor [3].

En el mundo UNIX la mayoría de proveedores mantienen informados a sus usuarios mediante avisos de seguridad. Estos pueden ser vistos a través de listas de correos (actualmente el método más popular), o otras opciones como pueden ser los feeds RSS o demonios que comprueban periódicamente actualizaciones. Este tipo de avisos son muy importantes y no solo deben informar acerca del problema de seguridad en un producto, sino que tiene que mostrar al usuario como parchear la vulnerabilidad.

Un gran número de avisos de seguridad han sido publicados durante la semana posterior al conocimiento de la noticia del bug CVE-2008-0600 y del exploit sobre `vmsplice()`. Debian GNU/Linux fue la primera en publicar una solución, pero un día o dos después la mayoría de distribuciones la siguieron con sus propios anuncios. En la tabla 1 se puede apreciar los tiempos de respuesta empleados ante dicha vulnerabilidad por algunas de las principales distribuciones Linux.

Hoy en día, muchas distribuciones populares no publican avisos de seguridad. Esto es especialmente cierto para distribuciones de escritorio, muchas de las cuales no tienen el suficiente personal para publicar anuncios de manera formal. Hay incluso distribuciones que no proporcionan actualizaciones en absoluto. En un mundo ideal, todos los usuarios de Linux deberían ejecutar una distribución que tuviera una infraestructura de seguridad bien definida y deberían estar suscritos a las listas de correo de seguridad de dichas distribuciones, pero el mundo real es

diferente. La seguridad en los sistemas operativos es aún algo que no es serio en los proyectos y a lo que los usuarios finales no prestan demasiada atención.

Muchas de las distribuciones Linux existentes no aparecen en la tabla 1. ¿Significa esto que el sistema operativo de dichas distribuciones es vulnerable al exploit de la llamada `vmsplice()`? Depende. Hay distribuciones como PCLinuxOS o Gentoo que no publican avisos de seguridad formales, pero si se observa en sus directorios actuales hay modificaciones en los kernels con un sello de tiempo en los días 11 o 12 de Febrero, por lo que presumiblemente han corregido dicho error. Sin embargo hay otras distribuciones en donde esto no es así.

3.4. GCC 4.3.0 expone un bug del kernel

No siempre las vulnerabilidades del kernel son descubiertas por gente que intenta explotarlas. Recientemente tras un cambio realizado en la nueva versión 4.3.0 de GCC que está cerca de ver la luz, se ha llegado a una situación complicada que puede implicar posibles problemas de seguridad debido al descubrimiento de un bug en el Kernel de Linux [11].

GCC cambió algunos supuestos sobre los flags del procesador x86, de acuerdo con el estándar ABI (Application Binary Interface), que puede conducir a corrupciones de memoria en programas construidos con GCC 4.3.0. Nadie ha sugerido una manera de explotar el defecto, al menos por ahora, pero claramente es un problema que necesitar ser tratado.

El problema gira en torno al flag de dirección del x86 (DF), que rige si las operaciones de bloques de memoria operan hacia adelante o hacia atrás. El principal uso de esta bandera es soportar las copias de memoria superpuestas (*overlapping*), donde se podría requerir un procesamiento en la memoria hacia atrás, ya que los datos que son copiados no se sobrescriben exactamente como se hace en el progreso de la copia. El hacker de Debian Aurélien Jarno reportó el problema a linux-kernel el 5 de Marzo de este año, cuando construía el Steel Bank Common Lisp (SBCL) usando el nuevo compilador.

La nueva versión de GCC (4.3.0) asume que la bandera de dirección ha sido quitada de la entrada de cada función, como está especificado por el estándar ABI. Desafortunadamente, esto entra en conflicto con los manejadores de señales de Linux, que son llamados incorrectamente con la bandera en cualquier estado cuando la señal ocurre. Esto tiene el efecto de la pérdida de un bit de estado en el espacio de usuario del proceso que está ejecutando cuando la señal ocurre hacia el manejador de señales.

