IES Zaidín-Vergeles

Análisis Forense en Linux.

25 de marzo de 2021

Tabla de contenidos

CAPÍTULO 1

Introducción

Aunque en informática forense la mayor parte de los peritajes o los incident response que se afrontan se hacen sobre sistemas Windows es verdad que también pueden aparecer casos de otros sistemas operativos y es importante conocer su funcionamiento para tener claras algunas cuestiones referentes a las evidencias encontradas.

Es decir, por una parte, es importante conocer el sistema operativo sobre el que es necesario realizar el análisis forense pero también es importante seleccionar las herramientas a utilizar para este análisis. No por tener que estudiar un sistema Windows estamos obligados a utilizar, por poner un ejemplo, Autopsy en Windows.

Existen algunas distribuciones de Linux que pueden ser nuestras herramientas básicas para hacer los análisis forenses sobre cualquier sistema operativo. Además, en determinados casos, para hacer análisis sobre sistemas Linux es muy recomendable utilizar frameworks propios de ese SSOO ya orientados al forense.

CAPÍTULO 2

Sistema de Ficheros Linux

Linux es un sistema operativo muy utilizado, sobre todo en servidores, es rápido y muy fiable. Soporta gran variedad de sistemas de ficheros: ext2, ext3, ext4, ReiserFS, XFS, JFS, UFS, FAT, FAT32 o NTFS. Existen también múltiples distribuciones como Red Hat, Debian, Fedora, Ubuntu y otras. Según la complejidad del sistema de ficheros, tienen características como previsión de apagones, posibilidad de recuperar datos, indexación de búsquedas rápidas, reducción de la fragmentación para agilizar la lectura de datos, etc.

Hay varios tipos de sistemas de ficheros como ya se ha comentado:

- **ext2:** está obsoleto. Tiene una fragmentación muy baja, aunque es algo lento manejando archivos de gran tamaño.
- **ext3:** Es la versión mejorada de ext2, con previsión de pérdida de datos por fallos del disco apagones, introduce journaling. Es compatible con el sistema de ficheros ext2.
- **ext4:** Es la última versión de la familia de sistemas de ficheros ext. Sus principales ventajas radican en su eficiencia (menor uso de CPU, mejoras en la velocidad de lectura y escritura) y en la ampliación de los límites de tamaño de los ficheros, ahora de hasta 16TB, y del sistema de ficheros, que puede llegar a los 1024 PB (PetaBytes).
- **Swap:** Es el sistema de ficheros para la partición de intercambio de Linux. Todos los sistemas Linux necesitan una partición de este tipo para cargar los programas y no saturar la memoria RAM cuando se excede su capacidad.
- **Btrfs:** Es un sistema de archivos desarrollado por Oracle, con la participación de Red Hat, SUSE, Intel, entre otras. Tiene como objetivo la implementación de características avanzadas y al mismo tiempo se centra en la tolerancia a fallos, la reparación y la fácil administración. Hace uso de la funcionalidad Copy-on-Write (CoW), permite realizar instantáneas (snapshots) de solo lectura o modificables, incluye soporte nativo para los Sistemas de archivos multi-dispositivo y soporta la gestión de sub-volúmenes. Además, protege la información (datos y metadatos)

mediante sumas de comprobación (checksums), soporta compresión, optimizaciones para discos SSD, empaquetado eficiente de ficheros pequeños y muchas otras más.

- **ZFS:** Es un sistema de archivos y administrador de volúmenes desarrollado originalmente por Sun Microsystems para su sistema operativo Solaris. El significado original era "Zettabyte File System", destaca por su gran capacidad, integración de los conceptos anteriormente separados de sistema de ficheros y administrador de volúmenes en un solo producto, nueva estructura sobre el disco, sistemas de archivos ligeros y una administración de espacios de almacenamiento sencilla.

2.1 Estructura Ext*

El sistema de ficheros de Linux consta de varias partes importantes: El Superbloque contiene la información sobre el sistema de ficheros, la tabla de inodos es el equivalente a las entradas de la FAT y los bloques de datos (en Linux cada bloque es de 512 bytes o múltiplos de 512). Existe también el bloque de carga que está reservado para almacenar el programa que utiliza el sistema para gestionar el resto de las partes del sistema de ficheros.

No existen unidades físicas, son ficheros que hacen referencia a ellas, integrados en la estructura de ficheros como cualquier otro.

2.1.1 El superbloque

El superbloque es un bloque que contiene la información más relevante y describe al sistema de ficheros. Se encuentra en el offset fijo 1024 del disco y ocupa 1024 bytes.

El superbloque contiene una descripción del tamaño y forma del sistema de ficheros. Esta información permite usar y mantener dicho sistema de ficheros. En condiciones normales únicamente el superbloque situado en el grupo de bloques 0 se lee cuando el sistema de ficheros es montado, sin embargo cada grupo de bloques contiene una copia duplicada. Entre otra información el superbloque contiene:

- Número Mágico (s_magic): Permite al software que reliaza el montaje comprobar que se trata efectivamente del superbloque para un sistema de ficheros Ext2.
- Nivel de revisión (s_rev_level): Los números mayor y menor de revisión permiten identificar ciertas características del sistema de archivos que encuentran presentes

únicamente en ciertas versiones.

- Contador de montajes (s_mnt_count) y máximo número de montajes (s_max_mnt_count): Juntos permiten determinar si se debe realizar una comprobación completa del sistema de ficheros, "e2fsck".
- Número del grupo de bloques (s_block_group_nr): El número del grupo de bloques que contiene la copia del superbloque.
- Tamaño de bloque (s_log_block_size): El tamaño del bloque en bytes.
- Bloques por grupo (s_blocks_per_group): El número de bloques en un grupo. Igual que el tamaño de bloque, este valor se fija cuando se crea el sistema de ficheros.
- Bloques libres (s_free_blocks_count): El número de bloques libres en el sistema de ficheros.
- Inodos libres (s_free_inodes_count): El número de inodos libres en el sistema de ficheros.
- Primer Inodo (s_first_ino): El número del primer inodo del sistema de ficheros. El primer inodo en un sistema Ext2 raíz debe ser la entrada de directorio para el directorio "/".

2.1.2 Los Inodos

Un inodo es una estructura de datos que almacena información sobre un fichero de nuestro sistema de archivos.

Un inodo no tiene nombre y se identifica mediante un número entero único. Cada inodo únicamente puede contener datos de un solo fichero del sistema de archivos. Por lo tanto, si tenemos 4 archivos y 4 directorios estaremos usando 8 inodos.

Algunos de los sistemas de archivos que trabajan con inodos son: ext2/3/4, UFS, ReiserFS, FFS, XFS, Btrfs, etc. Windows y MacOS no trabajan con inodos porque sus sistemas de archivos son Ntfs y HFS+. Solo trabajan con inodos sistemas operativos como por ejemplo UNIX, FreeBSD, GNU-Linux y otros sistemas operativos basados en Unix.

