

Control de Logs en GNU/Linux

Este artículo es bastante teórico, siento que sea así, pero es que la teoría es necesaria para luego saber aplicarla a la práctica, el borrado/edición de huellas :-). Aun así espero que sea lo más ameno posible ;)

Como partimos desde cero, voy a empezar explicando lo más básico... Si ya sabes lo que son los logs y donde se guardan, sáltate este apartado.

1. ¿Qué son los ficheros log?

Bien, supongo que la mayoría ya lo sabréis, o habréis oído hablar de ellos, pero siguiendo la filosofía de la revista vamos a explicarlo **todo** paso a paso.

Los ficheros log se encargan de registrar los eventos que ocurren en el sistema, aplicaciones incluidas. ¿Qué eventos? Pueden ser accesos al sistema, mensajes de información del kernel, el tráfico capturado por un sniffer e incluso los famosos logs del IRC, TODO esto son ficheros log, o en cristiano, ficheros de registro. Normalmente “ficheros log” lo solemos recortar a simplemente **logs**, que es como nos referiremos a ellos de aquí en adelante.

El usuario doméstico suele pensar que los logs no tienen ninguna utilidad, que para qué los van a revisar si a él no le van a comprometer el sistema... O simplemente no sabe de su existencia... El objetivo de este artículo es abrirles los ojos a los usuarios para que vean que los logs son de mucha utilidad y muy potentes.

Ficheros logs hay de dos tipos, los de **texto plano** y los **binarios**, que dependen de aplicaciones externas para ver la información que contienen.

2. ¿Dónde se guardan los logs?

En general, se pueden guardar donde el usuario quiera, algunas aplicaciones los guarda en una ruta determinada por defecto, pero casi siempre podremos indicarle a la aplicación la ruta donde queremos guardar los logs, bien mediante parámetros o bien mediante un fichero de configuración. El demonio **Syslogd** también maneja las rutas de logueo, pero esto lo veremos en su correspondiente apartado.

Como hemos dicho en este último párrafo, los logs se pueden guardar donde el usuario quiera, pero normalmente se usa un directorio “estándar” para almacenarlos, que es el **/var/log**. Dentro de este directorio suelen estar los logs del sistema por defecto. Algunas distros de GNU/LiNux usan otro directorio, el **/var/adm**.

```
root@Monpe:/var/log # ls
1
acpid
acpid.1.gz
apache2
aptitude
auth.log
auth.log.0
base-config.log.1
base-config.timings.1
btmp
cups
daemon.log
daemon.log.0
debian-installer
debug
debug.0
root@Monpe:/var/log # screenshot!

dmesg
evms-engine.1.log
evms-engine.2.log
evms-engine.3.log
evms-engine.4.log
evms-engine.5.log
evms-engine.6.log
evms-engine.7.log
evms-engine.8.log
evms-engine.9.log
evms-engine.log
exim4
fontconfig.log
gdm
kern.log
kern.log.0

kismet
ksymoops
lastlog
lpr.log
mail.err
mail.info
mail.info.0
mail.log
mail.log.0
mail.warn
messages
messages.0
mysql
mysql.err
mysql.log
nessus

news
screen1.png
scrollkeeper.log
scrollkeeper.log.1
syslog
syslog.0
user.log
user.log.0
uucp.log
wtmp
XFree86.0.log
XFree86.0.log.old
Xorg.0.log
Xorg.0.log.old
Xorg.20.log
```

3. Los ficheros log: cuales son y una breve explicación

Ahora vamos a entrar en materia ya... Vamos a ver los ficheros log más comunes de GNU/Linux y Unix en general, junto a una breve explicación de cada uno de ellos.

Puede que algunos de los logs que nombraré no aparezcan en vuestra distro, al menos por defecto. Cada distro es diferente y algunas toman políticas distintas respecto a la localización de los logs, yo voy a nombrar los que más se ven en la mayoría de distros, si alguno no aparece no te preocupes, sigue leyendo ;)

Los clasificamos en dos apartados: **Ficheros log de texto plano** y **Ficheros log binarios**.

3.1. Ficheros log de texto plano

- **/var/log/syslog**: En este fichero log se almacenan todos los logs por defecto. En teoría es el más importante, aquí estará todo lo que busquemos, al menos si no hemos cambiado nada del demonio Syslogd, que más adelante veremos.
- **/var/log/messages**: Contiene mensajes del sistema, no es un log "alerta", es más bien informativo. Viene bien para arreglar problemas en el sistema.
- **/var/log/secure**: En este fichero están los logs de todas las conexiones que se realizan hacia nuestra máquina.
- **/var/log/auth.log**: Contiene el registro de todas las autenticaciones en el sistema, tanto las aceptadas como las fallidas.
- **/var/log/debug**: Aquí es donde se almacena la información de depuración de los programas que ejecutamos en el sistema. Esta información la envía el núcleo del sistema (Kernel) o bien los propios programas.
- **/var/log/kern.log**: Aquí están los mensajes que proceden del Kernel.
- **/var/log/daemon.log**: Contiene los logs de los demonios que se cargan en el sistema.
- **/var/log/mail.log**: En caso de tener un servidor SMTP en el equipo, nos muestra información de todos los mensajes que entran y salen por nuestro servidor.
- **/var/log/boot.log**: Contiene los mensajes de arranque del sistema.
- **/var/log/loginlog**: Este fichero log contiene un registro de todos los intentos de login fallidos. Solo se registran si se han realizado 5 o más intentos, aunque este valor se puede cambiar en el fichero **/var/default/login**, en la entrada **RETRIES**.

