# SEGURIDAD EN SISTEMAS OPERATIVOS

4º Grado en Informática – Complementos de Ing. del Software

Curso 2017-18

Práctica 1. Administración de la seguridad en Linux

Sesión 1. Seguridad básica en Linux: privilegios de usuario y permisos

## 1. Introducción

La seguridad en un sistema se basa en los mecanismos de protección que ese sistema proporciona. Estos mecanismos deben permitir controlar qué usuarios tienen acceso a los recursos del sistema y qué tipo de operaciones pueden realizar sobre esos recursos.

Todo mecanismo de protección debe manejar 2 conceptos:

1. Recursos: son las partes del sistema utilizadas por los procesos.
2. Dominios de protección: son el conjunto de recursos y operaciones sobre estos recursos que podrán utilizar todos aquellos procesos que se ejecuten sobre él.

En general, en Linux, el conjunto de recursos está formado por todos los archivos del sistema, y el dominio será un usuario y los procesos que el ejecuta, definidos por un mismo UID efectivo. En los apartados siguientes, estudiaremos los aspectos de seguridad relacionados con los privilegios de usuario y sus permisos. Abordaremos también la configuración de la autenticación y el registro de eventos del sistema.

## 2. Archivos de claves

La información de usuario se almacena en los archivos de claves. En los sistemas actuales, en el archivo */etc/passwd* se guardan las credenciales UID y GID que son las que determinan los objetos a los que tiene acceso un usuario.

Los archivos */etc/shadow* y */etc/gshadow* gestionan las claves de usuarios y grupos, respectivamente. Recordad que se desgajan del primero por que los permisos de lectura son diferentes. Estos archivos necesitan para su manejo privilegios de root.

La diferencia entre ambos radica en que las claves */etc/gshadow* se configuran con mucha menor asiduidad. Podemos cambiar las clave de grupo con:

```
$ gpasswd proyecto
```

Ejercicio 1: Indicar los formatos de los archivos */etc/passwd*, */etc/group*, */etc/shadow* y */etc/gshadow*.

Cuando creamos un usuario o grupo, los valores por defecto están disponibles en el archivo */etc/logins.defs*. Si estamos interesados en una seguridad más estricta, deberemos considerar cambiar algunos de estas directivas de configuración que se muestran en la Tabla 1.

Tabla 1.- Directivas de relacionadas con seguridad para nuevos usuarios y grupos de /etc/login.defs.

| Directiva de seguridad | Descripción |
|---|---|
| **LOG_OK_LOGINS** | Los logins con éxito se registran en el archivo */var/log/syslog.conf* |
| **SYSLOG_SU_ENAB** | Se registran los usos de la orden `su`. |
| **SYSLOG_SG_ENAB** | Se registran los usos de la orden `sg`. |
| **SULOG_FILE** | Si se define, registra toda la actividad de `su`. |
| **PASS_MAX_DAYS** | Máximo número de días que se puede usar una clave. |
| **PASS_MIN_DAYS** | Mínimo número de días que se puede reservar la clave. |
| **LOGIN_TIMEOUT** | Tiempo máximo de un login de consola. |

Recordad las órdenes para la gestión de usuario y grupos vistas en Sistemas Operativos: `useradd`, `usermod`, `userdel`, `groupadd`, `groupmod`, `groupdel`, `groups`, y `chage`.

---

Ejercicio 2.- Modificar el archivo */etc/login.defs* para que los usuarios creados a partir de ese momento tenga un valor asignado para las directiva **LOGIN_TIMEOUT**. Crear un usuario y comprobar que tiene efecto la citada directiva.

---

## 3. Listas de control de acceso

Para controlar el acceso de los dominios a los recursos se utilizan las Listas de Control de Acceso por cada recurso. La Lista de Control de Acceso (ACL) especifica qué dominios tienen acceso al recurso y qué operaciones asociadas al recurso pueden utilizar. El problema que plantea la lista de control de acceso es su tamaño variable, ya que depende del numero de dominios que tengan acceso al recurso y de las operaciones que pueda realizar cada uno de ellos.

En Linux, vimos que para conseguir listas de tamaño constante, se utilizan 2 técnicas:

1. Reducir el numero de operaciones posibles sobre un recurso (archivo): podemos controlar 3 operaciones sobre los archivos, que son la lectura (r), escritura (w) y la ejecución (x).
2. Reducir el número de dominios que aparecen en la lista. Esto se consigue mediante el concepto de grupos de usuarios.

Todos los usuarios de un sistema Linux deben pertenecer, al menos, a un grupo. Como vimos existen 3 grupos o categorías en la relación entre un dominio ( usuario ) y un recurso (archivo ): propietario, grupo del propietario, y resto de usuarios.

Con estos 2 mecanismos la Lista de control de acceso se reduce a 9 bits, organizados en 3 grupos de 3. Esta Lista de control de acceso está almacenada en el campo `i_mode` del *inodo* (que es donde se mantienen todos los atributos -metadatos- asociados a un archivo). Parte de los contenidos, atributos, del inodo son visibles con la orden `ls -l`:

```
-rwxr--r-- 1 root root 512 Nov 24 17:59 mi_archivo
```

Es conveniente que el administrador asigne al sistema, por defecto, una máscara de permisos *umask* restrictiva. Por ejemplo 077 e incluirla en */etc/profile*. La orden `umask` establece los permisos que no se asignaran a los archivos. En el caso de 077 (`---rwxrwx`) al crear un archivo tendrá los permisos `rw-------`. Ejecución no se asigna por defecto.

Pero realmente la palabra de protección contiene 16 bits, en lugar de solo 9, cuyo contenido se interpreta según se muestra en la Figura 1.

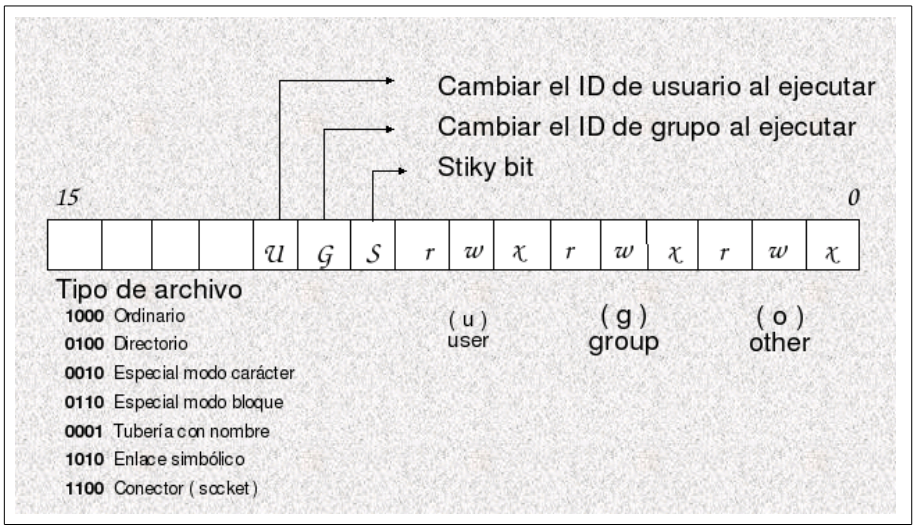


Figura 1.- Bits de la palabra de modo [1].