Esto, en sí mismo, es un bug con un impacto mínimo aparentemente. Antes de 4.3, GCC podía emitir un código de operación `cld` (clear direction flag) previamente al uso de cadenas *inline* o operaciones de memoria, por lo que estas operaciones podían empezar desde un estado conocido. En GCC 4.3, depende del mandato ABI que la bandera de dirección sea limpiada antes de la entrada a una

función, significando esto que el kernel necesita organizar este hecho antes de la llamada a un manejador de señales. Esto actualmente no es así, pero un pequeño parche lo corrige.

Este problema ha existido durante 15 años; GCC siempre ha emitido el código que funcionaba correctamente en dichos Kernels que no seguían el estándar ABI, hasta ahora. El principal problema es que hay un gran número de kernels instalados que son vulnerables a este problema, pero solo si GCC 4.3.0 está instalado en dicho equipo. Esta versión de GCC no está aún en uso oficial, por lo que se piensa que se podría revertir este comportamiento antes de que se incluya esta última versión en las distribuciones. Manejar estos bugs puede ser muy difícil y consumir mucho tiempo. Es por ello por lo que aunque el comportamiento de GCC es correcto y el kernel es el que está corrupto, podría ser muy útil volver atrás en este cambio, quizás proporcionando el nuevo comportamiento por medio de un argumento en la línea de comandos para aquellos quienes están seguros que sus binarios correrán en los kernels parcheados con el patch existente comentado anteriormente.

4. Conclusiones

Como hemos visto en este documento, existe un gran número de organizaciones y personas que trabajan en torno al sistema llamado comúnmente GNU/Linux. A pesar de que el núcleo de Linux lleva cerca de dos décadas en desarrollo y que en él contribuyen muchas personas, hoy día se siguen encontrando vulnerabilidades y problemas de seguridad en el mismo, pero por suerte estos problemas son solventados incluso antes de que nos enteremos de que han existido.

Al ser el código del kernel de Linux abierto y darse el hecho de que cualquier persona del mundo puede contribuir en el mismo, este sistema da un grado de solvencia y seguridad mucho mayor que otros sistemas operativos privados. Gracias a esto no siempre que se descubre un bug viene dado por el hecho de que alguien ha intentado explotarlo de alguna manera o lo ha descubierto alguno de los desarrolladores habituales, sino que se dan casos donde terceras personas descubren dichas vulnerabilidades, las comunican y son reparadas lo más rápido posible. Incluso estas terceras personas si tienen el conocimiento suficiente acerca del tema pueden corregir por si mismos la vulnerabilidad, debido a que disponen del código completo del kernel para modificarlo a su antojo.

El proceso de desarrollo del kernel de Linux como hemos visto anteriormente está en pleno desarrollo, y aproximadamente cada semana aparecen parches que incluyen pequeños arreglos o incluyen nuevas características que mejoran la experiencia del usuario final. La existencia de diferentes versiones del kernel disponibles en multitud de repositorios hace que un usuario pueda inclinarse hacia distintas formas de uso de su equipo dependiendo del objetivo de uso del mismo. Si por ejemplo se va montar un servidor Web, quizás no interese disponer de las últimas características en cuanto funcionalidad y se opta por una versión anterior del Kernel etiquetada como “estable” que nos asegure que va a funcionar sin ningún tipo de problema. Si por el contrario queremos participar de forma activa en el

desarrollo y testeo del kernel podemos bajarnos la última versión de alguna de las bifurcaciones del kernel especializadas a dicho objetivo, como la de Andrew Morton.

En los sistemas operativo privativos este caso no se suele dar, vamos a disponer generalmente de una última y única versión del sistema, que incluirá las mismas características y vulnerabilidades en todos los equipos donde esté instalado. Ya que el código de los núcleos respectivos no se muestran en ninguna circunstancia, no hay posibilidad de que terceras personas participen de forma voluntaria en el desarrollo del mismo, y en cuanto se descubre alguna vulnerabilidad esta suele ser explotada. Además, al ser mucho menor el número de desarrolladores activos en dichos sistemas el proceso de corrección de estos errores es notablemente más lento.