- **/var/log/sulog**: Contiene la información acerca de las ejecuciones de la orden **su**. Indica la fecha, la hora, el usuario que lanza el comando **su** y el login que adopta, la terminal asociada y si se ha realizado con éxito (+) o ha fracasado (-). En algunas distros de linux es necesario editar el fichero **/etc/login.defs** y marcar que si queremos loguear los registros de **su**.
- **/home/user/.bash_history**: Este fichero log se suele pasar por alto, al no estar donde los demás, pero lo cierto es que puede contener información muy sensible. Lo que contiene este fichero es un historial de todos los comandos que hemos ejecutado en nuestra shell **bash**.
- **/home/user/.mysql_history**: Este también es muy peligroso, encontraremos este fichero log si tenemos instalado el cliente de MySQL. Contiene todas las órdenes introducidas en el cliente MySQL, por ejemplo si introducimos una orden para cambiar la contraseña del usuario root de una base de datos, en este fichero quedará almacenada la orden, incluida la nueva contraseña en texto plano.

3.2. Ficheros log binarios

- **/var/log/wtmp**: Almacena toda la información de conexiones y desconexiones al sistema. La estructura de este fichero es *utmp*. Contiene información como el nombre de usuario, por donde accede, el origen y la hora de acceso. Al ser un fichero binario, para verlo se necesita una aplicación externa, en este caso se trata del comando **last**.
- **/var/log/utmp**: Contiene información de todos los usuarios conectados. No incluye los usuarios que están conectados a servicios como FTP, IMAP, etc, ya que estos servicios no utilizan utmp para loguear. Como en el caso anterior y en todos los binarios, para verlo es necesario teclear el comando **w** o **who**, que dará una salida con los logs de los usuarios conectados, la terminal por la que se han conectado, la fecha y la hora.
- **/var/log/lastlog**: Contiene la fecha y la hora de la última conexión de cada usuario existente en el sistema. Si un usuario nunca se ha logueado, en vez de salir la fecha y la hora de la última conexión saldrá el mensaje ****Nunca ha entrado****. La aplicación para ver la información de este log es el comando **lastlog**.
- **/var/log/faillog**: Contiene los intentos de login fallidos en el sistema. Es muy parecido al anterior, pero este únicamente muestra los intentos de login fallidos. Para verlo ejecutamos el comando **faillog**.
- **Acct (o pacct)**: Registra los comandos ejecutados por todos los usuarios. Solamente los comandos, los argumentos no. Funciona si está activado el proceso *"accounting"*, que se activa mediante el comando **accton**. Si nos dice que no encuentra el comando, es que no lo tenemos instalado, para instalarlo en una Debian tecleamos *"apt-get install acct & accton"*, si no tiraremos de las fuentes. Este log es bastante eficaz, por ejemplo aunque el atacante pare el proceso **acct**, en el log aparecerá el comando ejecutado por el atacante para pararlo. El único inconveniente es que si usamos bastante la máquina, este log se hará enorme, con las consecuencias que ello conlleva. Para ver estos logs es necesario ejecutar el comando **lastcomm** o **acctomm**.

Bien una vez explicados los ficheros log más importantes de linux, vamos a hablar del archiconocido demonio **Syslogd**.

4. El demonio Syslogd

¿Qué es Syslogd? **Syslogd** es el demonio de **Syslog**. Y ¿qué es Syslog? Syslog es un estándar *de facto* para controlar, redireccionar y administrar mensajes log en una red TCP/IP. El término "syslog" se utiliza indistintamente tanto para el protocolo syslog como para la librería o aplicación encargada del envío de los mensajes. Syslog se suele utilizar para la auditoría y administración de sistemas.

El protocolo Syslog es muy simple, el emisor de syslog envía mensajes de texto cortos al receptor de syslog. El receptor es el que hemos mencionado antes, el demonio Syslogd. Estos mensajes son enviados a través del protocolo UDP en texto plano.

Resumiendo, Syslogd es el demonio encargado de loguear los eventos que ocurren en el sistema.

Syslog tiene otro demonio más, aparte del mencionado Syslogd, es el **klogd**. De momento no lo vamos a ver, con saber que es el demonio que se encarga exclusivamente de los mensajes del kernel nos basta, de todas formas si estáis interesados... *man klogd* ;-).

Bien, espero que hayan quedado claras las diferencias entre Syslogd y syslog, ahora vamos a lo fuerte, vamos a ver como configurar Syslogd.

4.1. Configurando Syslogd



```

# /etc/syslog.conf      Configuration file for syslogd.
#
#       For more information see syslog.conf(5)
#       manpage.
#
# First some standard logfiles.  Log by facility.
#
auth,authpriv.*      /var/log/auth.log
*.*,auth,authpriv.none -/var/log/syslog
#cron.*              /var/log/cron.log
daemon.*             /var/log/daemon.log
kern.*               /var/log/kern.log
lpr.*                /var/log/lpr.log
mail.*              /var/log/mail.log
user.*              /var/log/user.log
uucp.*              /var/log/uucp.log

```

El demonio Syslogd se configura mediante reglas. ¿Dónde se colocan estas reglas? En su fichero de configuración, que está por defecto en **/var/log/syslog.conf**. Vamos a coger nuestro visor/editor de texto plano favorito y abrimos este fichero para ver su aspecto. Al principio tanta línea puede parecer complicado, pero al final del artículo sabremos leer todas y cada una de ellas :-p.