Por cada fichero, Linux tiene asociado un elemento en la tabla de inodos que contiene un número. Este número identifica la ubicación del archivo dentro del área de datos.

Un inodo contiene la totalidad de metadatos de un fichero de nuestro sistema de archivos.

Los metadatos almacenados en un inodo son los siguientes:

- **Número de inodo**. El número de inodo es un número entero único que sirve para identificar un inodo.
- **Tamaño del fichero** así como el número de bloques que ocupa el fichero en el disco duro.
- El **dispositivo de almacenamiento** en que está almacenado el fichero. (Device ID)
- **Número de enlaces**. Por lo tanto si hay 2 archivos que apuntan a un mismo inodo tendremos 2 enlaces. Si tenemos un directorio que contiene 15 archivos tendremos 15 enlaces.

- El **identificador de usuario (UID o User ID)**. Por lo tanto, los inodos especifican el propietario de un fichero.
- El **identificador de grupo (GID o Group ID)**. De este modo, un inodo contiene el grupo a que pertenece un fichero.
- **Marcas de tiempo** como por ejemplo la fecha en que se ha creado el archivo, la fecha del último acceso, etc.
- **Tabla de direccionamiento** donde se detallan los bloques del disco duro en que está almacenado el fichero.

Mediante el comando **stat** podemos consultar la información que un inodo guarda sobre un fichero. Si ejecutamos el comando stat seguido del nombre de un archivo, directorio o enlace obtendremos el siguiente resultado:

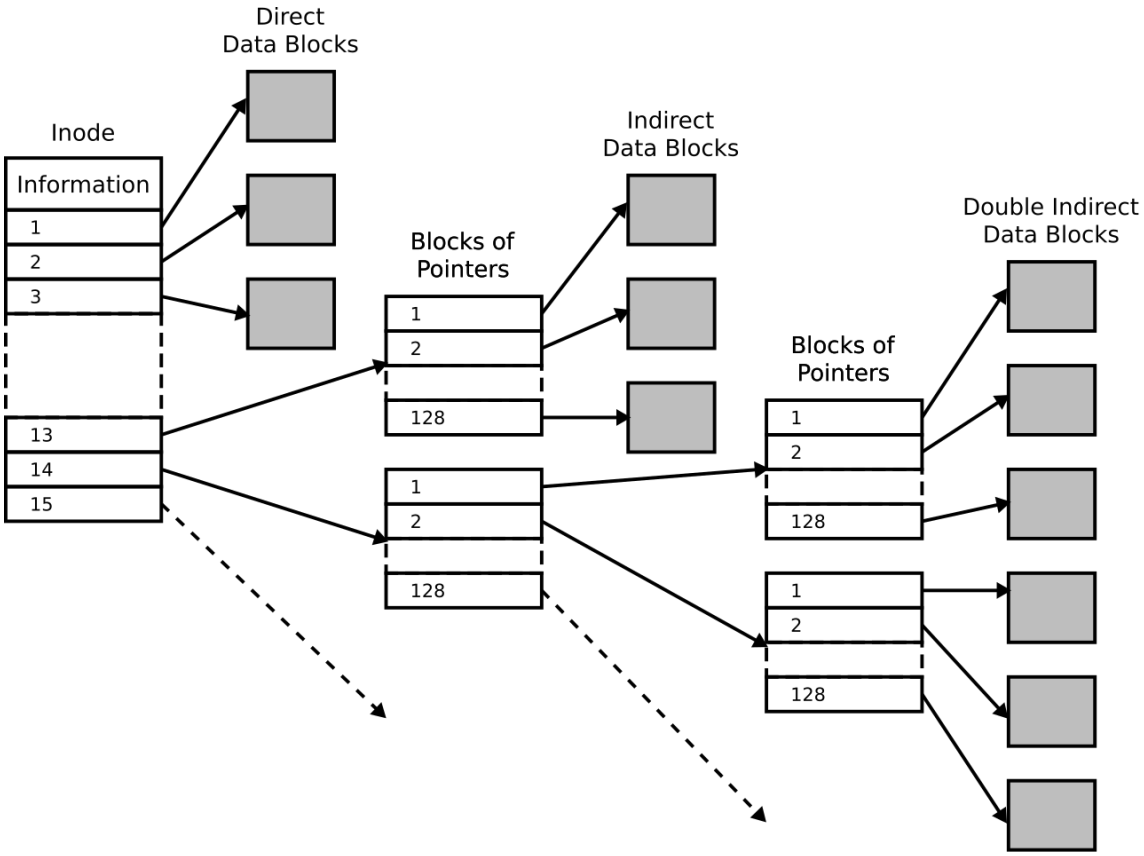
```
joan@debian:~$ stat archivo_1
Fichero: archivo_1
Tamaño: 7872      Bloques: 16     Bloque E/S: 4096
↪fichero regular
Dispositivo: 805h/2053d      Nodo-i: 1326361      Enlaces: 1
Acceso: (0644/-rw-r-r-)     Uid: ( 1000/ joan)     Gid: ( 1000/
↪joan)
Acceso: 2018-02-04 21:04:11.902405984 +0100
Modificación: 2016-12-25 11:24:27.051411224 +0100
Cambio: 2016-12-25 11:25:06.898576813 +0100
Creación: -
```

Como pueden ver, la salida del comando contiene la información que almacena el inodo 1326361 sobre el archivo archivo_1.

La principal función de la información almacenada en el inodo es que podamos acceder a la información almacenada en nuestro disco duro.

2.1.3 Los directorios y las dentries

Un directorio no es más que un fichero que contiene los nombres de los ficheros (o directorios) que están contenidos en él, junto con el número del inodo con la información de cada uno de ellos.



Las dentries tienen la función de definir la estructura de un directorio. Las dentries, conjuntamente con los inodos, serán los encargados de representar un fichero en la memoria.

Las dentries de un directorio se almacenan en una tabla. Esta tabla contiene la totalidad de nombres de los ficheros que están dentro del directorio y los asocia con su correspondiente número de inodo. Por lo tanto, una dentry es un nombre que apunta hacia un inodo.