Como podemos ver en la Figura, dentro de la palabra de protección del archivo, en la que se reflejan los diferentes permisos para los diferentes tipos de usuarios, existen unos bits que ocupan las posiciones 10, 11, y 12 cuyo estado puede comprometer de alguna forma la seguridad del sistema. Estos bits son:

- *Sticky-bit* (bit t/T sobre other) que activado sobre un directorio indica que un archivo de ese directorio solo puede ser borrado por: el propietario del archivo, el propietario del directorio o el root. Si aparece `t`, el bit sticky está a uno y el bit de ejecución de otros (other) está también a 1. Si aparece `T` el bit sticky está a uno y el bit de ejecución de otros está a 0. La orden que activa el sticky-bit sobre el directorio */home/usuario1/* es la siguiente:

  ```
  # chmod +t /home/usuario1/
  # ls -l /home
  drwxr-xr-t 13 usuario1 usuario1 4096 Apr 16 2004 usuario1
  ```

  Esta opción está activada por defecto en los directorios del sistema */tmp* y */var/tmp/*, que son utilizados por muchos programas. Estos programas presuponen que nadie puede borrar sus archivos una vez que han sido creados. Si se elimina de ellos el sticky-bit pueden quedar

expuestos a *ataques symlink*[1]. Este tipo de ataque aprovecha la creación de archivos temporales, por parte de algunos programas, en estos directorios y si dichos archivos tienen los permisos adecuados es posible crear enlaces simbólicos entre ellos y archivos del sistema, como */etc/passwd*, y conseguir el permiso de escritura sobre él al ejecutar dicho script.

- *Bit SGID* (bit s/S sobre group) que activado sobre un archivo, da los derechos del grupo a quien lo ejecuta, y solo durante la ejecución. Si aparece `s` el bit de *setgid* está a uno y el bit de ejecución del grupo está a 1. Si aparece `S` el bit de setgid está a uno y el bit de ejecución del grupo está a 0.
- *Bit SUID* (bit s/S sobre owner) que activado en un archivo ordinario, asigna los derechos del propietario del archivo a quien lo ejecuta, y solo durante el tiempo de ejecución. El sentido de s/S es el mismo que en SGID. La orden que activa el bit SUID sobre un archivo es la siguiente:

  ```
  # chmod +s /home/archivo
  ```

  El clásico ejemplo de archivo con el bit setuid activo es la orden */usr/bin/passwd*. Es necesario que sea así, ya que el usuario que cambia su *password* debe poder escribir sobre el archivo */etc/shadow* cuya propiedad es del root y necesita temporalmente los privilegios del root.

  ```
  -rwsr-xr-x 1 root root 26616 May 10 2004 /usr/bin/passwd
  ```

  Si alguno de los programas con el bit SUID activado presenta algún agujero de seguridad, a través de ellos un potencial atacante puede adquirir privilegios de root. Por tanto, es importante comprobar qué archivos en el sistema son SUID y vigilar si aumentan en número. Las ordenes siguientes permiten conocer qué archivos del sistema tienen activado el bit SGID y el bit SUID:

  ```
  # find / -type f ( -perm -2000 -o -perm -4000 )
  ```

  donde 2000 y 4000 son números en octal.

Es importante que el administrador impida la ejecución de archivos con el bit SUID activado en aquellos directorios en los que los usuario tienen permiso de escritura, como es el caso de */home*. La primera medida sera aislar /home en una partición de disco independiente y a continuación, en el archivo */etc/fstab*, activar la opción `nosuid` para dicha partición.

En el Apartado 6, volvemos sobre el tema y repasaremos como establecer los identificadores desde nuestros programas haciendo uso de las llamadas al sistema creadas para a tal efecto.

## 4. Atributos extendidos de un archivo

Antes citábamos algunos de los atributos de un archivo: propiedad del archivo, permisos, tamaño, fechas de creación-modificación-acceso, tipo, etc, es decir, los que estamos acostumbrados a manejar. Pero los sistemas de archivos actuales soporta un conjunto de nuevos atributos que se denominan *atributos extendidos*. Para poder usarlos debemos de montar el sistema de archivos con la opción de montaje `user_xattr`.

Además de la orden conocida para cambiar los permisos de un archivo, `chmod`, podemos cambiarle el propietario o grupo con las ordenes **`chown`** y **`chgrp`**, respectivamente. Por ejemplo, podemos cambiar

1 http://minsky.gsi.dit.upm.es/semanticwiki/index.php/Symlink_Attacks

el usuario y grupo de todos los archivos de directorio de marcos:

```
$ chwon -R marcos ~marcos/archivo
$ chgrp -R jefes ~marcos/archivo
```

Los sistemas archivos de Linux más frecuentes, como son ext2, ext3 y ext4, definen algunos atributos suplementarios, como define la Tabla 2. Estos atributos pueden consultarse con **`lsattr`** y modificarse con **`chattr`**.

Tabla 2. Atributos extendidos de ext2, ext3 y ext4.

| Atributo | Significado |
|---|---|
| A | No modificar nunca el tiempo de actualización del archivo (`atime`) |
| a | Permitir la escritura solo en modo de apertura (limitado al root) |
| c | Comprimir el archivo |
| D | Forzar la actualización de directorios para su escritura síncrona |
| d | No copias de seguridad (hace que volcado lo ignore) |
| i | Hacer al archivo inmutable e imborrable (limitado al root) |
| j | Mantiene un diario de los cambios de los datos y metadatos |
| s | Fuerza que los cambios se escriban de forma síncrona (no búfering) |
| u | Archivo imborrable |

Como es fácil imaginar los atributos como `A`, `i`, `j`, y `u` son especialmente interesantes de cara a la seguridad. Por ejemplo, las siguientes órdenes muestran y cambian los atributos extendidos de archivo texto.txt:

```
$ lsattr texto.txt
---------------- texto.txt
$ chattr +a texto.txt
```

Pero disponemos de atributos más sofisticados que tienen un nombre y un valor asociado. El nombre comienza con un identificador del espacio de nombre, seguido por un punto (.) seguido por una cadena terminada en nulo. Podemos añadir tantos nombre separados por puntos como queramos para crear clases de atributos.

Actualmente Linux tiene cuatro espacios de nombre para atributos extendidos:

- *user* – no tiene restricciones para crear nombre o contenidos.
- *trusted* – Estos atriburos son solo visibles y accesibles a quienes tienen la capacidad CAP_SYS_ADMIN, como los administradores, y se usan para implementar mecanismos en espacio de usuario que mantienen información de atributos extendidos a los que un proceso ordinario no debería tener.
- *security* – utilizado por los módulos de seguridad del kernel como SELInux y que vememos con posterioridad.
- *system* – utilizado para principalmente por el kernel para almacenar los ACLs y capacidades y solo puede establecerse por el root para añadir metadatos.

Los atributos de usuario como su nombre indica son utilizados por el usuario o las aplicaciones. Este espacio de nombres esta protegido por los permisos del archivo: si tenemos permiso de escritura podemos asignar nuevos atributos. Algunos ejemplos de atributos de usuario:

- `user.checksum.md5`
- `user.checksum.sha1`
- `user.checksum.sha256`
- `user.original_author`
- `user.application`
- `user.project`
- `user.comment`

Los tres primeros se utilizan para almacenar sumas de comprobación sobre los archivos utilizando tres métodos diferentes. El cuarto indica el autor original en caso de que varias personas tengan permiso de escritura sobre el archivo o en caso de que el archivo se asigne a otro usuario. El quinto indica la aplicación que ha sido utilizada para generar los datos. El sexto liga los datos con el proyecto asociado. Y el último, para añadir comentarios de propósito general.

En la Tabla 3 se muestran las herramientas que podemos utilizar para manejar los atributos extendidos.

