SEGURIDAD EN SISTEMAS OPERATIVOS

4º Grado en Informática

Curso 2018-19

Práctica 1.- Administración de la seguridad en Linux

Sesión 4.- SELinux (Security Enhanced Linux)

1.- Introducción

En esta prácticas, veremos los conceptos básicos y la terminología utilizada en SELinux. También veremos la arquitectura del mismo tal como se utiliza en sistemas Linux.

Para la realización de la práctica podemos instalar una versión de Fedora en una MV incluyendo en la instalación todo el soporte software para SELinux.

2.- Terminología

Para comenzar vamos a ver la terminología utilizada en SELinux:

- *Sujeto*: Un sujeto es el proceso que desea acceder a un objeto
- *Objeto*: Es un recurso como un archivo, o una interfaz de red, pero un proceso también puede ser un objeto, o incluso un sujeto mismo. Cada objeto tiene una *clase de objeto*.
- *Clase de objeto*: Cada objeto pertenece a una de las siguientes clases:
  - Objetos relacionados con archivos, como `dir`, `file`, `lnk_file`, `sock_file`
  - Objetos socket y de red, por ejemplo `socket`, `netif`, `tcp_socket`, etc.
  - Objetos relacionados con IPC, ejemplo, `msg`, `shm`.
  - Miscelánea, por ejemplo `capability`, `passwd`, etc.
- *Contexto de seguridad*: Cada sujeto u objeto en SELinux tiene un contexto de seguridad que consta básicamente de tres atributos: un *usuario*, un *rol*, y un *tipo*. Viene representado de la forma `usuario:rol:tipo`. Si se utiliza MLS/MCS (MultiLevel Security/MultiCategory Security), entonces también se utiliza el atributo *política*, existe también un atributo para el rango/categoría. Podemos ver el contexto de un archivo con diferentes órdenes que admiten la `z` entre sus opciones. Por ejemplo:

```
% ls -Z
-rw------- 1 root root  system_u:object_r:admin_home_t:s0 mi_archivo
% id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c123hal
...
```

- *Etiquetado*: Los archivos y directorios son etiquetados con un contexto de seguridad.
- *Usuario SELinux*: Los usuarios de SELinux no tienen que ver con los usuarios del sistema Linux. En verdad, hay varios usuarios del sistema que comparten un único usuario SELinux. El papel de un usuario SELinux es decidir que roles están permitidos que tome un usuario del sistema.

  Los usuarios de SELinux toman típicamente la forma `name_u`. El sufijo `_u` solo es una convención utilizada caracterizar a los usuario de SELinux. Algunos usuarios típicos son el root (que no tiene sufijo `_u`), y `system_u` que se utiliza para toda la clase de procesos del sistema, por ejemplo, el *init*.
- *Rol*: el rol se utiliza para determinar los dominios en los que puede entrar un proceso. No es importante para los objetos y existe solo para que el contexto de seguridad este completo.

  Para los roles también se utilizan sufijos, en este caso `_r`. Aquí el sufijo se utiliza por razones tradicionales y es opcional. Algunos de los roles estándares de SELinux son: `user_r` que se utiliza para usuarios comunes. El rol `staff_r` es similar a `user_r` con la diferencia que un miembro de `staff_r` puede cambiar al rol `sysadm_r`, que es el rol, que devuelve el máximo de los poderes del root, que cualquier otro rol, como por ejemplo prohibir `staff_r`. Los objetos siempre toman el rol espacial `object_r`, que está empotrado en el código, y nunca debe definirse o usarse de ninguna forma, esto puede ser un tema de seguridad, ya que un proceso en éste rol podría potencialmente entrar en cualquier dominio.
- *Tipo*: Cada objeto tiene un tipo que se utiliza para determinar que sujetos pueden acceder al objeto. El tipo es la porción más importante del contexto de seguridad. Los tipos se indican en la forma `tipo_t`.
- *Dominio*: Cuando hablamos de tipo de un proceso, lo llamaremos *dominio*. Técnicamente hablando no hay diferencia entre dominio y tipo. El acceso a un dominio debe permitirse explícitamente. Un proceso solo puede entrar en un dominio específico en su ejecución.
- *Permisos*: Los permisos definen la forma en la que un sujeto puede acceder a un objeto. Hay varios tipos de permisos, algunos los conocemos de Linux como el de lectura (*read*) o escritura (*write*), pero también hay otros que permiten operaciones de grano más fino como añadir (*append*) u obtener los atributos (*getattr*). También hay permisos específicos para clases de objetos, como *create* o *bind* para sockets, o *add_name* para crear o mover una archivo a un directorio. Podemos ver una lista de los permisos posibles en la dirección http://www.selinuxproject.org/page/ObjectClassesPerms.
- *Vector de acceso*: es un conjunto de permisos que un sujeto puede ejecutar sobre un objeto.
- *Alias*: Se puede definir para tipos. Tiene sentido, si cambia la política, por razones de compatibilidad. Por ejemplo, para hacer los que los tipos `mozilla_t` y `chrome_t` signifiquen lo mismo, podemos hacer:

```
type moxilla_t;
type alias mozilla_t alias chrome_t;
```

  o

```
type mozilla_t alias chrome_t
```

- ***Atributo***: Los atributos se pueden usar para agrupar tipos. Como ejemplo, vamos a agrupar atributos para construir una *regla allow*:

```
attribute config_file;        #declara el atributo config_file
# Metodo 1 para asignar tipos a atributos
type atc_t;
type shadow_t;
typeattribute etc_t config_file;
typeattribute shadow_t config_file;
# Metodo 2 para asignar tipos a atributos
type etc_t, config_file;
type shadow_t, config_file;
# Uso del atributo en la regla allowrule
allow sysadm_t config_file:file read;
```

  En este ejemplo, sin la definición del atributo deberíamos escribir dos veces la regla. Un atributo se expande internamente a la lista de tipos que tiene.