Imaginemos que el directorio Documentos contiene 2 archivos y un directorio. El inodo que representa al directorio Documentos será el siguiente:

Tabla 1: Ejemplo de entradas de directorio

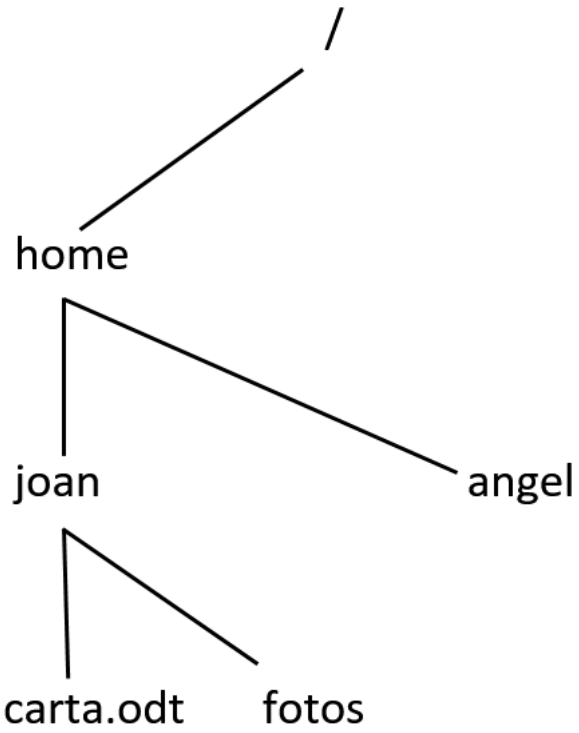
Ficheros dentro del directorio Documentos	Inodo
.	10000
	5000
Carta.odt	10043
CV.odt	10025
Directorio 1	13412

El contenido de la tabla es simple de comprender. Únicamente es necesario comentar las entradas que tienen un punto y dos puntos.

La entrada con un punto (.) hace referencia al directorio Documentos. Por lo tanto la información del directorio Documentos está almacenada en el inodo 10000.

La entrada con 2 puntos (..) hace referencia al inodo del directorio que contiene el directorio Documentos. Por lo tanto si la carpeta Documentos está dentro de /home/user, el inodo 5000 hace referencia al directorio user.

Para comprender mejor todo lo que hemos comentado hasta el momento imaginemos la siguiente estructura:



Las características de esta estructura son las siguientes:

Usa 6 inodos. El primer inodo para el directorio raíz, el segundo para el directorio home, el tercero para el directorio joan, el cuarto para el directorio angel, el quinto para el fichero carta.odt y el sexto para el directorio fotos. Usa 5 dentries. La primera enlaza carta.odt con joan, la segunda enlaza fotos con joan, la tercera enlaza joan con home, la cuarta enlaza angel con home y finalmente la última enlaza home con el directorio raíz.

2.1.4 Direccionamiento de un Inodo

La función de la tabla de direccionamiento de un inodo es indicar las posición del disco duro en que está almacenado un fichero.

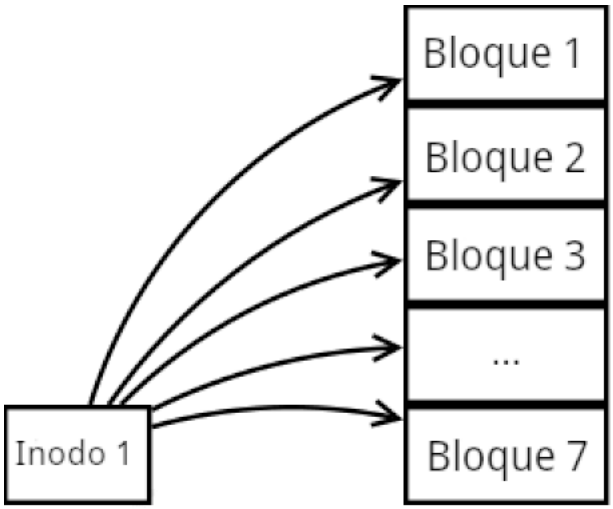
Los inodos tienen un tamaño fijo. Por lo tanto, el tamaño de un inodo es finito y en muchas ocasiones no es suficiente para almacenar la totalidad de datos que necesitamos para acceder a un archivo. Frente a esta situación se usan tablas de direccionamiento indirecto simples, dobles y triples.

Tablas de direccionamiento directo

Un inodo tiene una tabla de direccionamiento de 15 entradas. 12 de las 15 entradas permiten un direccionamiento directo a un bloque de datos del disco duro.

Si un archivo se almacena en 7 bloques, un inodo nos puede direccionar de forma directa al contenido del archivo.

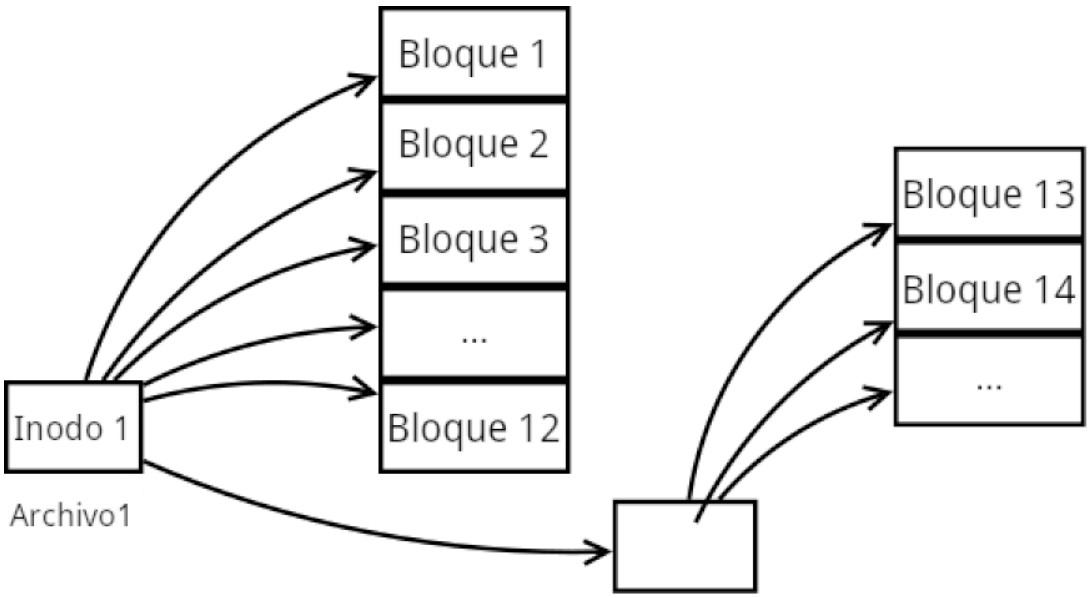
A modo de ejemplo, el archivo archivo1 está vinculado al inodo 1. Si consultamos el inodo 1 vemos que el contenido del archivo1 está almacenado en los bloques 1, 2, 3, 4, 5, 6 y 7. Por lo tanto, mediante un direccionamiento directo podemos acceder al contenido del archivo.