Como vemos, hay bastantes reglas predefinidas y comentarios (las líneas que empiezan por #). Vemos que todas las reglas siguen un mismo patrón, dos campos separados por una o varias tabulaciones. El primer campo se llama **selector** y el segundo campo **acción**.

El **selector** a su vez consta de de las **facilities** y de las **priorities**, que ahora veremos. La **acción**, como su nombre indica, es la acción que se debe cumplir cuando se cumpla el selector. Normalmente la acción es la ruta hacia un fichero log, pero también se puede enviar a los usuarios, a la impresora, a otro host... Un poco más adelante lo veremos.

Ahora si, vamos a ver que es eso de las facilities y las priorities, o en cristiano: las facilidades o servicios y las prioridades. Nos referiremos a ellas en inglés, ya que es como normalmente nos las encontraremos por la red.

Las facilities procuran identificar el programa que originó el/los mensaje/s log.

Las priorities clasifican el nivel de importancia del mensaje log.

Vamos a ver todas las facilities y priorities que disponemos:

Facilities:

- **kern**: Estos son los logs procedentes del kernel.
- **user**: Son los logs generados por procesos aleatorios del usuario.

- **mail**: Logs del sistema de correo, en caso de tener un servidor de correo instalado.
- **daemon**: Logs procedentes de los demonios del sistema.
- **authpriv y auth (este último en desuso)**: Logs de seguridad y del sistema de autorizaciones.
- **syslog**: Logs generados internamente por el demonio syslogd.
- **lpr**: Logs procedentes de las impresoras.
- **news**: Logs del sistema de noticias, en caso de tener un servidor de news instalado.
- **uucp**: Logs generados por el sistema UUCP. UUCP viene de Unix to Unix Copy, uno de los protocolos más antiguos y que originalmente se utilizaba para copiar ficheros entre sistemas Unix. Hoy en día casi no se utiliza.
- **cron**: Logs generados por el demonio CRON. Cron es el encargado de la programación de tareas periódicas, algo parecido a lo que hace el comando AT de los Windows.

Priorities:

- **debug**: Logs de depuración de un programa.
- **info**: Logs informativos.
- **notice**: Logs de sucesos significativos pero normales.
- **warn o warning**: Logs de advertencia.
- **err**: Logs de error
- **crit**: Logs que indican situaciones críticas.
- **alert**: Logs de alerta. Si nos encontramos uno de estos, más vale que actuemos rápidamente.
- **emerg**: El sistema se ha vuelto inoperable.

Los he ordenado de menor a mayor prioridad, *debug* es la prioridad mínima, mientras que *emerg* es la prioridad máxima.

Aquí pongo un ejemplo que nos servirá para tener esto un poco más claro. También lo utilizaremos como referencia para el resto del artículo, así que tenlo a la vista.

mail.warn news.warn;auth.*	/var/log/messages
kern mail.warning	-/var/log/kern_mail.log
kern mail.=warning	-/var/log/kern_mail_warn.log
kern.err kern.!=warning	/var/log/kern_warn_negado.log
*.alert	/var/log/alerts.log

Los selectores

Lo que vemos a la izquierda son los selectores. Dentro de los selectores, en **azul** tenemos las *facilities*, en **rojo** las *priorities*, y en **amarillo** los caracteres especiales, que ahora explicaré. Si has entendido todo lo explicado y eres buen observador verás que en casi todas las reglas hay 2 o más *facilities*, y en algunas dos *priorities*... Bueno, en los selectores se pueden concatenar las *facilities* y las *priorities* en una misma línea, si queremos por ejemplo que dos *facilities* con *priorities* diferentes se envíen a un mismo fichero log no tenemos más que concatenarlos con un punto y coma (carácter especial), como la primera línea. En la segunda línea vemos dos *facilities* pero no están separadas por punto y coma, si no por una coma simple (otro carácter especial), esto indica que para los dos se usará la misma prioridad, ahora lo veremos más detalladamente.

Un apunte importante... cuando se especifica una prioridad estamos diciendo “esa prioridad y sus **inferiores**”, por ejemplo, si especificamos la prioridad *notice* registrará también los mensajes con prioridad *debug* e *info*, así con todos. Si queremos evitar esto, debemos usar el carácter especial igual (=), como veremos en el próximo apartado.

Caracteres especiales

Los caracteres especiales nos permiten aprovechar más la estructura de las reglas, vamos a ver los tipos de caracteres especiales que hay, su función y algún ejemplo.