- ***Política***: Utilizamos el término política para describir el conjunto de reglas que definen *si* y *cómo* un sujeto puede acceder a un objeto.

- ***Re-etiquetado***: Una política puede permitir procesos confiables para cambiar el etiquetado de un objeto. Por ejemplo, si entramos en el sistema, el programa *login* cambia la etiqueta del terminal de control. Los permisos que están relacionados con los cambios de tipos son `relabelto` y `relabelfrom`. Por ejemplo, la regla siguiente indica a un proceso SELinux en el dominio `sysadm_t`, que un archivo de la clase objeto `chr_file`, debería obtener el tipo `sysadm_tty_device_t` cuando se re-etiqueta:

```
type_change sysadm_t tty_device_t:chr_file sysadm_tty_device_t;
```

- ***Imposición de Tipo*** (*type enforcement*): de las definiciones de usuario, rol y tipo, podemos ver que la más importante es el tipo: se utiliza para definir qué permisos de un sujeto de cierto tipo fuente están permitidos sobre un objeto de otro tipo objetivo. Esta tecnología se denomina *Type Enforcement*™.

- ***Control de acceso basado en roles***: Para determinar que permisos tiene un usuario del sistema sobre cierto objeto, este debe tener acceso a cierto dominio que le permite acceder al objeto especificado. Se define un rol para agrupar a aquellos usuarios que tienen derechos de acceso comunes. Cada usuario que puede cambiar este rol puede entonces realizar una *transición* al dominio necesario, si le está permitido.

- ***Control de acceso basado en usuario***: UBAC es un mecanismo que SELinux incorporó recientemente. Se basa en restricciones, que aplica a la porción de usuario involucrado en dos contextos de seguridad, uno del proceso que accede y otro del objeto. Para permitir el acceso a objetos con restricciones, ambas porciones de usuario deben ser iguales. UBAC no es una sustitución de RBAC, pero puede habilitarse adicionalmente.

3.- Archivos y sistemas de archivos SELinux

En SELinux hay dos ubicaciones importantes: el directorio de configuración /etc/selinux, y el sistema de archivos montado en /selinux.

- /selinux

Este sistema de archivo contiene información sobre el estado actual del sistema SELinux. Existen varios archivos que puede utilizarse para obtener/establecer estados y valores del sistema. Algunos ejemplos:

  - */selinux/enforce*: contiene el estado de cumplimiento (*enforcing*)
  - */selinux/booleans*: contiene un archivo por cada Booleano del sistema.
  - */selinux/class*: contiene subdirectorios para cada clase de objetos (archivos, directorios, socket, etc.) y los permisos disponibles.

- */etc/selinux/*

Este directorio contiene la configuración específica de SELinux y la política binaria. El archivo */etc/selinux/config* contiene el estado del sistema en el arranque, en concreto, el tipo de política y el modo de obligatoriedad (*enforcement*).

Existe también un directorio por cada política, es decir, para una política estricta esta */etc/selinux/strict*, que a su vez contiene la política binaria, los módulos de política activa e información sobre los contexto de usuarios SELinux y archivos.

3.1.- Etiquetado de archivos

Cada objeto y sujeto debe etiquetarse correctamente. Un proceso (sujeto) hereda el contexto de seguridad de su padre, pero ¿qué pasa con los archivos?. En SELinux, hay cuatro mecanismos para etiquetar un objeto relacionado con archivos:

1. Etiquetado basado en atributos extendidos (`xttras`).
2. Etiquetado basado en tareas.
3. Etiquetado basado en transiciones.
4. Etiquetado generalizado.

Si un sistema de archivos soporta atributos extendidos, como ext2, ext3, xfs, y reiserfs, el contexto de seguridad se almacena en ellos. Un archivo en estos sistemas de archivos hereda normalmente la

porción de usuario del contexto de seguridad del proceso creador. La porción rol será siempre `object_r`. La porción tipo es heredada del directorio que contiene al archivo, o si existe una regla de transición de tipo para el dominio del proceso y el tipo del directorio que contiene el archivo, se utiliza el tipo especificado en esa regla.

El etiquetado basado en tareas se utilizan pipes (cauces), el contexto de seguridad se hereda del proceso creador.

El etiquetado basado en transiciones es similar al basado en tareas, pero utiliza reglas de transición de tipos para el etiquetado, como el etiquetado basado en *xattrs*. Este se utiliza por ejemplo en los sistemas de archivos *tmpfs*, *shm* y *devpts*.

El etiquetado generalizado se utiliza en seudo-sistemas de archivos como */proc*, */sysfs*, o */selinux* y para sistemas de archivos que no soportan atributos extendidos como FAT o ISO9660.

Si deseamos cambiar la etiqueta por defecto de archivos no etiquetados, la orden `mount` soporta en SELinux las opciones `context=`, `fscontext=`, y `defcontext=`.