Tablas de direccionamiento indirectas simples

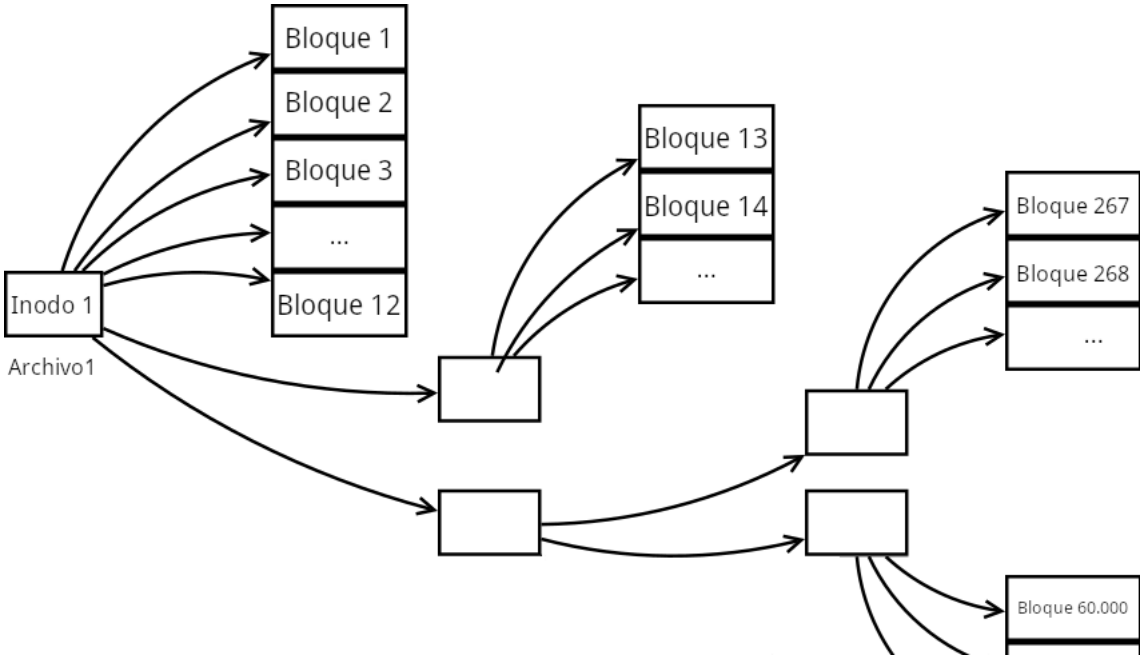
Si finalizamos el espacio que tenemos para las 12 entradas directas tendremos que usar un direccionamiento indirecto.

La treceava posición de la tabla de direccionamiento del inodo nos dirigirá a un bloque de datos que contendrá una nueva tabla de direcciones hacia los bloques que almacenan el contenido de nuestro archivo.



Tablas de direccionamiento indirectas dobles y triples

Del mismo modo que se hacen direccionamientos indirectos simples, también podemos hacer direccionamientos indirectos dobles y triples con las entradas 14 y 15. A continuación se muestra la estructura de un direccionamiento indirecto doble.



De este modo podemos llegar a conseguir archivos con un tamaño de hasta 16TB en sistemas de ficheros ext4.

Frente a este funcionamiento podemos llegar a las siguientes conclusiones:

- El acceso a los archivos será más lento cuando se hace un direccionamiento indirecto.
- Podremos acceder de forma mucho más rápida y eficiente en archivos de poco tamaño. Por este motivo los sistemas de archivos basados en inodos son ideales para manejar servidores web o servidores de email.

CAPÍTULO 3

Proceso de análisis en Linux

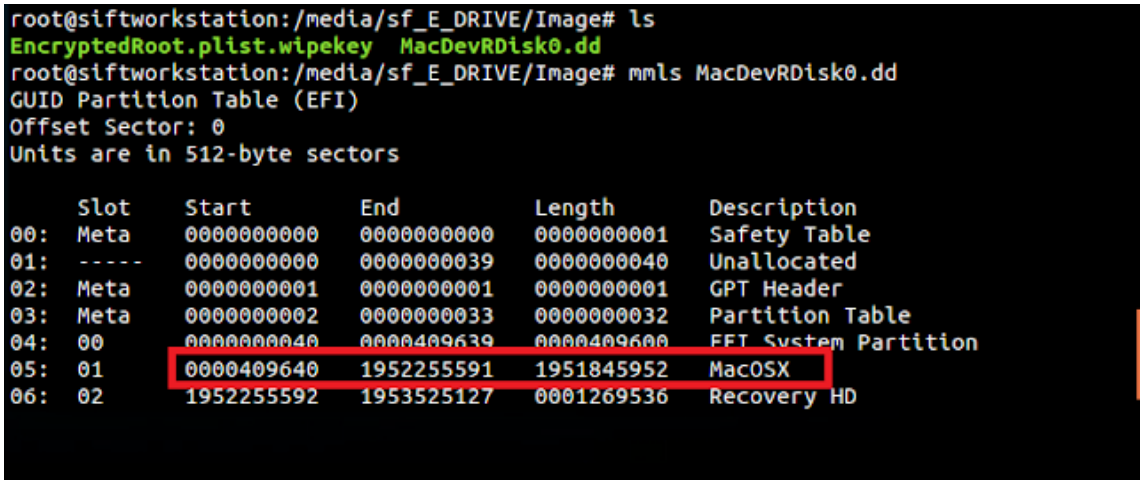
Cuando se inicia un análisis, un posible primer paso es montar la imagen realizada, bien desde consola con el comando mount o bien gráficamente. Si se decide hacerlo así, se deben elevar los privilegios como primer paso. Si se ha clonado a disco, se busca con «fdisk -l» y se monta la partición con:

```
mount /dev/sdxn media/caso
mount [origen] [destino]
```

Si se ha clonado a imagen, para ver el contenido y particiones de una imagen adquirida:

```
fdisk -l /media/adquisicion/imagen.dd
mmls /media/adquisicion/imagen.dd
```

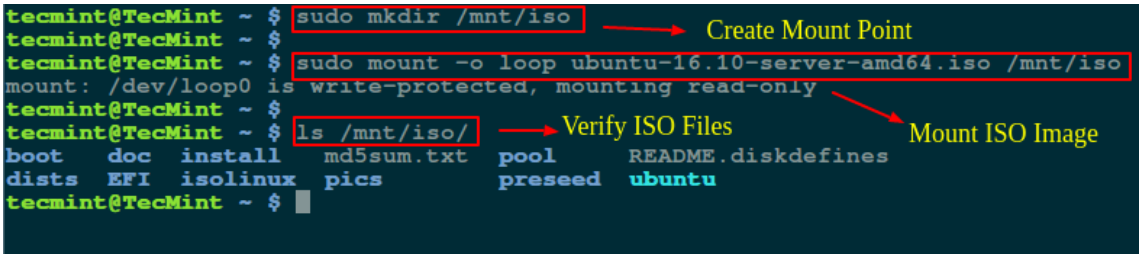
Para montar la imagen, en ocasiones habrá que especificar el sistema de archivos a montar (si es ext2, ext3, ext4...), cual es la partición que se debe montar y el bloque donde comienza la partición a montar. Para ello se requiere calcular el «offset» o desplazamiento en bytes. El comando «mmls» incluido en The Sleuth Kit ayuda a cumplir este objetivo. «mmls» muestra la disposición de las particiones en un volumen, lo cual incluye tablas de partición y etiquetas de disco.