- El asterisco (*): El asterisco se emplea como “comodín”. Cuando aparece un asterisco quiere decir que se refiere a todas las facilities o todas las priorities dependiendo de la posición en la que aparezca. Como ejemplo vamos a la figura 3 y vemos que la primera línea tiene el asterisco al final. Como está **detrás** del punto que separa las facilities de las priorities, sabemos que ese asterisco hace de comodín para las priorities, es decir lo que hacemos con ese comodín es indicar “Todas las priorities”. En la última línea de la misma figura vemos también como aparece otro asterisco, pero esta vez **delante** del punto, por lo que indica “Todas las facilities”. Explicar esto es un poco engorroso, pero una vez lo pillas es de lo más útil, ya que el asterisco como comodín no solo se utiliza aquí, seguro que en la línea de comandos lo has utilizado más de una vez para referirte a un archivo cuando no sabes su nombre completo :P

- El punto y coma (;): Como dije antes, este carácter especial se utiliza para concatenar varias facilities+priorities en una misma línea. Ejemplo: La primera línea del esquema 1 aparecen 3 facilities y 3 priorities (recuerda que el asterisco del final es comodín ;)), como vemos están concatenadas mediante un punto y coma. Como ya dije, la utilidad de concatenar varias facilities y priorities es la de dirigir las a un mismo destino, en este caso a un mismo archivo.

- La coma (,): La coma sirve para especificar a múltiples facilities una misma prioridad. En la segunda línea del esquema 1 vemos como hay dos facilities separadas por una **coma** y detrás del punto la prioridad de esos dos facilities.

- El espacio (): El espacio indica que no hay priorities definidas para las facilities.

- El igual (=): Con el = delante de una prioridad indicamos que solo se almacenen los mensajes con una prioridad determinada, no incluyendo sus inferiores. En la tercera línea del esquema 1 vemos el carácter especial “=” delante de la prioridad, lo que estamos diciendo es que se almacenen los mensajes de las facilities *kern* y *mail* con prioridad *warning* y solamente *warning*! :P Nada de priorities inferiores...

- El signo de exclamación (!): Con el ! delante de la prioridad indicamos que queremos exceptuar esa prioridad y sus inferiores. Lo podemos combinar con el **igual**, de tal forma que quede “!=”, con esta combinación lo que hacemos es exceptuar esa prioridad pero **no** sus inferiores. Como ejemplo, en la cuarta línea del esquema 1 vemos como con la combinación de estos caracteres especiales decimos que se logueen los mensajes del kernel con prioridad *info* e inferiores pero que **no** se registren los que tengan prioridad *warning* exclusivamente, ya que lleva el símbolo “=” combinado con “!”.

Las acciones

Siguiendo el esquema 1, vemos a la derecha el campo **acción**, separado por tabulaciones del campo selector. El campo acción tiene en **verde** la acción a hacer en caso de que se cumpla el selector y en algunas reglas tiene un guión (-), color **turquesa**. Este guión indica que no se sincroniza cada vez que existe una entrada en ese log, así que si el sistema se cae podemos perder datos. La ventaja que conseguimos con esto es una mejora en la velocidad, sobre todo si hay muchas aplicaciones enviando logs a Syslogd.

Voy a poner un nuevo esquema, con acciones que no hemos visto hasta ahora:

```
kern,mail.info      /var/log/messages
auth.*              /dev/tty2
kern.crit            /dev/console
*.=alert            root, ivan
*=emerg              *
```



```
.warn /dev/lp1
.info @nombre.del.host.remoto
```

Figura 4 – Esquema 2

Antes dije que la acción normalmente era la ruta a un archivo log, pero como también dije no es así, podemos enviarlo a otros medios, a continuación veremos todos los medios junto a una breve explicación de como hacerlo y un ejemplo (como no :P):

– **A un fichero log de texto plano:** Sobran las palabras creo, no? :) Lo que hemos estado viendo todo el rato :)

– **A una terminal o a la consola:** Syslogd nos ofrece la posibilidad de enviar los logs directamente a una terminal o a la consola con la que estamos trabajando, de forma que podríamos ver los logs “*in situ*”. La verdad es que fastidia un poco que te lleguen mensajitos mientras estás listando los archivos de tu home por ejemplo... pero seguro que los ves! :P. En la segunda y tercera línea del esquema 2 vemos un ejemplo de esto, la regla de la línea 2 envía los logs a la terminal 2 (tty), y la regla de la línea 3 los envía a la consola con la que estamos trabajando (/dev/console).

– **A un usuario (o a todos):** Esto es simple, se especifica solo el usuario o los usuarios a los que queremos enviarle los logs y se le mostrará en la consola. Si queremos enviar los logs a todos los usuarios, lo hacemos con el asterisco “*”. En el ejemplo del esquema vemos en la línea 4 como envía todos los logs del kernel con prioridad *crítica* a los usuarios “root” e “ivan”. En la línea 5 vemos como envía todos los logs con prioridad *emerg* a **todos** los usuarios.

– **A la impresora:** Si activamos el modo paranoico podemos enviar los logs directamente a la impresora, esto en mi opinión es un poco paranoico, pero por seguridad que no falte :P. Es similar a enviarlos a una terminal, pero a la impresora. En la sexta línea del esquema 2 vemos como todos los logs con prioridad warn e inferiores se envían al dispositivo /dev/lp1, que viene a ser la impresora.