Otros objetos como las interfaces de red o puertos también se etiquetan, pero las reglas son similares a las de los objetos relacionados con archivos.

3.2.- Obligatorio, permisivo o deshabilitado

Si SELinux esta habilitado, podemos ejecutar SELinux en dos modos: permisivo (*permissive*) y obligatorio (*enforced*). El modo por defecto se establece en el archivo */etc/selinux/config*. Este puede sobrescribirse al arrancar el kernel con la opción de arranque `enforcing=[0|1]`. Puede obtener y ajustar este estado con la orden `getenforce` y `setenforce`. La orden `sestatus` nos permite ver si SELinux esta o no habilitado, en qué modo y qué política está cargada.

En el estado *enforcing*, la política es obligatoria y los accesos se basan en la política. En el estado *permissive* la política no es obligatoria, pero se generan mensajes de logs como en el modo obligatorio. El estado *disable* (desabilitado) no es un verdadero estado como los anteriores, en realidad lo que hace es deshabilitar SELinux. También se puede hacer con la opción de arranque `selinux=0`. Si el kernel esta configurado, esto se puede hacer también en ejecución.

3.3.- Dominios, tipos y transiciones

¿Qué ocurre cuando tenemos activo SELinux y ejecutamos una simple orden como `ls`? Primero se ejecuta el sistema DAC clásico: lo podemos ejecutar si tenemos permiso de ejecución del binario */bin/ls* y de lectura del directorio en el que lo ejecutamos. Luego se aplica MAC que comprueba si existe una regla que permite al dominio de nuestro proceso shell ejecutar `/bin/ls`. En este caso, se ejecutará. Si no hay regla que lo permita, no podremos ejecutarlo.

/bin/ls es una orden que no tiene un riesgo potencial de seguridad, pero que ocurre si intentamos

ejecutar */bin/passwd*. Ahora es algo más complicado, ya que un error en el programa puede tener consecuencias desastrosas en la seguridad del sistema. Vemos la siguiente situación:

```
SEStrict ~ # id -un
root
SEStrict ~ # id -Z
root:sysadm_r:sysadm_t
SEStrict ~ # cat /etc/shadow
cat: /etc/shadow: Permission denied
SEStrict ~ # ls -lZ /bin/passwd /etc/shadow
-rws--x--x+ 1 root root system_u:object_r:passwd_exec_t 32844 Oct 28 15:55
/bin/passwd
-rw--w----+ 1 root root system_u:object_r:shadow_t 694 Dec 11 14:43
/etc/shadow
```

En este caso, podemos ver que ni el root en el rol `sysadm_r` puede leer directamente en archivo */etc/shadow*. Incluso lo más interesante es observar que `sysadm_r` es el rol más potente de los sistemas SELinux.

Pero nos planteamos ¿cómo puede la orden `passwd` acceder al archivo */etc/shadow*? Ya que la obligación de tipos (TE) es el mecanismo que define qué accesos están permitidos, vamos a analizar qué pasa. Si miramos la política, veremos que no hay una regla que permita al dominio `sysadm_t` acceder al objeto de tipo `shadow_t`. Dado que SELinux deniega cualquier acceso por defecto, deberá existir alguna regla que lo permita. Por supuesto encontramos alguna: una regla permite a `passwd_t` acceder a */etc/shadow*, pero no a `sysadm_t`. Por tanto ¿cómo puede */bin/passwd* acceder a `/etc/shadow` siempre incluso en el dominio `sysadm_t`?

El proceso que ejecuta un programa hereda el contexto de seguridad del proceso llamador. Si nuestra sesión del shell actual tiene como contexto de seguridad `user_u:user_r:user_t`, se ejecuta en el dominio `user_t`. La orden que ejecutemos, heredará el dominio `user_t`. Pero si necesitamos definir reglas especiales para programas como `passwd`, debemos establecer que estas reglas son únicas para este programa.

Revisemos en detalle que pasa al ejecutar `passwd`. Primero se aplicará DAC pero la diferencia radica en la parte MAC. Lo primero que se hace es comprobar que podemos ejecutar el archivo. Si esta permitido se ejecutará. Si nuestro proceso shell se ejecuta en el dominio `user_t` una regla *allow* se podría parecer a

```
allow user_t passwd_exec_t:file { getattr execute };
```

Ahora viene la parte especial: se debe producir una transición de dominio para ejecutar la orden en el dominio `passwd_t`. Para ello, necesitamos una regla intermedia dado que no podemos introducir `passwd_t` directamente. Vamos a darle al dominio `passwd_t` un permiso especial `entrypoint` sobre objetos con el tipo `passwd_exec_t` de la clase `file`. Este tique para el proceso shell se parece a

```
allow passwd_t passwd_exec_t:file entrypoint;
```

Esta regla define a que archivos esta permitido entrar desde el dominio `passwd_t`. Pero aún no es suficiente: esta regla define algo parecido a una puerta pero con un vigilante. Para que el vigilante nos deje pasar, necesitamos un permiso especial, como una identificación. La tarjeta para el proceso shell se parece a

```
allow user_t passwd_t:process transition;
```

Resumiendo: tenemos */etc/shadow* de tipo `shadow_t`, que solo puede leerse por procesos en el dominio `passwd_t`, en el que solo se puede entrar ejecutando un archivo de tipo `passwd_exec_t`.