Para calcular el «offset» o desplazamiento se requiere multiplicar el sector donde inicia la partición por el tamaño de sector en bytes. Con la información de la imagen anterior el cálculo seria (512 * 2048 = 1048576) para la primera partición, seria (512 * 393216 = 201326592) para la segunda partición y así sucesivamente.

Obtenido este dato se procede a utilizar el comando Mount para realizar el montado. Con la bandera «-o» se indican diferentes opciones necesarias para montar una imagen. Las principales que se usarán son «ro» para indicar que se monte en solo lectura, «loop» simula un dispositivo de bloques y «offset» la distancia en bytes desde el inicio del disco hasta el comienzo de la partición que se desea montar. Seguido de la imagen del sistema objetivo y la ruta donde se desea montar dicha partición.

```
mount -o ro,loop,offset=[bytes] [imagen] [destino]
```



Como se puede ver en la imagen anterior es posible usar sustitución de comandos para realizar el cálculo del offset «al vuelo», lo cual ahorra ciertos pasos y facilita la creación de scripts Una vez montada la imagen, se puede interactuar con el directorio mediante comandos. Un elemento fundamental es el estudio del timestamp de los ficheros encontrados.

3.1 TimeStamp en Linux

El timestamp es crítico en cualquier análisis y dependiendo del sistema de ficheros con el que se trabaja, este varía. Se van a mostrar diferentes acciones habituales sobre el sistema de ficheros como mover, copiar, descargar y su timestamp correspondiente.

Lo primero es conocer la zona horaria del sistema a analizar. En la mayoría de las distribuciones esto se puede encontrar en el fichero de configuración /etc/localtime.

```
$ ls -l /etc/localtime
/etc/localtime -> /usr/share/zoneinfo/Etc/UTC
```

Como se puede ver en la imagen anterior dicho sistema está configurado como UTC.

Toda entrada en la lista de inodo contiene los siguientes timestamps cuyas abreviaturas habituales son:

- **mtime** — fecha/hora de modificación (modify)
- **atime** - fecha/hora de acceso (access)
- **ctime** — fecha/hora de cambio (modificación de los metadatos)(change)
- **crtime** - fecha/hora de creación

3.1.1 Fecha/hora de Cambio - ctime

Es el timestamp del cambio de los metadatos. Como indica el nombre, refleja la modificación de metadatos de un archivo (por ejemplo, chown, rename).

3.1.2 Fecha/hora de Creación — crtime

El timestamp de creación es una de las marcas de tiempo más críticas para un analista. Algunos puntos a tener en cuenta al buscar este dato en Linux:

1. El sistema de archivos Ext3 solo admite tres marcas de tiempo: la de modificación, la de acceso y la de modificación de los metadatos.
2. En Ext4 se añadió la cuarta marca de tiempo que es la de dirección, pero la utilidad stat sigue mostrando solo tres marcas de tiempo.

```
$ stat pen.log
File: pen.log
Size: 789       Blocks: 8       10 Block: 4096  regular file
nevice: 801h/2049d      Inode: 3671760  Links: 1
Access: (0644/-rw-r--r--) Uid: (        0/      root)   Gid: ( ␣
 ↪0/      root)
Access: 2017-09-25 17:55:49.011074227 +0000
Modify: 2017-09-25 17:55:44.675238060 +0000
Change: 2017-09-25 17:55:44.675238060 +0000
Birth: -
```

La técnica de tiemstamp más común es usal el comando touch como:

```
touch fichero_a_modificar -r fichero_existente
```

Esto creará un archivo que será el fichero a modificar (1.txt) con el timestamp de modificación y el timestamp de acceso igual que el fichero existente (pen.log), no cambia el timestamp de modificación de metadatos. Los tiempos de modificación y acceso también se pueden cambiar individualmente; por ejemplo: touch -m -d “12017-08-10 10:12:12” fichero_a_modificar; touch -a -d “2017-08-10 10:12:12” fichero a modificar.

```
$ touch -m -d 'next Monday' 1.txt
$ stat 1.txt File: 1.txt.
Size: 0 Blocks: 0       I0 Block: 4096  regular empty file
Device: 801h/2049d      Inode: 3671778  Links: 1
Access: (0664/-rw-rw-r--) uid: ( 1000/  pilar)  Gid: ( 1000/    ␣
 ↪pilar)
Access: 2017-09-25 17:55:49.011074227 +0000
Modify: 2017-10-02 00:00:00.000000000 +0000
Change: 2017-09-26 21:50:24.777o23430 .0000
Birth: -
```

```
$ touch -m -d '2017-11-01 11:12:49' 1.txt
$ stat 1.txt
File: 1.txt.
```

(continué en la próxima página)

(proviene de la página anterior)

```
Size: 0 Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 3671778  Ltnks: 1
Access: (03664/-rw-rw-r--) Uid: ( 1000/ pilar)  Gid: ( 1000/ ␣
↪pilar)
Access: 2017-09-25 17:55:49.011074227 +0000
Modify: 2017-11-01 11:12:49.000000000 +0000
Change: 2017-09-26 21:53:50.248455334 +0000
Birth: -
```

El comando touch se usa para modificar la fecha de un fichero a la fecha actual o a una fecha que especifiquemos. Algunas de las opciones de touch son:

- -a: Cambia la fecha de acceso de archivo.
- -c: No crea archivo.
- -m: Cambia la fecha de modificación de archivo.
- -r: archivo_referencia: Utiliza la fecha correspondiente a archivo_referencia como el nuevo valor para la(s) fecha(s) modificada(s).
- -t time: Utiliza la fecha especificada como el nuevo valor para la(s) fecha(s) modificada(s). Dicho argumento debe ser un número decimal de la forma [[SS]AA]MMDDhhmm[.ss] con su significado obvio. Si SS no es especificado, el año SSAA es tomado como perteneciente al intervalo 1969-2068. Si ss no se especifica, se toma como valor 0. Es posible especificarlo dentro de los valores 0-61 así que es posible dar valores de cambio («salto») de minuto. La fecha resultante se toma como una fecha de la zona horaria especificada por la variable de entorno TZ. Se produce un error si la fecha dada es anterior al 1 de enero de 1970.

Por ello es conveniente revisar el histórico de comandos y ejecuciones (—/.bash_history) u otros mecanismos de registro en búsqueda de posibles empleos de esta técnica u otras similares. Se puede ver el crtime (create time) en el sistema de archivos Ext4 usando debugfs, por ejemplo:

```
sudo debugfs -R "stat [ruta]" [dispositivo origen] | grep crtime
```

3.1.3 Copia de archivo vs Mover archivo

1. Copiar un archivo en un directorio

Hereda la propiedad del directorio al que se copia.

a) El «mtime», «ctime» y «atime» del archivo cambian a la hora en que se copia el archivo.

b) El «mtime» y «ctime» del directorio cambian a la hora en que se copia el archivo.

2. Mover un archivo a un directorio

a) No hereda la propiedad del directorio al que se copia.

b) El «ctime» del archivo cambia a la hora en que se mueve el archivo.

c) El «mtime» y «ctime» del directorio cambian a la hora en que se mueve el archivo.