Tabla 3.- Herramientas para atributos extendidos.

| Sección del manual | Página Man |
|---|---|
| Utilidades (1) | getfattr<br>setfattr |

Para poder utilizarlas debemos asegurarnos de que el kernel este configurado y compilado para soportarlos (cosa que ocurre con los kernels recientes), también debemos asegurarnos de que el paquete *libattr* este instalado, y, por último, que el sistema de archivos en el que los vamos a utilizar este montado con la opción `user_attr`.

Como ejemplo de manejo de atributos de usuario, supongamos que tenemos un archivo denominado *test.txt* con algún texto:

```
$ echo "Archivo de texto para pruebas" > ./test.txt
$ more test.txt
Archivo de texto para pruebas
```

Podemos asignar un nuevo atributo al archivo con:

```
$ setfattr -n user.comment -v "Esto es un comentario" test.txt
```

Es decir, hemos añadido un atributo de comentario con el valor “Esto es un comentario”. Podemos ver la definición del atributo conclusión

```
$ getfattr test.txt
# file: test.txt
user.comment
```

Y podemos ver el valor del atributos conclusión

```
$ getfattr -n user.comment test.txt
# file: test.txt
user.comment="Esto es un comentario"
```

Podemos eliminar el atributo de la siguientes forma:

```
$ setfattr -x user.comment test.txt
$ getfattr -n user.comment test.txt
test.txt: user.comment: No such attribute
```

## 5.- Listas de Control de Acceso extendidas

El mecanismo de atributos extendidos, permite que los nuevos sistemas de archivos de Linux (ext2, ext3, ext4, btrfs, xfs, etc) permiten implementar ACLs completos, sin la limitación de compactarlos únicamente en el propietario, grupo y resto.

Si deseamos que algunos usuarios concretos tengan acceso a un archivo sin tener que crear un grupo especial para ellos, podemos crear un ACL. Por ejemplo, si creamos un archivo propiedad de jose:jose con permisos 0644, solo jose podrá editarlo. Si deseamos que Angela pueda escribirlo pero nadie más pueda hacerlo, podemos ejecutar:

```
$ setacl -m u:angela:rw archivoprueba
$ getacl archivoprueba
      # file: archivoprueba
      # owner: jose
      # group: jose
      # user:: rw-
      user:angela:rw-
      group::r-
      mask::rw-
      other::r-
```

Para manipular las listas de control de acceso, podemos usar las siguientes órdenes y funciones de la biblioteca que aparecen en la Tabla 4.

Tabla 4.- Órdenes y funciones para gestión de ACLs

| Sección | Pagina Man |
|---|---|
| Utilidades (1) | getacl<br>setacl |

Ejercicio 3.- Crear un ACL para un archivo de vuestro sistema de forma que el usuario creado en el Ejercicio 2 tenga acceso de lectura y escritura.

## 2.- PAM (*Puggable Authentication Modules*)

Los módulos PAM (Módulos de Autenticación Conectables) son un mecanismo flexible para autentificar usuarios. Gracias a él, los administradores pueden implementar diferentes políticas de autenticación para los distintos usuarios de forma individualizada para cada servicio. Al manejar esta característica debemos tener sumo cuidado ya que un error o despiste puede comprometer la seguridad del sistema[2].

2 Por ejemplo, si borramos los archivos de configuración /etc/pam.d/* o /etc/pam.conf bloquearemos el sistema.

PAM permite el desarrollo de programas independientes del mecanismo de autenticación a utilizar. De forma que es posible que un programa aproveche cualesquier de las facilidades de PAM sin cambiar su código. Además, permite al administrador construir diferentes políticas de autenticación para cada servicio. En resumen podemos sintetizar las ventajas de PAM para:

- Ofrecer un esquema de autenticación común y centralizado.
- Permitir a los desarrolladores abstraerse de las labores de autenticación.
- Facilitar el mantenimiento de las aplicaciones.
- Ofrecer flexibilidad y control  desarrolladores y administradores de sistemas.

- Arquitectura de PAM

La estructura de PAM viene reflejada en la Figura 2. Tiene cuatro elementos básicos:

- Consumidores PAM
- Biblioteca PAM
- Archivo de configuración *pam.conf*
- Módulos de servicio PAM o proveedores

Un módulo de servicio PAM es una biblioteca compartida que suministra autenticación u otros servicios seguridad a las aplicaciones que acceden al sistema, tales como `login`, `rlogin`, y `telnet`. Hay cuatro tipos de servicios:

- *Módulos de servicios de autenticación*: Conceden acceso a un usuario a una cuenta o servicio. Los módulos que suministran este servicio autentifican usuarios y fijan credenciales de usuario.
- *Módulos de gestión de cuentas*: Determinan si la cuenta del usuario actual es válida. Comprueban claves, expiración de las mismas, y accesos restringidos por tiempo.
- *Módulos de gestión de sesiones*: Establecer y terminar sesiones de trabajo.
- *Módulos de gestión de claves*: asegurar las reglas de fortaleza de claves y ejecutar las actualizaciones de tokens de autenticación.

Los dos primeras categrorías de módulos se referencia cada vez que un programa utiliza con éxito PAM para la autenticación. Los módulos de sesión se ejecutan si es necesario después de los dos anteriores. Los módulos de claves se acceden bajo demanda.

Un módulo PAM puede implementar uno o varios de estos servicios. El uso de módulos sencillos con tareas bien definidas incrementa la flexibilidad de configuración. Así, los servicios PAM deberían implementarse en módulos separados. Los servicios pueden usarse según las necesidades tal como se definen en el archivo *pam.conf*.

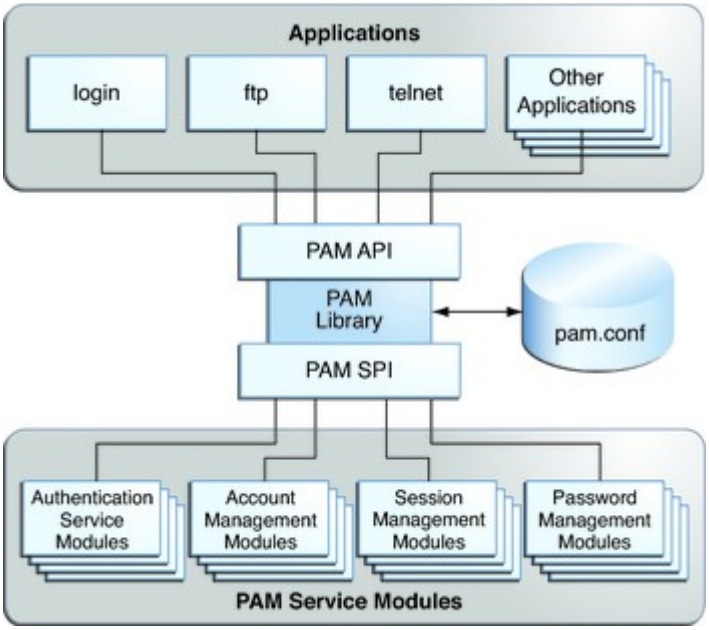


Figura 1.- Arquitectura de PAM [Oracle, *Developer's Guide to Oracle Solaris Security*, 2011]

La Biblioteca PAM, `libpam`, es el elemento central de la arquitectura que:

- Exporta una API, `pam`, que las aplicaciones pueden usar para autenticar, gestionar cuentas, establecer credenciales, gestionar sesiones, o cambiar claves.
- Importa del archivo de configuración maestra, *pam.conf*, los requisitos de los módulos PAM para cada servicio disponible. Este archivo esta gestionado por el administrador del sistema.
- Importa un SPI (Service Provider Interface), `pam_sn`, que es exportado por los módulos de servicio.