Nos falta un tema que resolver. Todas las reglas necesarias para la transición y ejecución están preparadas, pero cómo se realiza realmente la transición para */etc/passwd* al dominio correcto. Si `passwd` esta preparado para trabajar con SELinux, puede hacerlo él mismo. Pero como no es el caso, nos falta una última regla para que el sistema gestione la transición automáticamente:

```
type_transition user_t passwd_exec_t:process passwd_t;
```

Esto hace que un proceso creado a partir de un archivo de tipo `passwd_exec_t` haga la transición a `passwd_t` automáticamente.

Pero ¿por qué necesitamos todo este esfuerzo? Si */bin/passwd* es el único archivo de tipo `passwd_exec_t` del sistema, es obviamente la única forma de acceder a */etc/shadow*, y por consiguiente ningún otro proceso estará en condiciones de acceder a él. Si encontramos un error en */etc/passwd*, seremos capaces de dañar */etc/shadow*, pero esto cae mas allá del alcance de SELinux y es un problema que solo puede resolver el diseñador del programa.

3.4.- Roles

Si bien los roles no son el principal mecanismo de acceso de SELinux, pueden usarse para limitar el uso de los dominios. Como los roles son asignados a usuarios SELinux, un rol define que dominios coexisten juntos con qué porción de usuario.

Para que el programa */bin/passwd* funciones, necesitamos una quinta regla. Una regla que hace que el contexto de seguridad `user_u:user_r:passwd_t` sea válido:

```
role user_t types passwd_t;
```

Para los roles también existen transiciones de roles, pero no son tan frecuentemente usadas como las transiciones de dominios. Una transición de rol se necesita por ejemplo cuando un demonio se arranca por el administrador del sistema

```
role_transition sysadm_r daemon_exec_t system_r;
```

Esta regla establece que si un proceso con rol `sysadm_r` ejecuta un archivo de tipo `daemon_exec_t`, el rol debe cambiar a `system_r`, de esta forma el demonio con este tipo siempre se ejecuta con el mismo rol.

Existe una declaración *allow* que se aplica a roles, y aunque es parecida a la que vimos para tipos, son dos cosas diferentes:

```
allow sysadm_r system_r;
```

3.5.- Booleanos

Si una regla de política no satisface nuestras necesidades, o deseamos un comportamiento diferente, afortunadamente podemos cambiar la política y cargarla en el kernel en ejecución. Pero incluso si esto es mejor que rearrancar el kernel en cada cambio, sería mejor no tener complicarnos con la escritura de políticas, analizar los logs, mediar en la necesidad de reglas especiales, etc.

Para ciertas reglas existen booleanos. Los booleanos son variables SELinux que podemos ajustar para cambiar el comportamiento de la política sobre la marcha sin tener que recargarla. Podemos obtener una lista de los booleanos disponibles con **`getsebool -a`**.

En la política un booleano se define con la declaración **`bool`** seguido del nombre del booleano y por último se indica el valor por defecto, que debe ser true o false. A continuación se muestra la definición y uso de un booleano:

```
bool user_ping flase;

#...
if (user_ping) {
      #permite ping para usuarios regulares
} else {
      #opcionalmente hacer otra cosa
}
```

Como vemos, en este ejemplo usamos la declaración `if`, si bien no todas las clases de reglas la admiten. Tampoco se admiten las declaraciones anidadas, pero hay algunos operadores booleanos que pueden usarse.

3.6.- Dominación

Se puede definir un rol para dominar a otros roles. Esto se puede hacer con la declaración **`dominance`**. Si un rol domina a otros, este hereda todas sus asociaciones de tipos. En el ejemplo siguiente, `uber_r` gana todos los accesos que los roles `sysadm_r` y `secadm_t` antes de la declaración *dominance*.

```
dominate {rol uber_r {role sysadm_r; rol secadm_r}; }
```

Cada tipo asociado con, por ejemplo, `sysadm_r`, después de esto no afecta al rol `uber_r`. Una dominación puede utilizarse en una declaración de restricción para expresar una relación entre dos roles. Existen cuatro operadores de relación de dominación:

- `dom` – r1 se definió para dominar a r2
- `domby` – r2 se definió para dominar a r1
- `eq` – lo mismo que ==
- `incomp` – ambos roles no se dominan el uno al otro.

3.7.- Restricciones

Si bien todos los acceso en SELinux deben permitirse, existen ocasiones que deben usarse mecanismo para no permitir un permiso permitido, es decir, restringimos el permiso. Esto se hace con la declaración **`constrain`**:

```
constrain  process transition ( u1 == u2);
```

Esta regla restringe el premiso de transición para la clase objeto `proceso` y solo la permite para un proceso cuya porción de usuario de ambos contexto de seguridad es igual. Las palabras clave `u1` y `u2` indican, respectivamente, la porción de usuario del contexto de seguridad del sujeto y del objeto. También existen las palabras clave `r1`, `r2`, `t1` y `t2`, que se aplican a las porciones de los roles y tipos del sujeto y del objeto. En lugar del operador anterior, también podemos utilizar el operador != o alguno de los operadores de dominación, si comparamos los roles. El lado izquierdo de la comparación siempre debe ser una palabra clave y el derecho puede ser una palabra clave o un identificador.

Otro mecanismo de restricciones es la declaración **`validatetrans`** que se puede usar para un controlar además la habilidad para re-etiquetar un objeto. Solo se menciona aquí por completitud.