3.2 Históricos

En Linux se mantiene una serie de ficheros históricos de lo que hacen cada uno de los usuarios que son muy útiles en los análisis forenses. Estos archivos se suelen borrar cuando existe una intrust en el sistema para que no quede reflejada la metodología de acceso utilizada y eliminar así las evidencias. En Linux se pueden mostrar los archivos ocultos con «ls —a» y eso nos mostrara ficheros como el historial de lo realizado por el usuario en un intérprete bash, por defecto en muchas distribuciones, (.bash_history), el historial de operaciones de la herramienta less (.lesshst), o el histórico de ssh (.ssh/).

3.3 Análisis de directorios

Dependiendo de la naturaleza de la pericial puede interesarnos revisar uno u otro directorio, en ocasiones incluso puede ser necesario revisarlos todos. Se muestra a continuación lo que se puede encontrar en funci6n del directorio:

Directorio	Contenido
/bin/	Ejecutables triviales como cat, ls . . .
/sbin/	Ejecutables con necesidad de privilegios para su invocación
/dev/	Dispositivos conectados al equipo
/media/	Unidades extraíbles montadas en el sistema (típicamente)
/mnt/	Discos montados en el sistema (típicamente)
/lib/	A priori no es interesante, pero no por ello se debe pasar por alto su contenido
/boot/	Es el arranque, contiene el grub/lilo, configuraciones y el Kernel
/srv/	Servicios (en una imagen debería estar vacío)
/proc/	Sistema de archivos virtuales (en una imagen debería estar vacío)

Importante para el análisis forense es estudiar /var que contienen entre otros:

Directorio	Contenido
log/	Contienen los logs del sistema (importante estudiar este directorio)
mail	Información de los correos electronicos configurados
spool/	Colas de impresión del correo, programas, impresión
tmp/	Temporales de las aplicaciones (perduran más en el tiempo)
crash	Información sobre caídas del sistema
backups/	Copias de seguridad
lock/	Bloqueos
lib/	Librerías
run/	Procesos que en un momento dado habría en el sistema, para que el siguiente inicio el sistema tarde menos (información sobre usuarios, dispositivos montados, configuración dns, redes, Aquellos con extensión. pid son los que han estado conectados

También es importante estudiar /etc que principalmente configuraciones:

Fichero	Contenido
passwd	Información sobre usuarios
shadow	Claves cifradas de los usuarios
group	Grupos de usuarios
network/	Información sobre la configuración de red
init.d/	Todos los daemons/servicios que se arrancan (revisar por si hay alguno anómalo)
fstab	Información sobre el montaje de los sistemas de ficheros
hosmame	Nombre del Host
hosts	Correspondencia entre direcciones ip y dominios

3.4 Comandos muy útiles en forense

Podemos utilizar una serie de comandos para complementar la búsqueda de información que necesitemos para el análisis forense:

Utilidad	Comando
Estructura de un archivo	**file**
Cadenas en un archivo	**strings**
Entradas y salidas del sistema	**last**
Intentos de acceso fallidos para un usuario concreto	**lastb**
Ultimo intento de conexión de red efectuado por un usuario, lo que resulta muy útil para verificar intrusiones externas	**lastlog**
Ficheros Ocultos	ls —a

A través de editores hexadecimales, como **xxd**, podemos encontrar cierta información relevante en el interior de los archivos.

3.5 Análisis de logs

El sistema de logs es un mecanismo estándar que se encarga de recoger los mensajes generados por programas, aplicaciones y demonios. Cada mensaje consta de la fuente (el programa o aplicación que lo generó) , la prioridad, fecha y la hora. Algunos ejemplos son:

Información	Fichero
Log que guarda un registro de los mensajes del kernel	**/var/log/kern.log**
Registro de mensajes del sistema y de sus programas	**/var/log/syslog**
InformaciÓn de buffer del kernel del sistema	**/var/log/dmesg**
Información de depuración de los programas	**/var/log/debug**
Información sobre el entorno grafico	**/var/log/Xorg.O.log**
Información del arranque	**/var/log/boot.log**
Configuración de las fuentes del sistema	**/var/log/fontconfig.log**
Logs del servidor de correo	**/var/log/mail.log**
Conexiones al sistema incluidos los intentos fallidos y los accesos como root	**/var/log/auth.log**
Tareas programadas (cron)	**/var/log/crond**
Alertas específicas de algunos demonios	**/var/log/daemon.log**
Nos muestra errores	**/var/log/errors.log**
Apache por defecto registra los eventos	**/var/log/httpd**
Alertas generales del sistema	**/var/log/messages.log**
Registro de eventos de MySQL	**/var/log/mysqld.log**
Registro de seguridad	**/var/log/secure**
El log del servidor FTP (vsfp).	**/var/log/vsftpd.log**

Podemos ayudarnos de los comandos grep y tail para filtrar mejor la información, por ejemplo:

Utilidad	Comando
Con grep se filtran patrones	**$ grep cadena_de_caracteres archivo_donde_buscar**
Con tail se muestra la parte final de uno o varios ficheros	**$ tail archivo_donde_buscar**

3.6 Recuperación de ficheros borrados

Se muestran a continuación una muestra de las múltiples herramientas que existen para la recuperación de ficheros borrados:

1. **TestDisk** Recupera datos perdidos en particiones y repara discos de arranque. También se puede recuperar las tablas de particiones de un disco duro que, por una razón u otra, ha quedado sin formato.
2. **Foremost** Es capaz de detectar tipos de ficheros por su cabecera, no por su extensión (que puede estar cambiada para ocultar información), y permite encontrar ficheros borrados.
3. **Photorec** Recuperar archivos perdidos de diversos formatos y desde diferentes medios de almacenamiento.

CAPÍTULO 4

Forense de RAM

Destacar LIME por su compatibilidad con múltiples arquitecturas Linux, incluidas plataformas móviles Android. Además, permite la adquisición en distintos formatos (lime, raw, . . .) y el envío del volcado por red. Tambien existe la herramienta fmem. Para hacer el volcado de memoria se puede intentar almacenar la memoria /dev/mem en un fichero:

```
dd if=/dev/mem of=./memdump.bin
```

Pero esto solo copia 1 Mb, por ello es necesario la utilización de herramientas como las que se verán a continuación.

4.1 Memdump

El comando **memdump** se puede utilizar para volcar la memoria del sistema al stream de salida, saltando los huecos de los mapas de la memoria. Por defecto vuelca el contenido de la memoria física (/dev/mem). Se distribuye bajo la Licencia Pública de IBM.

4.2 Fmem

Es una herramienta para el volcado de memoria en sistemas Linux que no se basa en patrones, por lo que funciona con cualquier tipo de sistema /kernel Linux origen. Integrado en dicha herramienta se incluye un 1km (o Loadable Kernel Module) que crea un nuevo dispositivo (/dev/fmem) el cual permite acceder directamente a la memoria y, por lo tanto, obtener un volcado de la misma utilizando dd.