En nuestra práctica, solo vamos a estudiar los archivos de configuración PAM, quedando fuera del mismo las interfaces *pam* y *pam_sn* de la biblioteca PAM. Para que os hagáis una idea en el Apéndice 1, encontraréis el código de una aplicación para utilizar la funcionalidad PAM y la operatoria de funcionamiento. También, podéis encontrar más información en [6].

Volviendo a la Figura 1, si una aplicación quiere hacer uso de los mecanismos ofrecidos por PAM interactúa con la biblioteca PAM sin conocer ningún detalle de como esta configurado el sistema para tal aplicación. Será la biblioteca quien se encargue de leer la configuración para saber que política de autenticación debe ejecutarse. Para ellos los módulos de servicio o administración se apilan según el grupo de gestión y el orden en el que aparecen en la configuración. El orden de los módulos en la pila determina va a determinar el comportamiento para un servicio dado.

- Archivos de configuración PAM

Los archivos de configuración[3] se almacenan en el directorio */etc/pam.d*. Este directorio contiene en

3 Los archivos de configuración no distinguen entre mayúsculas o minúsculas, salvo en las rutas de acceso a los módulos así como los argumentos, que siguen las reglas de cada módulo. Si el módulo PAM se encuentra en la ruta determinada, */lib/security/*, no es necesario explicitar la misma, solo hay que hacerlo en caso de ser diferente.

general un archivo de configuración por cada aplicación que solicita autenticación PAM. Si una aplicación llama a PAM pero no hay un archivo de configuración asociado, se utiliza el archivo *other* (indicar que este archivo es la configuración por defecto para todas las aplicaciones consciente PAM, y, por tanto, si este es débil nuestro sistema puede ser vulnerable a ataques).

Dentro de los archivos de configuración, suele haber llamadas a archivos de inclusión que comienzan con *common-*. Estos son archivos de configuración generales cuyas reglas debería aplicarse en la mayoría de situaciones.

Los módulos que se referencian en los archivos de configuración poden localizarse con la orden `ls /lib/*/security`.

- Cómo PAM evalúa la autenticación

Cuando una aplicación consulta al sistema PAM para autenticación, este lee los archivos de configuración relevantes. Los archivos de configuración tienen una lista de módulos PAM que deben manejarse. Cada módulo es invocado por turno y cada llamada al módulo genera un resultado de éxito o fallo. Basado en esos valores, el archivo de configuración decide si debe retornar un mensaje "autenticación ok" o un fallo de autenticación.

La configuraciones pueden fallar al retorno del primer fallo de uno de los módulos invocados, o se pueden configurar políticas alternativas. Por ejemplo, un sistema puede permitir que usuarios LDPA se autentifiquen, y si esto falla, puede comprobar de nuevo la lista de usuarios locales.

Las líneas de cada archivo de configuración se evalúan desde el inicio al fin, salvo que una línea de evaluación provoque que se salte el resto del archivo de configuración.

- Cómo leer las líneas de política

Cada línea de un archivo de configuración tiene la siguiente sintáxis (cada campo esta separado por un espacio en blanco):

```
[servicio] tipo control camino-módulo [argumentos_modulos]
```

Los archivos de /etc/pam.d excluyen el campo servicio y, en su lugar, nombran el archivo de configuración después de la aplicación que sirven. Si no hay directorio pam.d, se necesita un archivo pam.conf y las aplicaciones relevantes deben listarse al inicio de cada línea.

`tipo` – es la clase de servicio o grupo de administración que suministra. Cae dentro de una de las categoría de módulos de servicios antes citadas (autenticación, cuentas, claves o sesión).

`control` – especifica la acción a tomar cuando recibe el estado de retorno de la llamada al módulo. Puede ser alguno de los siguientes valores (puede tomar adicionalmente una sintaxis más compleja, que no veremos de momento):

`required` – **conducirá a un fallo de autenticación si la llamada al módulo da un fallo. Si embargo, se invocarán el resto de módulos.**

`requisite` – **Se comporta como "required" pero provoca inmediatamente un fallo de autenticación, en lugar de llamar a los restantes módulo.**

`sufficient` – **Si esta línea tiene éxito (siempre y cuando no hubo un fallo de módulo "requerido"), la autenticación retorna inmediatamente con éxito si ejecutar los módulos restantes. Un fallo, simplemente provoca un salto a la llamada al siguiente módulo.**

`optional` – **El éxito o fallo de esas líneasno debe verse como un éxito o fallo del sistema de autenticación general salvo que solo haya una llamada a un único modulo de este tipo.**

`include` – **Indica que las líneas de un tipo dado deben leerse desde otro guión de configuración. Es la forma habitual de referirse a los archivos "common-".**

`substack` – **Similar a "include" pero los fallos o éxitos no provocan la salida del archivo completo, solo de substack.**

`camino_módulo` – **es el nombre de los módulos a invocar.**

`argumentos_módulo` – **son los parámetros opcionales a pasar a los módulos que son necesarios algunas veces para que el módulos sepa que acciones tomar si tiene éxito.**

**Los archivos individuales pueden referenciar a otros archivos que deben comprobarse usando la sintaxis:**

```
@include archivo_configuracion
```

**Esto leerá el archivo completo de configuración especificado, que es diferente del tipo de control "include", que solo lee las líneas del mismo tipo.**

- **Módulos PAM**

**Podemos ver un listado de los módulos PAM de nuestro sistema con:**

```
$ ls -w 60 /lib/security/
pam_access.so          pam_lastlog.so     pam_smbpass.so
pam_ck_connector.so    pam_limits.so      pam_stress.so
pam_cracklib.so        pam_listfile.so    pam_succeed_if.so
pam_debug.so           pam_localuser.so   pam_tally2.so
pam_deny.so            pam_loginuid.so    pam_tally.so
pam_echo.so            pam_mail.so        pam_time.so
pam_env.so             pam_mkhomedir.so   pam_timestamp.so
pam_exec.so            pam_motd.so        pam_umask.so
pam_faildelay.so       pam_namespace.so   pam_unix.so
pam_filter             pam_nologin.so     pam_userdb.so
pam_filter.so          pam_permit.so      pam_warn.so
```

```
pam_ftp.so             pam_pwhistory.so  pam_wheel.so
pam_gnome_keyring.so   pam_rhosts.so     pam_winbind.so
pam_group.so           pam_rootok.so     pam_xauth.so
pam_issue.so           pam_securetty.so
pam_keyinit.so         pam_shells.so
```

Un análisis detallado se puede ver en [7, 8]. Brevemente, alguno de estos módulos son:

`pam_access` Se encargada de la gestión de accesos al sistema basado en nombres de usuario, de *host*, de dominio o de direcciones de Internet o de red. Usa el archivo de configuración */etc/security/access.conf* para determinar que usuarios, grupos, dirección de red o terminal tienen acceso al sistema, de tal forma que cuando alguien trata de acceder al sistema, `pam_access.so`, busca en el archivo de configuración *access.conf* si existe alguna política que determine si debe ser aceptada o rechazada la conexión.

`pam_cracklib` Se encarga de comprobar el nivel de seguridad de una contraseña. Básicamente lo que hace es pasar la contraseña a través de *cracklib* para determinar si es una contraseña débil.

`pam_debug` Esta pensado para la depuración de PAM, por lo cual proporciona información de los códigos de retorno de cada módulo de la pila PAM.

`pam_deny` Este módulo puede ser utilizado para denegar el acceso. Es conveniente usarlo para denegar la entrada a otros (*other*) servicios no configurados.