– **A otro programa através de un fichero FIFO (tubería con nombre):** Podemos enviar los logs a otro programa através de un fichero FIFO (First In First Out, lo Primero que Entra es lo Primero que Sale), también llamado tubería con nombre o fichero de tipo PIPE. Los ficheros FIFO son bastante comunes en sistemas Unix, la función de estos ficheros es actuar de intermediario entre dos programas, un programa envía los datos al fichero FIFO y éste los almacena temporalmente, para poder ser recogidos después por un segundo programa. Por ejemplo, la salida de un programa servirá de entrada de otro programa. Cabe destacar de los ficheros FIFO que los datos que le llegan no se guardan en el disco duro, si no en un buffer, cuando el segundo programa recoge los datos se vacía ese buffer y sigue a la espera de nuevos datos. En nuestro caso lo que hacemos es enviar los logs a un archivo FIFO para que otro programa externo pueda recogerlos y tratar con ellos. Para aclarar esto un poco, vamos a ver **logcolorise.pl**, un script en Perl que recogerá los logs de un PIPE y los redirigirá a una terminal, pero con un cambio, formateará los logs y saldrán de colores, para que sean más legibles y más bonitos :P.

Lo primero que tenemos que hacer es bajarnos el script **logcolorise.pl**, su web oficial no va, yo lo he bajado de la primera url que me ha salido en google, <http://linuxbrit.co.uk/downloads/logcolorise.pl>. Lo descargais con wget en /usr/bin, y le dais permisos de ejecución con **chmod +x /usr/bin/logcolorise.pl**. Ahora seguimos los pasos que muestro a continuación, siempre como root. He puesto un comentario en cada orden para explicar que es lo que hace cada una, empieza por una almohadilla y está en color **turquesa**, eso no lo copieis!

```
mkdir /var/log/pipes #creamos el directorio pipes en /var/log
```

```
mkfifo -m 600 /var/log/pipes/autenticaciones #creamos un archivo FIFO llamado autenticaciones con chmod 600
mkfifo -m 600 /var/log/pipes/todo #idem del anterior, pero llamado "todo"
```

Ahora editamos el fichero de configuración `/etc/syslog.conf` y creamos las reglas para que nos envíe los logs a los archivos FIFO que hemos creado, esto lo hacemos poniendo el símbolo de pipe (`|`) antes de la ruta del archivo FIFO. El carácter "`|`" se pone con Altgr+1.

```
auth.* | /var/log/pipes/autenticaciones
*. * | /var/log/pipes/todo
```

Con estas reglas estamos diciendo que envíe los logs de la facility `auth` con todas las prioridades al archivo FIFO situado en `/var/log/pipes/autenticaciones`, y que los logs que vengan de todas las facilities con todas las prioridades (todos los logs) los envíe al archivo FIFO `/var/log/pipes/todo`.

Ahora haremos un script de inicio para que se ejecute al arrancar el SO, con una orden que hará que logcolorise.pl lea los logs del archivo FIFO `/var/log/pipes/todo` y lo imprima en la terminal tty3 todo colorido :). Supongo que ya sabreis como hacer scripts de inicio... No? Joe... Bueno, lo explico para sistemas Debian o basados en él, si usais otra distro buskais por el foro como hacerlo o en google :). Primero creamos el bash-script en `/etc/init.d/` con el nombre "`logsdecolores`" :P El archivo contendrá:

```
#!/bin/bash
/usr/bin/logcolorise.pl /var/log/pipes/todo >/dev/tty3&
```

Supongo que lo entenderéis, porque en el párrafo anterior he dicho lo que hacía... Venga, guardamos el bash-script y le damos permisos de ejecución con "`chmod +x /etc/init.d/logsdecolores`". Después ejecutamos la orden "`update-rc.d logsdecolores defaults`" y si no da ningún error ya tenemos el script para que se ejecute al inicio del SO.

Ahora toca probar el sistema, si has seguido todos los pasos exáctamente como he descrito saldrá bien. Podemos hacer dos cosas, reiniciar el SO para que el mismo levante los servicios, o levatarlos nosotros y no reiniciar... Si habeis elegido esto último, aquí los pasos para hacerlo (como root):

Miramos el PID de syslogd (`ps aux | grep syslogd`), matamos el proceso (`kill PID`) y volvemos a ejecutar `syslogd`, esto lo hacemos para que `syslogd` cargue las nuevas reglas que hemos añadido. Después ejecutamos el script `/etc/init.d/logsdecolores` y ya lo tenemos todo preparado, pero antes de nada abrimos una consola y nos logueamos y deslogueamos varias veces como root, metemos la password mal adrede... y por fin, vamos a la terminal tty3 (Ctrl+Alt+F3) y... Ohh! Se ven los logs coloridos!!! Si volvemos al entorno gráfico (Ctrl+Alt+F7), volvemos a loguearnos y a desloguearnos unas cuantas veces y vamos a la terminal tty3, veremos que aparecen los nuevos logs, esto es gracias a que logcolorise.pl está ejecutado en el background a la espera de que entren nuevos logs al fichero FIFO `/var/log/pipes/todo` para recogerlos e imprimirlos :).

Espero que esto se haya entendido, es muy importante, si no lo habeis entendido a la primera volverlo a leer, si aun no lo acabas de entender, no dudes en preguntarlo en el foro, que yo o quien sepa te responderá gustosamente :) La url del foro es: <http://www.hackxcrack.com/phpBB2/index.php>.

– **A otro host:** Esta es de las mejores opciones. Los logs se pueden enviar a otro host, esto nos la gran ventaja de que si comprometen el servidor, los logs no estarán almacenados en él, si no en otro host destinado exclusivamente al almacenamiento de logs, aunque esto no tiene porque ser así, pero es lo ideal para la seguridad, ya que el

atacante tendría que comprometer también el host que almacena los logs, que podría ser un equipo con los mínimos servicios corriendo, una buena política de seguridad y un buen firewall, vamos que sería difícil hackearlo ;)

Para que el host remoto pueda recibir los logs tiene que ejecutar *syslogd* con la opción “-r”. También hay que tener en cuenta que para recibir los logs tiene que aceptar todo lo que entre por el puerto 514 de UDP, así que a configurar los firewalls toca!