Es necesario recompilarlo con la versión exacta del kernel de la máquina de la que se desea adquirir el volcado. Se descarga la herramienta fmem (https://github.com/NateBrune/fmem) en nuestra estación forense, se compila con el kernel adecuado al equipo objetivo

y se vuelca en un dispositivo de almacenamiento que se utilizara en el equipo objetivo del volcado.

En el equipo objetivo con el fmem ya cargado es necesario conocer la cantidad de ram instalada en dicho equipo, para ello se utilizará el comando «free -m» que mostrará el tamaño en megabytes. Se procederá a volcar la memoria desde el dispositivo /dev/finem, es importante indicar el tamaño de la ram que se ha obtenido con el paso anterior:

```
dd if=/dev/fmem of=[destino] bs=1MB count=[RAM]
```

4.3 LIME (Linux Memory Extractor)

Es una utilidad desarrollada con el objetivo de ser ampliamente compatible con múltiples arquitecturas Linux, incluidas plataformas móviles Android. Además, permite la adquisición en distintos formatos (lime, raw y padded) y el envío del volcado por red.

Es necesario recompilarlo con la versión exacta del kernel de la máquina de la que se desea adquirir el volcado. Se descarga la herramienta lime (https://github.com/504ensicsLabs/LiME) en nuestra estación forense, se compila con el kernel adecuado al equipo objetivo y se vuelca en un dispositivo de almacenamiento que se utilizara en el equipo objetivo del volcado. El dump se ejecutará cuando el módulo sea cargado usando los parámetros que recibe como argumentos.

```
insmod [lime KO] "path=[destino] format=lime"
```

Otra función que proporciona es realizar el volcado por red, para ello:

```
insmod [lime KO] "path=tcp:[puerto] format=lime"
```

Mientras en la estación forense:

```
ncat [ip_origen] [pto_origen] > [destino]
```

Y para descargar el módulo:

```
rmmod [lime KO]
```

Y finalmente se debe comprobar que el tamaño del volcado corresponde con el tamaño de la memoria RAM.

4.4 AVML

AVML (https://github.com/microsoft/avml) es una herramienta de adquisición de memoria volátil para el usuario X86_64 escrita en Rust, destinada a implementarse como un binario estático. AVML se puede utilizar para adquirir memoria sin conocer a priori la distribución del sistema operativo o el kernel de destino. No se necesita compilación o toma de huellas dactilares en el objetivo.

Características:

- Guarda las imágenes grabadas en ubicaciones externas a través de Azure Blob Store o HTTP PUT
- Reintento automático (en caso de problemas de conexión de red) con retirada exponencial para cargar en Azure Blob Store
- Compresión de nivel de página opcional usando Snappy.
- Utiliza formato de salida LiME (cuando no se utiliza compresión).

Se puede descargar los binarios de la herramienta (https://github.com/microsoft/avml/releases) o compilarla a partir del codigo fuente como se muestra más abajo.

```
# Install MUSL
sudo apt install musl-dev musl-tools musl

# Install Rust via rustup
curl https://sh.rustup.rs -sSf | sh -s -- -y

# Add the MUSL target for Rust
rustup target add x86_64-unknown-linux-musl

# Build
cargo build --release --target x86_64-unknown-linux-musl

# Build without upload functionality
cargo build --release --target x86_64-unknown-linux-musl --no-
↪default-features
```

4.5 Linux Memmory Grabber

Se trata de un script para volcar la memoria de Linux y crear perfiles de Volatility (TM).

Para analizar la memoria de Linux, primero se debe poder capturar la memoria de Linux. AVML funciona muy bien, pero si el SO no tiene /proc /kcore o /dev/crash, necesitará el extractor de memoria Linux (LiME) de Joe Sylve. Pero se necesita tener un módulo LiME compilado para el kernel del sistema donde desea obtener RAM.

Volatility (TM) es excelente para analizar imágenes de memoria de Linux. Pero necesita un perfil que coincida con el sistema donde se capturó la memoria. Construir un perfil significa compilar un programa en C en el sistema apropiado y usar dwarfdump (depurador de archivos ejecutables) para obtener las direcciones de las estructuras de datos importantes del kernel. También necesita una copia del archivo System.map del directorio /boot.

4.6 Análisis plataforma Linux con Volatility

Antes de iniciar el análisis es indispensable entender cómo se debe crear el perfil específico para el sistema operativo que se va a analizar ya que es normal encontrar cientos de distribuciones y variedades de versiones de kernel. Existen repositorios con varios perfiles para diferentes versiones de Linux. Por ejemplo https://github.com/volatilityfoundation/profiles y https://github.com/KDPryor/LinuxVolProfi les. Los Plugins de Linux se pueden encontrar en https://github.com/volatililyfoundation/volatility/wiki/Linux#using-the-plugins.

1) Muestra la relación padre/hijo entre procesos

```
pythonvol.py linux_pstree -f volcado.lime --profile=perfilX64
```

2) Indica las interfaces activas

```
pythonvol.py linux_ifconfig -f volcado.lime --profile=perfilX64
```

3) Muestra la tabla ARP

```
pythonvol.py linux_arp -f volcado.lime --profile=perfilX64
```

4) Recupera el bash history de la bash process memory

```
pythonvol.py linux_bash -f volcado.lime --profile=perfilX64
```

5) Lista sockets abiertos

```
pythonvol.py linux_bash -f volcado.lime --profile=perfilX64
```

6) Indica los dispositivos montados fs/devices

```
pythonvol.py linux_mount -f volcado.lime --profile=perfilX64
```

7) Imprime un banner de Linux versión

```
pythonvol.py linux_banner -f volcado.lime --profile=perfi1X64
```

8) Volcado del mapa de memoria para las task de linux

```
pythonvol.py linux_memmap -f volcado.lime --profile=perfilX64
```

9) Imprime información sobre cada procesador activo

```
pythonvol.py linux_cpuinfo --f volcado.lime --profile=perfilX64
```

10) Modulos del kernel cargados

```
pythonvol.py linux_lsmod --f volcado.lime --profile=perfilX64
```

CAPÍTULO 5

Análisis de Malware

Un rootkit es un conjunto de herramientas software que permite acceso privilegiado a un sistema informático y está normalmente oculto. Así puede corromper el funcionamiento lógico del sistema o aplicaciones, camuflando otros procesos maliciosos, archivos, puertas traseras o cambios de registro sospechosos en el sistema. Los rootkits se pueden instalar incluso a través de productos comerciales de seguridad y en extensiones de aplicaciones de terceros aparentemente seguras. Su detección y eliminación puede no ser sencilla. Se muestran a continuación una serie de herramientas para la detección de este tipo de malware, pero una de las desventajas que tienen es que suelen tener algunos falsos positivos y se deben ejecutar en sistemas en "vivo".