`pam_echo` Está ideado para informar al usuario sobre eventos especiales. Las secuencias que comienzan con el símbolo de porcentaje (%) son expandidas siguiendo las siguientes reglas: %H: nombre de la máquina remota, %h: nombre de la máquina local, %s: nombre del servicio, %t: nombre del terminal, %U: usuario remoto y %u: nombre del usuario local.

`pam_env` Configura las variables de entorno. El archivo de configuración por defecto es */etc/security/pam_env.conf*.

`pam_exec` Este módulo PAM permite ejecutar una orden externa.

`pam_group` Este módulo no autentica al usuario, sino que le otorga pertenencia a grupos de acuerdo a lo definido en el archivo de configuración */etc/security/group.conf*. La funciones de este módulo PAM son paralelas e independientes a las del archivo de configuración */etc/group*.

`pam_lastlog` Permite mostrar información sobre la última conexión del usuario.

`pam_limits` Permite fijar los límites de los recursos del sistema que se puede obtener en una sesión de usuario. Estos límites están recogidos en el archivo de configuración `/etc/security/limits.conf`.

`pam_nologin` Impide a los usuarios iniciar sesión en el sistema cuando */var/run/nologin* o */etc/nologin* existe. El contenido del archivo *nologin* se muestra al usuario al intentar ingresar al sistema. Este módulo no tiene efectos sobre la capacidad del

usuario *root* para acceder al sistema.

`pam_permit` Es un módulo que siempre permite el acceso. Si en el proceso de autenticación no se ofrece un nombre de usuario, este se ajustará a *nobody*.

`pam_rootok` Autentica al usuario si su UID es 0. Tenga en cuenta que algunas aplicaciones pueden tener activo el *bit setuid*, el cual permite ejecutar la aplicación con los privilegios del propietario del ejecutable. Con *pam_rootok* podrá verificar si el usuario que ejecuta la aplicación es realmente el usuario *root*.

`pam_securitty` Este módulo solo permite la entrada de *root* sólo si el intento de inicio de sesión se hace desde una terminal segura, de acuerdo a lo definido en */etc/securetty*. Este módulo no afecta a usuarios no *root*.

`pam_shells` Este módulo solo permite el acceso al sistema si la shell del usuario está listada en el archivo */etc/shells*.

`pam_tally2` Este módulo mantiene un recuento de los intentos de acceso, y deniega el acceso en caso de que demasiados accesos fracasen. Un intento exitoso reinicia el contador a cero.

`pam_unix` Es el módulo PAM estándar de autenticación en UNIX. Utiliza llamadas estándar para recuperar y establecer información de la cuenta, así como la autenticación, generalmente de los archivos /etc/passwd y /etc/shadow.

`pam_warn` Este módulo permite el registro del servicio, terminal, usuario y máquina a través de las herramientas de registro del sistema.

`pam_wheel` Este módulo se utiliza para exigir la pertenencia al grupo wheel para obtener una escalada de privilegios hacia root. Por defecto solo se permite el acceso a root si el usuario solicitante pertenece al grupo wheel, en caso de que no exista dicho grupo el usuario solicitante deberá pertenecer al grupo con gid 0.

`pam_xauth` Este módulo permite el reenvío de las claves xauth (cookies) entre usuarios. Esto significa que, por ejemplo, puede configurar su sistema para ejecutar programas X desde una sesión xterm "logeado" como otro usuario como por ejemplo *root.*

- Ejemplo 1

El archivo /var/log/faillog está destinado a almacenar los fallos de autenticación de usuarios en el sistema. Con la orden `faillog` podemos visualizar su contenido pero en general la información puede que no esté actualizada (dependiendo de la configuración del sistema).

Si el sistema no esta configurado adecudamente, para conseguir que registre los fallos de login de usuarios es necesario recurrir a PAM (en distribuciones antiguas se hacía vía *login.def*). Los pasos son:

1º En el archivo */etc/pam.d/common-auth* añadir como primera línea el siguiente contenido:

```
auth  required    pam_tally.so     per_user    magic_root onerr=fail
```

si además, deseamos que no de errores, podemos añadir al final de la línea anterior "silent"

2º Para accesos via `ssh`, en el archivo */etc/pam.d/sshd* añadir inmediatamente antes de *@include common-auth*:

```
auth  required    pam_tally.so     per_user    onerr=fail
```

Si no lo esta, debemos habilitar PAM "UsePAM yes" en el archivo */etc/ssh/sshd.config* y reiniciar `ssh`.

- Ejemplo 2

Vamos a examinar ahora un ejemplo de configuración de los requisitos de autenticación del demonio de planificación `atd`.

```
$ ls /etc/pam.d/atd
auth    required        pam_env.so
@include common-auth
@include common-account
@include common-session-noninteractive
session  required    pam_limits.so
```

La primera línea llama al módulo pam_env, que si miramos en las páginas de manual es el módulo utilizado para ajustar algunas variables de entorno, que se almacenan en por defecto */etc/security/pam_env.conf*. Como el control indica "required", la configuración fallará si el módulo retorna fallo, cosa que no debería ocurrir.

Las siguientes tres líneas leen en los archivos *common-auth*, *common-account* y *commons-session-noninteractive*, para tratar con sus respectivos papeles.

La línea final llama al módulo *pam_limits*, que comprueba el archivo */etc/security/limits.conf* o el directorio */etc/security/limits.d*. Estos se utilizan para imponer límites sobre el número de usuarios que pueden usar un servicio a la vez, entre otras cosas.

Profundizando en el tema, si vemos el archivo *common-auth*, tendremos:

```
$ ls /etc/pam.d/common-auth
auth    [success=1 default=ignore]      pam_unix.so nullok_secure
auth    requisite                       pam_deny.so
auth    required                        pam_permit.so
auth    optional                        pam_ecryptfs.so unwrap
```

La primera línea indica una llamada al módulo *pam_unix* que suministra la autenticación estándar de Unix, configurada a través del archivo */etc/nsswitch.conf*. Normalmente comprobando los archivos */etc/passwd* y */etc/shadow*.
El argumento *nullok_secure* que se para al módulo unix especifica que las cuentas sin clave están bien, en tanto en cuanto la información se revisa con el archivo */etc/securetty*.

Respecto al identificador de control, indicar que obedece a una nueva sintaxis de la forma ***[valor=accion]*** que proporciona mayor control sobre como se autentica un usuario. Algunos de los posibles valores son: `success`, `open_err`, `symbol_err`, `service_err`, `system_err`, `buf_err`, `perm_denied`, `auth_err`, `cred_insufficient`, `authinfo_unavail`, `user_unknown`, `maxtries`, `new_authtok_reqd`, `acct_expired`, `session_err`, `cred_unavail`, `cred_expired`, `cred_err`, `no_module_data`, `conv_err`, `authtok_err`, `authtok_recover_err`, `authtok_lock_busy`, `authtok_disable_aging`, `try_again`, `ignore`, `abort`, `authtok_expired`, `module_unknown`, `bad_item` y `default`. Puede obtener una lista completa en */usr/include/security/_pam_types.h*.

Los posibles valores de acción pueden ser:

`ignore`: Cuando se usa en una pila de módulos, el estado devuelto por el módulo no contribuirá al código de retorno que obtiene la aplicación.

`bad`: Esta acción indica que el código de retorno debe ser considerado como indicativo de fallo del módulo. Si este módulo es el primero que falla en una pila, su valor de estado se usa para toda la pila de módulos.

`die`: Equivalente a `bad` con el efecto secundario de terminar la pila de módulos inmediatemente, devolviendo el control a la aplicación.