También hay que configurar los dos *syslogd* para procesar bien los logs y que vayan a donde tienen que ir. En el servidor tenemos que redirigir los logs al host encargado de almacenar los logs, para hacer esto creamos una regla cualquiera pero en el campo acción ponemos una arroba (@) y el nombre del host remoto, previamente definido en */etc/hosts* para más seguridad. En el host remoto tenemos que crear las reglas para que actúe consecuentemente, creamos la misma regla que en el servidor pero en el campo acción ponemos el destino que se nos antoje, a una terminal, a un archivo log... Vamos a hacer una práctica para que quede más claro.

Tenemos un servidor de páginas web llamado WebServer. También tenemos un host encargado de almacenar los logs de WebServer, este host se llama Logger y tiene la IP 192.168.0.3.

En el WEBSEVER, editamos el fichero */etc/hosts* y añadimos al final la IP del Logger que es 192.168.0.3 y el nombre, que viene a ser “Logger”, guardamos, salimos del editor y le hacemos un ping a Logger para ver si lo hemos hecho bien, “*ping -c 2 Logger*”. Ahora editamos el fichero */etc/syslog.conf* y creamos una nueva regla, por ejemplo una que envíe todos los logs de facility *auth* y todas las prioridades al equipo Logger. Quedaría así el */etc/syslog.conf*:

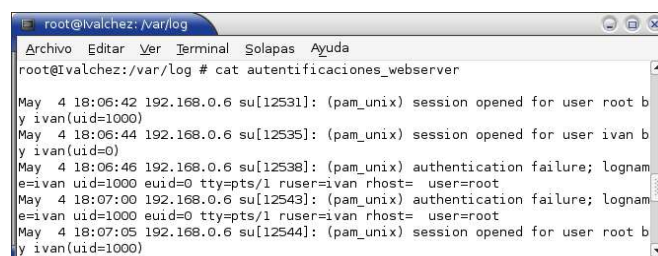
```
[...]
auth.* @Logger
```

Bien, ahora vamos al host Logger y editamos también el fichero */etc/syslog.conf* para que envíe todo lo que venga de *auth.** al fichero */var/log/autenticaciones_webserver*. Quedaría así:

```
[...]
auth.* /var/log/autenticaciones_webserver
```

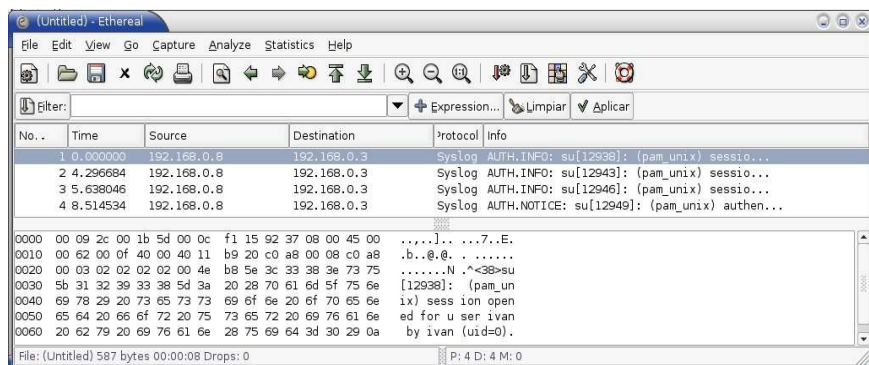
Ya tenemos los dos *syslogd* configurados, ahora falta reiniciar *syslogd*, pero en el host remoto lo iniciaremos con la opción “-r”. Como root, en el WebServer buscamos el PID de *syslogd* como antes dije que se hacía, lo matamos y volvemos a ejecutar “*syslogd*”. En el host Logger lo mismo, pero lo ejecutamos con “*syslogd -r*”. Y ya está!! Ahora en el WebServer hacemos un par de logins fallidos y no fallidos, vamos al Logger y le hacemos un cat al fichero */var/log/autenticaciones_webserver* y ohh! Salen las autenticaciones, indicando que proviene de nuestra IP.

Pero, ¿qué pasa si en el host Logger nos autenticamos, también se logueará? Pues sí, pero en lugar de poner la IP del WebServer pondrá el nombre de la máquina local, aquí una captura:



```
root@lvalchez: /var/log
Archivo Editar Ver Terminal Solapas Ayuda
root@lvalchez:/var/log # cat autenticaciones_webserver
May  4 18:06:42 192.168.0.6 su[12531]: (pam_unix) session opened for user root b
y ivan(uid=1000)
May  4 18:06:44 192.168.0.6 su[12535]: (pam_unix) session opened for user ivan b
y ivan(uid=0)
May  4 18:06:46 192.168.0.6 su[12538]: (pam_unix) authentication failure; lognam
e=ivan uid=1000 euid=0 tty=pts/1 ruser=ivan rhost= user=root
May  4 18:07:00 192.168.0.6 su[12543]: (pam_unix) authentication failure; lognam
e=ivan uid=1000 euid=0 tty=pts/1 ruser=ivan rhost= user=root
May  4 18:07:05 192.168.0.6 su[12544]: (pam_unix) session opened for user root b
y ivan(uid=1000)
```