4.- La política

Si bien existen muchas clases de acceso a un sistema, un programa necesita solo unas cuantas clases. El comportamiento de SELinux es denegar cada acceso y una buena regla debería solo permitir una clase de acceso para el proceso que lo necesita.

4.1 Política de referencia

Existen muchos programas que se ejecutan normalmente en Linux por lo que no tiene sentido reinventar la rueda cada vez que instalamos SELinux. Afortunadamente hay reglas básicas para los programas utilizados comúnmente, tales como `mount`, `ifconfig`, etc. Estas reglas estándares configuran la política de referencia o base.

4.2 Configurando la política

Construir una política es como construir un kernel. Primero y normalmente, descargaremos las fuentes de la política de referencia que contienen las reglas para la mayoría de las cosas. La política de referencias estará dividida en varios módulos de fuentes de política. Existen módulos fuentes para muchas tareas, por ejemplo, para aplicaciones como Mozilla o demonios como `distcc`, pero

también para roles como `sysadm_r`. Para que estan fuentes sean utilizables por el kernel, deben compilarse a su forma binaria. Esto se realiza con el compilador de la política **`checkpolicy/checkmodule`**. Los módulos creados tienen la extensión .pp. Nosotros no utilizaremos la política compilada, sino que utilizaremos makefiles, que tiene cuidado de todo.

Podemos construir una política monolítica, que es un gran módulo. O podemos construir una política modular, que consiste en un módulo base y opcionalmente algunos módulos cargables. Esta última forma tiene la ventaja de que no necesitamos reconstruir toda la política para cualquier pequeño cambio que hagamos ya que podemos cargar módulos para los programas conforme los necesitemos.

Si compilamos una política modular, el módulo base contendrá los módulos fuentes más importantes, es decir, que que contendrán las reglas necesarias en tiempo de carga.

Finalmente, podemos activar/desactivar un módulo con la orden **`semodule`**. Las opciones más importantes de esta orden son:

`-R` : recarga la política.

`-i modulo.pp` : instala un nuevo módulo política.

`-u module.pp` : actualiza un módulo de política existente.

`-l` : lista los módulos actualmente cargados.

5.- Creación de usuarios

```
root@SEStrict ~ # useradd -m newuser
root@SEStrict ~ # semanage user --roles 'user_r staff_r ' --prefix user
          --add newuser_u
root@SEStrict ~ # semanage login --add --seuser newuser_u newuser
root@SEStrict ~ # passwd newuser
New UNIX password:
BAD PASSWORD: it is WAY too short
Retype new UNIX password:
passwd: password updated successfully
root@SEStrict ~ # chcon -R -u newuser_u /home/newuser/
```

Utilizamos `useradd` para crear un usuario a Linux. La orden `semanage` añade un usuario SELinux al sistema y le asigna los roles `user_r` y `staff_r`. La orden `semanage login` conecta el usuario SELinux con nombre `newuser_u` con el usuario Linux de nombre `newuser`. La orden `passwd` asigna una clave para el usuario, como estamos acostumbrados. Por último y no menos importante, cambiamos el contexto de seguridad para el directorio home del usuario con la orden `chcon`.

Es posible cambiar el rol con la orden `newrole`. Por ejemplo, podemos cambiar de nuestro rol al de administrador de la siguiente manera:

```
root@SEStrict ~ # newrole -r sysadm_r
Authentication root.
```

```
Passwd:
```

En la mayoría de los sistemas SELinux, podemos encontrar los siguientes roles:

`user_r` Se utiliza para usuarios restringidos. Este rol solo permite tener procesos con los tipos específicos para aplicaciones de usuario final. Para este usuario no se permiten tipos privilegiados, como por ejemplo, los usados para conmutar Linux.

`staff_r` Se utiliza para tareas de operador no críticas. Se limita a tareas como el anterior pero tiene permitidos algunos pocos tipos privilegiados. Es el rol por defecto para el operador.

`sysadm_r` Se utiliza para tareas de administración del sistema y es muy privilegiado. Sin embargo, algunos tipos de aplicaciones de usuario final (especialmente los tipo que se utilizan para software potencialmente vulnerable o no-confiable) pueden no soportarlo al objeto de intentar mantener el sistema libre de infecciones.

`system_r` Se utiliza para demonios y procesos en *backgruond*. Es poco privilegiado, pero sin embargo no se permite en este rol aplicaciones de usuario final y otras tareas administrativas.

`unconfined_r` Se utiliza para usuarios finales y tiene permitido un limitado número de tipos, pero esos tipos son muy privilegiados ya que están pensados para ejecutar cualquier aplicación por un usuario de una forma más o menos confinada (no restringida por las reglas SELinux). Este rol solo esta disponible si el administrador del sistema quier proteger ciertos procesos (principalmente demonios) mientas mantiene el resto de operaciones del sistema casi sin tocar por SELinux.

En sistema como Fedora también podemos encontrar los roles **`guest_r`** y **`xguest_r`** (ver documentación del sistema).

Ejercicio 2.1 Crear un usuario SELinux denominado **`admin`** con el rol **`sysadm_r`**.