`ok`: Esta acción indica que el código de retorno debe contribuir al código devuelto por la pila de módulos completa

`done`: Equivalente a `ok` con el efecto secundario de terminar la pila de módulos inmediatamente, devolviendo el control a la aplicación.

`reset`: Borra toda la memoria del estado de la pila de módulos e inicia de nuevo con el siguiente módulo apilado.

Los grupos de administración `required`, `requisite`, `sufficient` y `optional`, tienen su expresión equivalente en esta nueva sintaxis [valor=acción]:

```
required: [success=ok new_authtok_reqd=ok ignore=ignore default=bad]
requisite: [success=ok new_authtok_reqd=ok ignore=ignore default=die]
sufficient: [success=done new_authtok_reqd=done default=ignore]
optional: [success=ok new_authtok_reqd=ok default=ignore]
```

Volviendo a la primera línea de *common-auth*, indica que llama al módulo *pam_unix* que suministra la autenticación estándar de Unix, configurada a través del archivo */etc/nsswitch.conf*. Normalmente comprobando los archivos */etc/passwd* y */etc/shadow.*

El argumento *nullok_secure* que se para al módulo unix especifica que las cuentas sin clave están bien, en tanto en cuanto la información se revisa con el archivo */etc/securetty*.

El campo de control `[success=1 default=ignore]` como hemos comentado sustituye a los parámetros `required`, `sufficiente`, etc. permitiendo un control más fino. En el ejemplo, si el módulo retorna éxito, se salta la primera línea que sigue. El caso por defecto, que maneja

cualesquiera otro valores de retorno, provoca que la línea sea ignorada y se sigue adelante.

La segunda línea que tiene el indicador de control "requisite" significa que si falla, la configuración completa devuelve fallo inmediatamente. También, llama al módulo *pam_deny*, que devuelve un fallo por cada invocación. Esto significa que siempre fallará. La única excepción es cuando se salte la línea, que ocurre cuando la primera línea retorna éxito.

La tercera línea es `required` e invoca al módulo *pam_permit* que retorna éxito cada vez. Esta sencillamente resetea el registro actual paso/fallo en este instante para asegurar que no hay valores previos extraños.

La cuarta línea se lista como opcional y llama a *pam_ecryptfs* con la opción `unwrap`. Se utiliza para abrir una frase de paso (passphrase) usando la clave suministrada, lo que permitirá montar un directorio privado. Esto solo es relevante cuando usamos esta tecnología opcional.

Revisando el archivo *common-account*, tenemos los siguientes elementos:

```
$ ls /etc/pam.d/common-account
account      [success=1      new_authtok_reqd=done      default=ignore]
pam_unix.so
account requisite       pam_deny.so
account requisite       pam_permit.so
```

Algunos tienen similitudes con los vistos anteriormente. La primera línea llama a una comprobación de cuenta con el módulo *pam-unix*. Los módulos pueden realizar diferentes acciones dependiendo del tipo de la llamada. Para llamadas a cuentas, *pam_unix* comprueba que la cuenta no ha expirado y no esta controlada por las restricciones de login basadas en tiempo.

Si se pasa, salta la línea siguiente, *pam_deny*, y procesa la regla `permit`. En otro caso, si falla, sigue por la línea "deny" y termina con un fallo.

Para el archivo de configuración de la sesión:

```
$ ls /etc/poam.d/common-session-noninteractive
session [default=1]         pam_permit.so
session requisite           pam_deny.so
session required            pam_permit.so
session optional            pam_umask.so
session required            pam_unix.so
session optional            pam_ecryptfs.so unwrap
```

Las primeras tres líneas pueden resultar extrañas. Para entenderlas debemos conocer el flujo del programa, ya que pueden parecer arbitrarias y redundantes. La primera de ellas siempre tiene éxito, la siguiente se salta y la tercera también tiene siempre éxito.

Están así ya que muchas configuraciones PAM son auto-generadas y pueden modificarse para suministrar más reglas sustanciales cuando disponemos de métodos de autenticación conscientes de PAM. Esto crea un marco donde podemos insertar nuevas reglas que afecten el flujo del programa a posteriori.

La cuarta línea es una invocación al módulo *pam_umask* y ser marca como opcional. Establece la

máscara de creación de archivos para la sesión. Comprobará varias ubicaciones de archivos intentando una ubicación de máscara relevante.

La quinta se requiere para llamar a *pam_unix* de nuevo. Ya que es una llamada de tipo "sesión", el módulo de unix se comporta diferente. En este caso, implementa logging con las utilidades del sistema.

La última línea es una nueva llamada a *pam_ecryptfs* que realiza una función similar a la del archivo "auth".

Un ejemplo de archivo *other* puede ser:

```
#
# The PAM configuration file for the `other' service
#
auth      required   pam_deny.so
auth      required   pam_warn.so
account   required   pam_deny.so
account   required   pam_warn.so
password  required   pam_deny.so
password  required   pam_warn.so
session   required   pam_deny.so
session   required   pam_warn.so
```

Donde el módulo *pam_deny* deniega el acceso y *pam_warn* enviará un mensaje syslog a *auth.notice*.

---

Ejercicio 4.- En el sistema que tenemos en uso, indicar los archivos de configuración existentes y comentar la misión de un par de ellos y cómo lo hacen.

---

Ejercicio 5.- (a) Modificar la configuración para que se la autenticación exija que la clave de un usuario tenga una longitud mínima. Deberemos utilizar el módulo *pam_cracklib* ¡Cuidado! pues modificaciones inadecuadas pueden dejar sin acceso a usuarios que existen en el sistema.

(b) Piensa otra modificación de tu preferencia e implementala. Por ejemplo, deshabilitar el acceso a root directo por consola, evitar que un usuario que no es el root tire el sistema, etc.

---

## 3. Contraseñas y encriptación

Las contraseñas de Linux utilizan el algoritmo de cifrado de IBM Data Encryption Standard (DES). http://www.itl.nist.gov/div897/pubs/fip46-2.htm. Estas contraseñas son de longitud fija de 8 caracteres. Pero, siempre que sea posible es preferible utilizar contraseñas MD5. MD5 es un algoritmo hash que mejora a DES en los siguientes aspectos: las contraseñas son de longitud infinita, permiten al inclusión de signos de puntuación y otros caracteres, y además, es exportable fuera de Estados Unidos, ya que no ha sido desarrollado por su gobierno.

Como vimos, en principio se almacenan, en el archivo */etc/passwd* que debe tener permisos 644. Por seguridad y, dado que es necesario mantener la lectura sobre este archivo, se utilizan las

*shadows passwords* y, en este caso, las contraseñas se almacenan en el archivo */etc/shadow* con permisos 600, es decir sin lectura para el grupo y los otros. De esta forma se impide que los usuarios normales puedan ver las contraseñas cifradas. El propietario de ambos archivos debe ser *root*.

Es conveniente cambiar periódicamente las contraseñas, tanto los usuarios como el administrador *root*. La utilidad **chage** cambia la fecha de caducidad de las contraseñas. La orden:

```
# chage -M 30 usuario
```

obliga a usuario cambiar su contraseña cada 30 días. Y la orden:

```
# chage -W 5 usuario
```

avisa a usuario que su contraseña va a expirar en 5 días.

También es conveniente periódicamente ejecutar la orden **pwconv** para asegurarnos de que todas las contraseñas tienen shadow. En ocasiones, y debido a manipulaciones anómalas, puede haber contraseñas que se quedan en */etc/passwd* en vez de en /etc/shadow. Con `pwconv` sincronizamos ambos archivos.

La integridad del archivo */etc/passwd* se realiza con la orden **pwck**, que comprueba que el formato del archivo es correcto y los datos de cada uno de sus campos son válidos.