Referencias

- [1] «¿Qué es un exploit?» Internet. [Último acceso: Abril 2008].
<http://www.elhacker.net/exploits/>
- [2] «Bug en kernel linux permite ser root». Internet, 2008. [Último acceso: Abril 2008].
<http://www.kriptopolis.org/bug-kernel-linux>
- [3] «Distributions and security updates». Internet, 2008. [Último acceso: Abril 2008].
<http://distrowatch.com/weekly.php?issue=20080218>
- [4] «Patching CVE-2008-0600, Local Root Exploit». Internet, 2008. [Último acceso: Abril 2008].
http://kerneltrap.org/Linux/Patching_CVE-2008-0600_Local_Root_Exploit
- [5] ADSL.NET: «Skype probablemente atacado por un exploit DoS». Internet, 2007. [Último acceso: Abril 2008].
<http://www.adslnet.es/index.php/2007/08/18/skype-probablemente-atacado-por-un-exploit-dos/>
- [6] BNLUG: «YaBB Forum Profile CRLF Injection Remote Privilege Escalation Vulnerability». Internet, 2007. [Último acceso: Abril 2008].
http://www.bnlug.org/vulnerability/24455/YaBB_Forum_Profile_CRLF_Injection_Remote_Privilege_Escalation_Vulnerability
- [7] CIARROCCHI, PAOLO: «Introduction to linux kernel development process». Internet, 2005. [Último acceso: Abril 2008].
<http://paolo.ciarrocchi.googlepages.com/introductiontolinuxkerneldevelopmentprocess>
- [8] HELPNETSECURITY: «Jinzora Multiple HTML Injection and Cross-Site Scripting Vulnerabilities». Internet, 2008. [Último acceso: Abril 2008].
<http://www.net-security.org/vuln.php?id=4627>

-
- [9] LINUX-CROSS-REFERENCE: «linux/Documentation/SubmittingPatches». Internet. [Último acceso: Abril 2008].
<http://lxr.linux.no/linux+v2.6.24.4/Documentation/SubmittingPatches>
- [10] LINUXJOURNAL: «Contributing to the Linux Kernel». Internet, 2000. [Último acceso: Abril 2008].
<http://www.linuxjournal.com/article/3991>
- [11] LWN.NET: «GCC 4.3.0 exposes a kernel bug». Internet, 2008. [Último acceso: Abril 2008].
<http://lwn.net/Articles/272048/>
- [12] PURCZYNSKI: «Vmsplice() system call vulnerability». Internet, 2008. [Último acceso: Abril 2008].
<http://www.nic.com/~dave/SecurityAdminGuide/SecurityAdminGuide.html>
- [13] SECURITYFOCUS: «phpBB Multiple HTML Injection Vulnerabilities». Internet, 2007. [Último acceso: Abril 2008].
<http://www.securityfocus.com/bid/18931>
- [14] SECURITYLAB: «Skype Network Remote DoS Exploit». Internet, 2007. [Último acceso: Abril 2008].
<http://en.securitylab.ru/poc/301420.php>
- [15] SECURITYFOCUS: «Drupal Multiple HTML Injection Vulnerabilities». Internet, 2008. [Último acceso: Abril 2008].
<http://www.securityfocus.com/bid/28026>
- [16] WIKIPEDIA: «Exploit». Internet. [Último acceso: Abril 2008].
<http://es.wikipedia.org/wiki/Exploit>
- [17] ——. «History of Linux». Internet. [Último acceso: Abril 2008].
http://en.wikipedia.org/wiki/History_of_the_Linux_kernel
- [18] WRESKI, DAVE: «Linux Security Administrator's Guide». Internet, 1998. [Último acceso: Abril 2008].
<http://www.nic.com/~dave/SecurityAdminGuide/SecurityAdminGuide.html>