6.- Utilizando booleanos

Podemos ver y cambiar booleanos con las órdenes `getsebool` y `setsebool`. En el ejemplo, listamos primero los booleanos definidos y a continuación modificamos el valor de uno de ellos, `user_dmesg` con la opción `-P` que hace la modificación permanente, es decir, sobrevivirá a un rearranque del sistema.

```
root@SEStrict ~ # getsebool -a
...
use_nfs_home_dirs --> off
use_samba_home_dirs --> off
user_direct_mouse --> off
```

```
user_dmesg --> off
user_ping --> off
user_rw_noexattrfile --> off
...
root@SEStrict ~ # setsebool -P user_dmesg=on
root@SEStrict ~ # getsebool user_dmesg
user_dmesg --> on
```

7.- Inspeccionando los archivos logs

Los archivos logs relevantes para los administradores de SELinux pueden variar según la distribución. Normalmente, en sistemas SELinux ejecutaremos el demonio `auditd`, que registra las notificaciones en el archivo configurado en */etc/audit/auditd.conf*. Si `auditd` no se está ejecutando, por ejemplo durante el arranque antes de que se ejecute el demonio, o si no lo usamos, todos los mensajes relevantes de SELinux son manejados por el demonio `syslog`. Cualesquiera de los archivos que inspeccionemos, las entradas más interesantes serán probablemente las que obtenga AVC (Cache de Vectores de Acceso). Normalmente revisaremos estos archivos cuando algo no funciones o cambiemos la política.

Para entender los mensajes,vamos a generar uno usando la orden `su` para convertirnos en root desde un usuario no privilegiado, o cuando ejecutamos `ls /proc/1`, suponiendo que estamos en estado *enforcing*. ¿Qué aspecto tiene el mensaje de denegación? Si miramos en el mensaje en los archivos de auditoría bien directamente, bien con la orden **`ausearch`** (si esta activo el sistema de auditoría de Linux):

```
# /sbin/ausearch -m avc -ts recent
----
time->Fri May 31 20:05:15 2013
type=AVC msg=audit(1370023515.951:2368): avc:  denied  { search }
  for  pid=5005 comm="dnsmasq" name="net" dev="proc" ino=5403
  scontext=system_u:system_r:dnsmasq_t
  tcontext=system_u:object_r:sysctl_net_t tclass=dir
```

En el ejemplo mostramos los mensajes de denegación, dado por la AVC. Este mensaje lo podemos descomponer en su correspondientes elementos que viene dado por la siguiente Tabla 1.

Con la información de la Tabla 1, el mensaje anterior lo podemos leer como "SELinux ha denegado la operación de búsqueda del proceso `dnsmasq` (con PID 5005) contra el directorio "net" (con inodo 5403) dentro del dispositivo proc. El proceso `dnsmasq` se ejecuta con la etiqueta `system_u:system_r:dnsmasq_t` y el directorio "net" con la etiqueta "`system_u:object_r:sysctl_t`".

Tabla 1.- Elementos de un registro de la AVC.

Field name	Description	Example
(SELinux action)	This is the action that SELinux took (or would take if run in the `enforcing` mode). This usually `denied`, although some actions are explicitly marked for auditing and would result in `granted`.	`denied`
(permissions)	These are the permissions that were checked (action performed by the process). This usually is a single permission, although it can sometimes be a set of permissions (for example, `read write`).	`{ search }`
Process ID	This is the ID of the process that was performing the action.	`for pid=5005`
Process name	The process name (command). It doesn't display any arguments to the command though.	`comm="dnsmasq"`
Target name	It is the name of the target (resource) that the process is performing an action on. If the target is a file, this usually is the filename or directory.	`name="net"`
Target device	It is the device on which the target resource resides. Together with the next field (inode number) this allows us to uniquely identify the resource on a system.	`dev="proc"`
Target file inode number	It is the inode number of the target file or directory. Together with the device, this allows us to find the file on the filesystem.	`ino=5403`
Source context	It is the context in which the process resides (the domain of the process).	`scontext=system_u:system_r:dnsmasq_t`
Target context	It is the context of the target resource.	`tcontext=system_u:object_r:sysctl_net_t`
Resource class	It is the class of the target resource, for example, a directory, file, socket, node, pipe, file descriptor, file system, capability, and so on.	`tclass=dir`

Algunas denegaciones tienen diferentes campos, como por ejemplo:

```
avc: denied { send_msg } for msgtype=method_call
  interface=org.gnome.DisplayManager.Settings
  member=GetValue dest=org.gnome.DisplayManager
  spid=3705 tpid=2864
  scontext=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
  tcontext=system_u:system_r:xdm_t:s0-s0:c0.c1023 tclass=dbus
```

Si bien los campos son diferentes, es aún legible y significa que "SELinux ha denegado al proceso con PID 3705 invocar la llamada al método remoto Dbus (el valor método "GetValue" de la interfaz "org.gnome.Display.Manager.Settings" contra la implementación "org.gnome.DisplayManager" ofrecida por el proceso con PID 2864. El proceso fuente se ejecuta con la etiqueta "`unconfined_u:unconfined_r:unconfined_t:s0-s0.c0.c1023`" y el proceso objetivo con la

etiqueta "`system_u:system_r:xdm_t:s0-s0.c0.c1023`"

Dependiendo de la acción y la clase de objetivo, SELinux utiliza campos diferentes para dar toda la información que necesitamos para detectar el problema. Por ejemplo, la siguiente denegación se produce por que "la base de datos PostgreSQL fue configurada para escuchar por un puerto que no es por defecto (6030 en lugar del puerto por defecto 54329".

```
avc: denied  { name_bind } for  pid=23849
  comm="postgres" src=6030
  scontext=system_u:system_r:postgresql_t
  tcontext=system_u:object_r:unreserved_port_t
  tclass=tcp_socket
```

Ejercicio 2.2: localiza algunos mensajes de los logs de tu sistema, o genera alguno, y describe la denegación que producen.