Aunque más adelante se hablará del sistema de logs, hay que indicar que las modificaciones o intentos de modificaciones de las contraseñas quedan registradas en el archivo */var/log/messages*. Utilizando el comando **less** y los filtros adecuados se pueden controlar los mensajes correspondientes a estos cambios o intentos de cambios de contraseña.

---

Actividad 6: Crear en el sistema un usuario con las características que deseéis, entrando como ese usuario cambiar la contraseña y analizar los archivos log para ver el mensaje correspondiente.

---

Respecto a las normas para la formación de una contraseña se puede decir que es conveniente:

1. No utilizar nuestro nombre, apellidos o apodo.
2. No utilizar nombres de parientes o amigos.
3. No seleccionar una contraseña en la que se repita la misma letra o dígito.
4. Utilizar una contraseña con 8 o más dígitos.
5. Utilizar mezclas de letras mayúsculas y minúsculas.
6. Utilizar algún carácter no alfabético, como signos de puntuación.
7. Utilizar contraseñas fáciles de memorizar, y no escribirlas en ningún sitio.
8. Utilizar contraseñas que se puedan escribir rápidamente, por si alguien nos vigila.

Aunque la incorporación de las shadows paswords introduce un nivel de seguridad en el sistema, es necesario que los usuarios sean rigurosos al asignarse contraseña y no utilicen la más sencilla y fácil de recordar[4].

El administrador debe, también, periódicamente ejecutar una herramienta para comprobar de

4 Un ejemplo de como asignar una clave segura lo podemos ver en el video http://www.youtube.com/watch?v=15i6hdV_6gE.

passwords, como Crack ( http://www.crypticide.org/users/alecm/ ) o John the Ripper ( http://openwall.com/john ), para detectar claves descifrables y forzar su cambio. A continuación hay que añadir a la lista de palabras utilizada por Crack las contraseñas encontradas de los usuarios. Se pueden encontrar más programas de este tipo en: http://packetstorm.security-guide.de, junto con un gran número de diccionarios y listas de palabras. Algunos de estos programas se utilizará en sesiones posteriores.

Otra posibilidad es que la comprobación de contraseñas (utilizando una lista de palabras y una serie de reglas) se haga antes de actualizar la base de datos de contraseñas (comprobación proactiva de las contraseñas). Para ello existen herramientas como:

- **passwd+**: hace un log de todas las sesiones, errores, usuarios que han cambiado su contraseña, reglas que no cumplían las contraseñas y si ha tenido éxito o no el cambio de contraseña. En la dirección: ftp://ftp.dartmouth.edu/pub/security.
- **npasswd**: sustituto del comando passwd del sistema, más completo y eficiente. En la dirección: http://www.utexas.edu/cc/unix/software/npasswd.

Otro aspecto a tener en cuenta de cara a la seguridad relacionado con las cuentas de usuario es desactivar todas aquellas cuentas que bien no se usen o cuyo password haya expirado.

## 4.- Seguridad y control de acceso del root

En ocasiones el causante de algún desastre en el sistema es el propio administrador. Por ello, es importante que el *root* tenga su propia cuenta de usuario y que se conecte siempre como usuario normal. Cuando tenga que realizar tareas de administración puede pasar a *root* ejecutando el comando **su**.

De igual forma es interesante no incluir en la variable PATH el camino actual y así forzar a que se introduzca delante del ejecutable los caracteres ./ o bien ejecutarlos con un shell (`bash ejecutable`).

Es importante que el password del root no viaje por la red en texto plano. Debe hacerlo de forma cifrada. Evitar el uso de ordenes remotas del tipo `rlogin`, `rcp`, `rsh`,... y utilizar las ordenes `ssh` o `scp`, en las que la contraseña viaja encriptada.

Se puede utilizar la orden **sudo** para permitir a ciertos usuarios realizar tareas de administración. Para ello, hay que especificar en el archivo de configuración */etc/sudoers* que usuarios están permitidos y que acciones pueden llevar a cabo. El sistema mantiene un registro de todas las actividades realizadas por esos usuarios autorizados.

El formato general para añadir usuarios al archivo /etc/sudoers es:

```
userhost(s)=command(s)[run_as_user(s)]
```

Por ejemplo, si deseamos ceder permiso al usuario pepe para ejecutar la orden `shutdown`, deberemos especificar los siguiente:

```
pepe ALL = /etc/shutdown
```

La directiva `ALL` se utiliza como comodín para cualquier campo del archivo *sudoers*.

---

Actividad 7: Modificar el archivo *sudoers* para que un usuario determinado tenga acceso a todas las órdenes del root.

---

IMPORTANTE: Si utilizamos sudo para ceder acceso de root a un usuario para ejecutar ciertas órdenes, debemos asegurarnos de que estas órdenes no permitan "escapar" a un shell. Si eso ocurriese, el usuario en cuestión tendría acceso limitado como root a cualquier orden a través de ese shell. Por ejemplo, el editor **`vi`** permite ejecutar este shell a través de su función de escape.

En principio y como medida de seguridad, solo se debe poder acceder a la cuenta *root* desde la consola. Como ya se ha comentado, si se necesita hacer un acceso remoto a la cuenta root, entrar primero con cuenta de usuario y luego ejecutar el comando `su` para pasar a *root*. Estos intentos quedan registrados en el archivo de logs */var/log/messages*. De esta forma un posible atacante tendría que conocer el nombre de un usuario del sistema, conocer su clave y también conocer la clave del root, y añadiría dificultades para obtener privilegios remotos en el sistema.

No utilizar las ordenes `rlogin`/`rsh`/`rcp` como root. Pueden ser objeto de diversos tipos de ataques y es peligroso ejecutarlas como *root*. No crear nunca un archivo *.rhosts* para root.

En el archivo */etc/securetty* se especifican aquellas terminales (`ttyn` | `vc/n`) desde las que se puede conectar el root como tal. Se puede limitar la conexión de root, como tal, desde las terminales que se deseen. Si el root debe conectarse desde un lugar diferente de la consola y esta limitado desde */etc/securetty*, deberá conectarse como usuario y luego ejecutar la orden `su`.

También, deberemos limitar el acceso al root a través de montajes NFS (Network File System). En este caso debemos prestar especial atención al archivo */etc/export* donde se declaran los sistemas de archivos locales que se exportan vía NFS. Por defecto, el servidor NFS mapea el ID 0 (root) sobre un usuario no privilegiado, como *nobody*. Este comportamiento puede corregirse evitando a toda costa el uso de la opción `no_root_squash` del servidor NFS en el archivo */etc/export*.

## 5. Log del sistema.

Un *registro* (*logging*) es cualquier procedimiento por el que un sistema operativo o aplicación graba eventos mientras ocurren y los guarda para su estudio posterior. Los archivos de log de un servidor deben ser propiedad del usuario y grupo *root* y no tener ningún permiso para otros. Además, por seguridad, se les debería poner el flag de solo añadir:

```
# chattr +a nombre_archivo
```

Linux registra muchas operaciones y eventos importantes del sistema en archivos de texto, para su consulta. Están ubicados en el directorio */var/log/*. Entre ellos están:

- */var/log/lastlog*: almacena información sobre el último acceso (login) hecho por los usuarios.

Ejecutando la orden **`lastlog`** se ve el contenido del archivo. Si se quiere observar a un usuario en concreto hay que utilizar la opción `u`.