Un rootkit es en esencia, un programa o conjunto de programas que tiene como finalidad esconderse a sí mismo y además, esconder otros programas, procesos maliciosos, archivos, puertas traseras (puertos que permiten al intruso mantener el acceso a un sistema para remotamente comandar acciones o extraer información sensible) o cambios de registro sospechosos en el sistema. Digamos que entran dentro de la categoría de malware.

En GNU/Linux el peligro no son virus como en el mundo Windows sino las vulnerabilidades que se descubren de los programas que usa el sistema. De ahí la importancia de verificar y actualizar el sistema en su totalidad periódicamente.

5.1 Detección de rootkits

Hay varias aplicaciones y utilidades disponibles para la detección de rootkits. En entornos UNIX y GNU/Linux tenemos a chkrootkit y Rootkit Hunter como las utilidades más populares. Ambos se centran en analizar tu sistema en busca de rootkits y cambios sospechosos en configuraciones críticas del sistema.

A la hora de ejecutar estos programas, ninguno de los dos se ejecuta en segundo plano, hacen el análisis a demanda del usuario, por lo que ambos pueden convivir en el mismo sistema. Es más, contar con ambos será de gran utilidad para tener una segunda opinión y descartar posibles falsos positivos.

Además de las dos utilidades anteriores, podríamos reforzar la seguridad de nuestro sistema haciendo uso de la aplicación **Tripwire**. Esta aplicación monitoriza y alerta sobre los cambios realizados a determinados archivos de sistema.

5.1.1 CHKROOTKIT

Se trata de un script para sistemas UNIX y Linux que realiza un análisis rápido en busca de signos y cambios en ciertas configuraciones del sistema que indiquen la presencia de rootkits.

```
$ apt install chkrootkit
```

En sistemas debian se puede instalar desde los repositorios predefinidos.

Para ejecutar chkrootkit se requieren permisos de superusuario (root).

Para ver la sintaxis completa del comando chkrootkit: **$ man chkrootkit**.

Como puede verse en la página man de chkrootkit, este no dispone de muchas opciones de escaneo, lo más útil es hacer un escaneo estándar. El análisis irá mostrando por pantalla los diferentes archivos analizados. Para realizar un escaneo estándar ejecutar el siguiente comando:

```
$ sudo chkrootkit
```

La salida de chkrootkit da mucha información. Para que sea más fácil encontrar las detecciones y los avisos o las alertas, podemos utilizar el parámetro -q mediante el cual, chkrootkit mostrará por pantalla solamente las detecciones y los avisos o las alertas.

```
$ sudo chkrootkit -q
```

También podemos enviar la salida de chkrootkit a un fichero de log e incluso hacer uso de utilidades como grep para filtrar la información de salida. Un ejemplo de envío de la salida de chkrootkit a un fichero log podría ser el siguiente:

```
$ sudo chkrootkit | grep INFECTED > /ruta/al/archivo/log/
↪Chkrootkit-$(date +%Y-%m-%d).txt
```

Con esta orden mandaremos al archivo de log solo las salidas en las que aparezcan la palabra INFECTED y añadiremos al nombre de archivo la fecha en que se generó.

5.1.2 RKHUNTER

Es un script para UNIX y Linux que escanea el sistema a nivel local para detectar cualquier signo de rootkit o backdoors. Principalmente funciona mediante base de firmas, pero no solo se limita a eso, sino que también examina que los permisos del sistema sean correctos, que no haya puertos abiertos, etc.

Lo primero que hacemos es instalarnos el paquete rkhunter. En el resto de las distribuciones supongo que sepan como hacerlo, en Debian:

```
$ sudo apt install rkhunter
```

En el archivo /etc/default/rkhunter se define que las actualizaciones de la base de datos sean semanales, que la verificación de rootkits sea diaria y que los resultados sean enviados por e-mail al administrador del sistema (root).

No obstante, si queremos asegurarnos, podemos actualizar la base de datos con el siguiente comando:

```
root@server:~# rkhunter --propupd
```

Para comprobar que nuestro sistema esté libre de rootkits ejecutamos:

```
$ sudo rkhunter --check
```

La aplicación comenzará a realizar una serie de comprobaciones y en su momento nos pedirá oprimir la tecla ENTER para continuar. Todos los resultados los podremos consultar en el fichero /var/log/rkhunter.log

5.1.3 UNHIDE

Esta herramienta destinada a identificar anomalías en sistemas Unix/Linux, lo hace identificando procesos y puertos tcp/udp ocultos, conexiones inversas, que suele ser síntoma inequívoco de la presencia de un rootkit en nuestro sistema. No emplea firmas y se basa en los conceptos del tipo «deteccion de anomalias» contrastando diferentes fuentes de información y buscando inconsistencias entre ellas.

Unhide es una excelente herramienta forense de GNU/Linux que permite encontrar tanto procesos como puertos en escucha ocultos en el sistema. Este tipo de procesos y listeners suelen aparecer cuando el sistema ha sufrido algún tipo de ataque y ha sido infectada con un rootkit. También está disponible para Windows, ambas versiones se pueden descargar desde el sitio web www.unhide-forensics.info

Para descargarla por repositorios en Debian y Ubuntu:

```
# apt-get install unhide
```

Básicamente, unhide (modo ps)utiliza tres técnicas para encontrar los procesos:

- Comparar la salida del comando /bin/ps con los datos que hay en el filesystem /proc. Para utilizar esta técnica ejecutamos el comando con el parámetro «proc»:

```
# $ unhide proc

Unhide 20100201

http://www.security-projects.com/?Unhide

[*]Searching for Hidden processes through /proc scanning
```

- Comparar la información del comando /bin/ps con la información recolectada de syscalls (system call scanning). Para ello pasamos el parámetro «sys»:

```
$ unhide sys

Unhide 20100201
http://www.security-projects.com/?Unhide

[*]Searching for Hidden processes through kill(..,0) scanning

[*]Searching for Hidden processes through  comparison of␣
↪results of system calls

[*]Searching for Hidden processes through getpriority() scanning

[*]Searching for Hidden processes through getpgid() scanning

[*]Searching for Hidden processes through getsid() scanning

[*]Searching for Hidden processes through sched_getaffinity()␣
↪scanning

[*]Searching for Hidden processes through sched_getparam()␣
↪scanning

[*]Searching for Hidden processes through sched_getscheduler()␣
↪scanning

[*]Searching for Hidden processes through sched_rr_get_
↪interval() scanning

[*]Searching for Hidden processes through sysinfo() scanning

HIDDEN Processes Found: 0
```

- Y por última la técnica desesperada con el parámetro «brute», disponible sólo es kernels superiores a la versión 2.6. Esta técnica consiste en buscar por fuerza bruta en todos los PIDs del sistema.

```
# unhide brute
```

El modo TCP de unhide permite identificar los puertos que están escuchando (TCP y UDP) en nuestro sistema pero que por contra no aparecen al ejecutar el comando netstat:

```
# unhide-tcp
```