Una cosa interesante también es que si ponemos un sniffer a la escucha, cuando hagamos logins veremos los paquetes que el WebServer manda al Logger :)



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.8	192.168.0.3	Syslog	AUTH.INFO: su[12938]: (pam_unix) sessio...
2	4.296694	192.168.0.8	192.168.0.3	Syslog	AUTH.INFO: su[12943]: (pam_unix) sessio...
3	5.638046	192.168.0.8	192.168.0.3	Syslog	AUTH.INFO: su[12946]: (pam_unix) sessio...
4	8.514534	192.168.0.8	192.168.0.3	Syslog	AUTH.NOTICE: su[12949]: (pam_unix) authen...

0000 00 09 2c 00 1b 5d 00 0c f1 15 92 37 08 00 45 007..E.
0010 00 62 00 0f 40 00 40 11 b9 20 c0 a8 00 08 c0 a8 .b..@..@.....
0020 00 03 02 02 02 02 00 4e b8 5e 3c 33 38 3e 73 75N^<38>su
0030 5b 31 32 39 33 38 54 3a 20 28 70 61 6d 5f 75 6e [12938]: (pam_un
0040 69 78 29 20 73 65 73 73 69 6f 6e 20 6f 70 65 6e ix) sess ion open
0050 65 64 20 66 6f 72 20 75 73 65 72 20 69 76 61 6e ed for u ser ivan
0060 20 62 79 20 69 76 61 6e 28 75 69 64 3d 30 29 0a by ivan (uid=0).

Paquetes de Syslogd capturados con Ethereal

Figura 6

Esto da para algunas preguntas... ¿No se podrían “interceptar” los paquetes? Si, tanto si la red es compartida como si es conmutada (número 11 de la revista, artículo de ARP-Spoofing de moebius)... ¿Hay alguna manera de evitar esto? Pues si, una solución podría ser unir estos dos equipos con dos tarjetas de red aparte y cable cruzado, y la otra podría ser cifrando los paquetes, ¿Cómo? Con una combinación de lo que hemos visto en el apartado anterior (archivos FIFO) + SSH... Pero no me quiero extender más con esto, así que si estais interesados, a google!

4.2 – Enviando mensajes a Syslogd

Podemos enviar mensajes a Syslogd con el programa **logger**, puede servir si tenemos un shellscript y queremos que guarde logs, o si queremos hacer el tonto un rato y simular un ataque... lo que se os ocurra! Los logs actuarán según las reglas de syslogd. La sintaxis para enviar un mensaje desde una determinada facility y con una prioridad es: “**logger -p facility.priority “Mensaje”**”.

Esto aparte de por si queremos que un shellscript pueda loguear, no le veo mucha más utilidad, pero bueno seguro que vuestra imaginación es mejor que la mía :).

Con esto ya hemos acabado con Syslogd... ya era hora eh! Nada, ahora vamos a ver lo que supongo que estareis esperando... la edición y el borrado de logs!

5. Borrado y edición de Logs!

5.1 Zappers

Realmente esto no tiene mucho misterio, ya que tenemos muchas herramientas que nos facilitan la tarea de editar y borrar los logs, estas herramientas se llaman “zappers”.

Cuando los logs son de texto plano lo podemos hacer nosotros mismos, con cat y grep se pueden hacer maravillas (ahora veremos algún ejemplo), pero para los binarios lo tenemos más difícil, por eso necesitamos los zappers ;). Vamos a ver tres zappers, el **clear-1.3** del conjunto de utilidades **thc-uht** del grupo **THC**, el **0x333shadow** del grupo **0x333** y el **vanish2** de “Neo the Hacker”.

Para instalar **clear-1.3** debemos bajar el **thc-uht** (THC-UnixHackingTools) de <http://www.thc.org/download.php?t=r&f=thc-uht1.tgz> y descomprimirlo, una vez descomprimido veremos muchas utilidades, entre ellas habrá un archivo comprimido llamado “clear-1.3.tar.gz”, lo descomprimimos (**tar xvzf clear-1.3.tar.gz**) y nos creará una

carpeta llamada clear-1.3 con un Makefile, un README y dos archivos C (clear13a.c y clear13b.c). Tenemos que compilarlos, pero antes vamos a hacer unas pequeñas modificaciones a los archivos C poniendo la ruta correcta de nuestros archivos log. Para esto editamos primero el fichero clear13a.c y las constantes WTMP_NAME, UTMP_NAME y LASTLOG_NAME las cambiamos a la ruta donde tenemos los logs en nuestro sistema, normalmente */var/log* (para *wtmp* y *lastlog*) y */var/run/utmp* (para *utmp*). Nos quedaría así:



```
*clear13a.c
/* $Id$ in the tnc-magazine #3 ...
*****

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/file.h>
#include <fcntl.h>
#include <utmp.h>
#include <pwd.h>

/* all systems support utmp and wtmp */
#define WTMP_NAME "/var/log/wtmp"
#define UTMP_NAME "/var/run/utmp"
/* utmpx and wtmpx are present on Sys VR4 systems. solaris, irix */
/* #define WTMPX_NAME "/var/adm/wtmpx"
#define UTMPX_NAME "/var/adm/utmpx"
*/
/* lastlog is present on linux, sunos, solaris, irix, ultrix
   osf, digital unix. Not on hpux and bsd, freebsd and therelike. */
#define LASTLOG_NAME "/var/log/lastlog"

#define TMP_NAME "/tmp/.XthcX.tmp"

#ifdef LASTLOG_NAME
#include <lastlog.h>
#endif
#define WTMPX_NAME
```