- */var/log/wtmp*: almacena la lista de todos los usuarios que han hecho *login* y *logout* desde que se creó el archivo. El comando **`last`** vuelca esta información. A `last` se le puede indicar la cuenta de usuario que se quiere controlar o la terminal tty.
- */var/run/utmp*: registra las entradas de los usuarios que todavía están conectados al sistema. Cada vez que un usuario se desconecta se borra la entrada correspondiente en *utmp*. El contenido de este archivo es utilizado por la orden `who`.
- */var/log/btmp*: contiene todos los intentos fallidos de conexión de los usuarios del sistema. Su contenido se visualiza con la orden **`lastb`**.
- */var/log/sudo*: registra toda la actividad de la orden `sudo`. El formato de sus entradas es `date:user:HOST=hostname:TTY=terminal:PWD=dir:USER=user:COMMAND=cmd`
- */var/log/messages*: almacena (en el orden en que se presentan) los mensajes del kernel y del sistema. Estos son manipulados por los demonios *syslogd* y *klogd*.

*syslogd* es el servicio de login para programas y aplicaciones (no para conexiones y desconexiones de usuarios). Guarda el nombre del programa, el tipo de servicio, prioridad, etc. Su archivo de configuración es */etc/syslog.conf*. En él se establece qué eventos se van a registrar y en qué archivos. La estructura de la línea del archivo de configuración es:

```
opcion.nivel_de_registro destino_registro
```

-`Opción` puede ser uno de los siguientes valores:

*auth*: servicio de seguridad que sigue la pista de cualquier acción de un usuario que requiera un nombre de usuario y una contraseña para hacer login.

*cron*: sigue los mensajes del sistema cron.

*kern*: sigue los mensajes del kernel.

*lpr*: sigue los mensajes del sistema de impresión.

*mail*: sigue los mensajes del sistema de correo.

Ejemplo: `kern.* /dev/console` envía todos los mensajes del kernel a la consola.

-`Nivel de registro` indica que las aplicaciones generan entradas en el registro con un cierto nivel. Syslogd, en función de ese nivel, puede aceptarla o rechazarla. Este campo no es necesario.

*alert*: problemas serios que requieren una atención inmediata.

*emerg*: el sistema no funciona

*crit*: condiciones críticas.

*err*: errores típicos de stderr.

*debug*: mensajes con información sobre depuración.

*info*: mensajes de información.

*warning*: avisos estándar

Ejemplo: `kern.err /var/log/messages`

-`Destino_registro` indica cual va a ser el destino de los mensajes:

- Un archivo: */ruta/archivo*
- Un terminal: */dev/ttyx*
- Una máquina remota:*@nombre_máquina*
- Uno o varios usuarios: *usuario1, usuario2,...*

* Mensajes a todos los usuarios conectados

Si se quiere monitorizar los mensajes del sistema en una consola, por ejemplo la 12, se debe editar el archivo */etc/syslog.conf* y se añade la linea:

```
*.* (TAB) /dev/tty12
```

Salir del editor guardando los cambios y reiniciar el servicio *syslog*:

```
# /etc/rc.d/init.d/syslog restart
```

---

Actividad 8: Analiza el contenidos de estos archivos de registro del sistema de prácticas y comprueba que efectivamente se registran los eventos indicados.

---

Hay programas que mantienen y manejan por sí mismos los logins. Por ejemplo, `bash` mantiene su propio histórico de órdenes. Las variables que se pueden configurar y que son utilizadas para hacer su propio log son:

1. `HISTFILE`: nombre del archivo de historial, por defecto *~nombre_usuario/.bash_history*.
2. `HISTFILESIZE`: nº máximo de órdenes que puede contener el archivo.
3. `HISTSIZE` : número de órdenes que recuerda ( con las teclas de cursor).

Estas variables se pueden configurar a nivel general, para todos los usuarios, en */etc/profile*.

Para finalizar, las órdenes **`last`**, **`lastb`** y **`lastlog`** muestran los accesos al sistema:

**`last`** – muestra las entradas y salidas del sistemas

**`lastb`** – enumera los intentos de accesos fallidos para un usuario concreto.

**`lastlog`** – muestra el último intento de conexión de red efectuado por un usuario, lo que resulta muy útil para verificar intrusiones externas.

---

Actividad 9: Analizar las conexiones al sistema de prácticas y al de casa. ¿hay o ha habido alguna conexión ajena al equipo?

---

## 7 Bibliografía


[1] M. Jang y R. Messier, *Security Strategies in Linux Platforms and Applications, 2nd Edition*, Jones & Bartlett Learning, 2017.

[2] Andreas Grünbacher "POSIX Access Control Lists on Linux", 2003. En línea, disponible en http://www.vanemery.com/Linux/ACL/POSIX_ACL_on_Linux.html.

[3 ] R. J. Hontañón, *Linux Security*, SYBEX, 2001.

[4] G. Mourani, *Securing and Organizing Linux: The Hacking Solution*, Open Network Architecture, 2002.

[5] B. Toxen, *Real World Linux Security: Intrusion Prevention, Detection, and Recovery*, Prenrice Hall, 2000.

[6] ORACLE, *Debeloper's Guide to Oracle Security*, 2011, disponible en https://docs.oracle.com/cd/E23823_01/pdf/816-4863.pdf.

[7] ORACLE, *Guía de administración del sistema: servicios de seguridad*, 2011, disponible en http://www.securitybydefault.com/2014/02/troyanizacion-de-modulos-pam.html

[8] Andrew G. Morgan y Thorsten Kukuk, "The Linux-PAM System Administrator's Guide", 2010, disponible en http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html.

## Apéndice: Cómo escribir una aplicación consciente de PAM

**Escribimos un programa denonimado *testpam.c* que utiliza la interfaz de la biblioteca PAM:**

```
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <stdio.h>

static struct pam_conv conv = {
    misc_conv,
    NULL
};

int main(int argc, char *argv[])
{
    pam_handle_t *pamh=NULL;
    int retval;
    const char *user="nobody";

    if(argc == 2) {
        user = argv[1];
    }
    if(argc > 2) {
        fprintf(stderr, "Usage: check_user [username]\n");
        exit(1);
    }
    retval = pam_start("check_user", user, &conv, &pamh);
    if (retval == PAM_SUCCESS)
        retval = pam_authenticate(pamh, 0);    /* is user really user? */
    if (retval == PAM_SUCCESS)
        retval = pam_acct_mgmt(pamh, 0);       /* permitted access? */
    /* This is where we have been authorized or not. */
    if (retval == PAM_SUCCESS) {
        fprintf(stdout, "Authenticated\n");
    } else {
        fprintf(stdout, "Not Authenticated\n");
    }
    if (pam_end(pamh,retval) != PAM_SUCCESS) {     /* close Linux-PAM */
        pamh = NULL;
        fprintf(stderr, "check_user: failed to release authenticator\n");
        exit(1);
    }
    return ( retval == PAM_SUCCESS ? 0:1 );       /* indicate success */
}
```

**que compilaremos con las bibliotecas dinámicas de pam y pam_misc:**

```
$ gcc testpam.c -o testpam -lpam -lpam_misc
```

**Configuramos PAM como:**

```
vim /etc/pam.d/check_user
```

```
 # check authorization
  auth       required     pam_unix.so
  account    required     pam_unix.so
```

**Finalmente, ejecutamos la aplicación:**

```
./testpam padam
Password: *****
```

**Authenicated**

**Por otra parte, podemos saber si un programa utiliza la funcionalidad PAM con la orden:**

```
$ ldd $(which programa) | grep libpam
```

**o tener una lista de aplicaciones que pueden usar PAM:**

```
$ ldd /{,usr/}{bin,sbin}/* | grep -B 5 libpam | grep '^/'
```