Guardamos y cerramos. Con el clear13b.c hacemos lo mismo. Después vamos a la consola, entramos al directorio donde tenemos el clear-1.3 y tecleamos “make” para compilarlos. Si no hay ningún problema ya lo tendremos compilado. Os estareis preguntando que diferencia hay entre clear13a y clear13b... Pues bien, los dos borran los logs del usuario que le pases como argumento en la línea de comandos, pero clear13a borra TODAS las entradas de ese usuario, mientras que clear13b solo borra la última entrada. Uno puede ser útil para unas situaciones y el otro para otras. La sintaxis es muy sencilla, simplemente *./clear13[ab] nombre_de_usuario*, siendo [ab] el clear13a o el clear13b, lo dicho anteriormente.

Ejemplos:

```
./clear13b hackxcrack -> Borrará la última entrada del usuario hackxcrack.
./clear13a hackxcrack -> Borrará todas las entradas del usuario hackxcrack.
```

El **vanish2** es más completito, le pasas el usuario, el hostname y la IP que quieres borrar y los busca por todos esos ficheros, y cuando lo localiza... se lo carga :) Lo podeis bajar de <http://packetstorm.linuxsecurity.com/UNIX/penetration/log-wipers/vanish2.tgz>, lo único que habrá que editar el fichero vanish2.c y en todas las funciones exit() que salen añadir un 0 al paréntesis, si no dará error al compilarlo. Os lo podeis bajar ya compilado de www.hackxcrack.com/programas/vanish2.zip, pero lo he compilado con la ruta de mis logs de apache2, que es /var/log/apache2, si teneis esta misma ruta pues adelante si no editarlo a mano y añadirles los 0 a las funciones exit(). Su sintaxis es sencilla: “*./vanish2 <user> <host> <IP>*”, como root. Este zapper busca en los archivos WTMP, UTMP, lastlog, messages, secure, xferlog, maillog, warn, mail, httpd.access_log, and httpd.error_log.

Ejemplo:

```
./vanish2 hackxcrack HxC 212.124.113.53 -> Borrará todos los logs que contengan el
usuario hackxcrack, o el host HxC o la IP 212.124.113.53
```

El **0x333shadow** es bastante más completo que los dos anteriores, lo bajamos de <http://packetstormsecurity.org/UNIX/penetration/log-wipers/0x333shadow.tar.gz>, lo descomprimos y lo compilamos con `gcc 0x333shadow.c -o 0x333shadow -D Linux`. Es capaz de matar syslogd, y por defecto busca en los directorios `/var/log`, `/var/adm`, `/usr/adm`, `/var/mail`. El uso y sus opciones lo podeis ver vosotros mismos ejecutandolo sin ningún parámetro, pero lo voy a explicar de todas formas, por si las dudas :P

Su sintaxis es: `./0x333shadow [acción] -i [cadena] -l [segundos] -m [dir1/archivo1] [dir2/archivo2] [...]`

Acción: Pueden ser dos opciones, “-a” o “-b”. El primero limpia los directorios por defecto que tengan la cadena que especifiquemos. El segundo lo mismo que el primero, pero este solo limpiará los logs binarios (*utmp*, *wtmp*, *utmpx*, *wtmpx* y *lastlog*).

Cadena: Lo que queremos que borre en caso de que coincida esta cadena, por ejemplo nuestra IP.

Segundos: Da la posibilidad de que empiece la limpieza de logs pasados los segundos que especifiquemos, esto es útil porque podemos desloguearnos y esto hará que limpie el logout también.

Dir1/archivo1...: Gracias al parámetro -m podemos especificar archivos log que hayamos visto en el sistema y que el programa no “los conozca”, o si los logs están en otro directorio. Podemos especificar más de un archivo/directorio, separados por un espacio.

Ejemplos:

`./0x333shadow -i 212.124.113.53 -m /var/log/apache2/access.log ->` Borrará todos los logs de la IP 212.124.113.53 del archivo access.log del Apache2.

`./0x333shadow -b -i hackxcrack -l 30 ->` Borrará todos los logs del usuario hackxcrack en los archivos de log binarios cuando pasen 30 segundos.

`./0x333shadow -a -i hackxcrack ->` Borrará todo log que contenga la cadena “hackxcrack”.

5.2 Métodos caseros

Los logs de texto plano podemos modificarlos a mano, o desde un shellscript creado por nosotros, tan solo con programas típicos de unix **cat** o **grep**. Un ejemplo podría ser borrar los logs de Apache, podríamos hacer: `“cat /var/log/apache2/access.log | grep -v NUESTRAIP >/var/log/apache2/access.log”`, o para borrar nuestros intentos de logueo `“cat /var/log/auth.log | grep -v NUESTROUSER >/var/log/auth.log”`, y así con los que queramos. El parámetro “-v” de grep lo que hace es no mostrar las coincidencias.

Iván Alcaraz <ivanalcaraz@gmail.com>