Otra forma de determinar que algo no funciona podemos utilizar `audit2allow`. En siguiente ejemplo, invocamos varias veces a la orden, pero antes de hacerlos recargamos la política con `semanage -R`**. La primera vez que utilizamos** `audit2allow` **utilizamos las opciones:** `-a` **para indicar que utilice como entrada auditlog, y** `-l` **para considerar solo las entradas desde la última recarga de la política.**

```
root@SEStrict ~ # semodule -R
root@SEStrict ~ # audit2allow -la
#============= user_t ==============
allow user_t proc_kmsg_t:file getattr;
root@SEStrict ~ # audit2allow -laR

      require {
            type user_t;
      }
      require {
            type user_t;
      }
#============= user_t ==============
kernel_getattr_message_if(user_t)
root@SEStrict ~ # audit2allow -lae
#============= user_t ==============
# audit (1230787789.236:41):
# scontext="user_u:user_r:user_t" tcontext="system_u:object_r
proc_kmsg_t"
# class="file" perms="getattr"
# comm="ls" exe="" path=""
# message="type=AVC msg=audit (1230787789.236:41): avc: denied {
getattr } for
# pid=4961 comm="ls" path="/proc/kmsg" dev=proc ino=4026531948
# scontext=user_u:user_r:user_t tcontext=system_u:object_r:
proc_kmsg_t
# tclass=file"
allow user_t proc_kmsg_t:file getattr;
```

Podemos ver una regla que, si se usa, podría permitir al dominio `user_t` obtener atributos de */proc/kmsg*.

La segunda vez que utilizamos la orden en el ejemplo anterior, utilizamos `-R` que genera una salida del tipo "estilo de política de referencia". Esto significa que buscará en los archivos de interfaz y mirará si encuentra una macro que hace exactamente lo mismo que una regla *allow*. También generará las órdenes requeridas necesarias. Podemos observar que en algunas versiones más antiguas se imprime dos veces la declaración solicitada). Para ver por qué esta regla se ha generado realmente, utilizamos la opción `-e` en el tercer ejemplo. `Audit2allow` imprime el mensaje ópticamente preparado, lo que es muy útil para ver si deseamos utilizar esta regla o necesitamos alcanzar nuestro objetivo de forma diferente.

```
root@SEStrict ~/userkmsg # audit2allow -laRm userkmsg -o userkmsg.te
root@SEStrict ~/userkmsg # bzcat /usr/share/doc/selinux-base-policy
-20081210/ Makefile.example.bz2 > Makefile
root@SEStrict ~/userkmsg # ls
Makefile userkmsg.te
root@SEStrict ~/userkmsg # #check userkmsg.te if it really does what
you want
root@SEStrict ~/userkmsg # make
Compiling strict userkmsg module
/usr/bin/checkmodule: loading policy configuration from tmp/userkmsg.
tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 8) to tmp
/userkmsg.mod
Creating strict userkmsg.pp policy package
rm tmp/userkmsg.mod.fc tmp/userkmsg.mod
root@SEStrict ~/userkmsg # ls
Makefile tmp userkmsg.fc userkmsg.if userkmsg.pp userkmsg.te
root@SEStrict ~/userkmsg # semodule -i userkmsg.pp
root@SEStrict ~/userkmsg # semodule -l|grep userkmsg
userkmsg 1.0
root@SEStrict ~/userkmsg #
```

8.- Políticas

La orden **`sestatus`** nos permite ver las políticas de SELinux

```
# sestatus
SELinux status:                 enabled
SELinux mount:                  /selinux
SELinux root directory:         /etc/linux
Loaded policy name:             ubuntu
Current mode:                   enforcing
Mode from config file:          permissive
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

De la información anterior, podemos ver que SELinux esta habilitado, que la política es ubuntu. Esta política se establece en el arranque y esta definida en el archivo */etc/selinux/config*. La mayoría de

las distribuciones utilizan una política de referencia gestionada como un proyecto de software libre (http://oss.tresys.com/projects/refpolicy/) que les permite disponer de una política funcional sin tener que reescribirla desde cero.

La línea "Current mode" nos indica el estado de SELinux (disable, enforcing o permissive). El estado también se define en el arranque a través del archivo */etc/linux/selinux/config*. Si el estado es deshabilitado, el sistema se comporta como si no hubiésemos arrancado SELInux. Si es permisivo, SELinux esta activo pero no obliga la aplicación de la política. En su lugar cualquier se notifica cualquier violación de la política. Si esta *enforcing*, se obliga la política y las violaciones son denegadas y reportadas. Podemos cambiar el estado modificando el archivo y rearrancando el sistema. También podemos hacerlo temporalmente con la orden `setenforce` (ver manual) o escribiendo el valor 0 o 1 en el archivo */selinux/enforce*.

Ejemplo 2.3.- Indicar la orden que debemos ejecutar para pasar de un estado permisivo a uno obligatorio.

Es posible deshabilitar la protección de SELinux para un servicio concreto en lugar de deshabilitar SELinux completamente. Por ejemplo, podemos ejecutar un servidor DLNA, de tipo `minidlna_t`, en modo permisivo con la orden:

```
# semanage permissive -a minidlna_t
```

También podemos ver el estado de MLS: *enable* o *disable*. Si esta deshabilitado, el contexto SELinux no tiene el cuarto campo con la información de rango/categoría que indicábamos al inicio(en algunos documentos se habla de sensibilidad) en él. Si está habilitada, encontraremos el campo sensibilidad. Por ejemplo en nuestro caso como esta habilitada, cuando por ejemplo hacemos `id -Z` tendremos

```
# id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c255
```

donde el último campo es el nivel de sensibilidad.

La información que muestra la línea "Policy deny_unkown status" establece que hacer con los casos de permisos desconocidos, lo que suele ocurrir en los kernels más modernos de Linux donde se pueden introducir permisos que la política no conoce. En este caso, esta información nos dice que SELinux permitirá el acceso. Además, de *allow*, podemos encontrar los valores "*deny*" para denegar el acceso asumiendo que nadie tiene permitido realizar esta acción, o "*reject*" que para de cargar la política y detiene el sistema.

Podemos escribir una política SELinux para que sea muy estricta, limitando a las aplicaciones lo más próximo posible a su comportamiento real, pero también puede escribirse de forma liberal en cuanto a qué pueden hacer las aplicaciones. Un concepto existente en algunos sistema SELinux es la idea de dominio sin confinar (*unconfined*). Cuando esta habilitado este dominio, ciertos contextos de procesos (dominios) tiene permitido hacer casi cualquier cosa (limitado por DAC) y solo unos pocos seleccionados están confinados verdaderamente en sus acciones. Podemos ver si en nuestro sistema

esta habilitado de la forma

```
#seinfo -tunconfined
      unconfined_t
```

Por tanto, la orden nos devuelve **`unconfined_t`** que indica que esta habilitado, Si no lo estuviera devolvería error.

La Tabla siguiente muestra las políticas establecidas según las distribuciones:

Distribution	Policy store name	MLS?	deny_ unknown	Unconfined domains?	UBAC?
Gentoo	`strict`	No	`denied`	No	Yes (configurable)
Fedora 19	`minimum`	Yes (only `s0`)	`allowed`	Yes, but limited	No
Gentoo	`targeted`	No	`denied`	Yes	Yes (configurable)
Fedora 19	`targeted`	Yes (only `s0`)	`allowed`	Yes	No
Gentoo	`mcs`	Yes (only `s0`)	`denied`	Yes (configurable)	Yes (configurable)
Gentoo	`mls`	Yes	`denied`	Yes (configurable)	Yes (configurable)
Fedora 19	`mls`	Yes	`allowed`	Yes	No

Cuando un sistema utiliza UBAC ciertos tipos son protegidos por restricciones adicionales. Esto asegurará que un usuario no puede acceder a archivos de otros usuarios. En esencia esta echo para aislar unos usuarios de otros.

La característica de la tabla MCS indica que se soportan *Mútiples Categorias de Seguridad*, en lugar de una única sensibilidad.

Ejercicio 2.4: Completar la tabla anterior para la distribución de Linux que esté usando cada uno de vosotros.

Cuando ejecutamos `sestatus` también tenemos información de la versión de la política que estamos usando. Esta información también la podemos ver con la orden `seinfo`.

```
# seinfo
Statistics for policy file: /etc/selinux/targeted/policy/policy.27
Policy Version & Type: v.27 (binary, mls)
...
```

La Tabla que sigue resumen las principales características de la política según la versión:

Version	Linux kernel	Description
12		It is the "Old API" for SELinux, now deprecated
15	2.6.0	It is the "New API" for SELinux
16	2.6.5	It provides conditional policy extensions
17	2.6.6	It provides IPv6 support
18	2.6.8	It adds fine-grained netlink socket support
19	2.6.12	It provides enhanced multi-level security
20	2.6.14	It doesn't access vector table size optimizations", the version (20) improved the access vector table size (it is a performance optimization).
21	2.6.19	It provides object classes in range transitions
22	2.6.25	It provides policy capabilities (features)
23	2.6.26	It provides per-domain permissive mode
24	2.6.28	It provides explicit hierarchy (type bounds)
25	2.6.39	It provides filename based transition support
26	3.0	It provides role transition support for non-process classes
27	3.5	It supports flexible inheritance of user and role for newly created objects
28	3.5	It supports flexible inheritance of type for newly created objects

Al escribir una política podríamos indicar que no queremos registrar alguna denegación dado que es evidente y si escritura podría desbordar los logs. Una declaración de este tipo se denomina *dontaudit*. Con la orden `seinfo` podemos ver cuantas reglas de SELinux (tales como *allow* o *dontaudit*) esta realmente cargadas en el sistema.

```
# seinfo | grep -E '(dontaudit|allow)'
   Allow:            34631     Neverallow:            0
   Auditallow:           1     Dontaudit:          5414
   Type_member:          6     Role allow:            7
# semodule --disable_dontaudit --build
```

En la última orden utilizamos **`semodule`** para reconstruir la política actual sin las declaraciones dontaudit, que es muy útil si una aplicación no esta funcionando adecuadamente y creemos que la razón es SELinux, pero no vemos ninguna denegación.

En Fedora y RedHat existen dos herramientas que pueden ayudarnos con las denegaciones: **`setroubleshoot`** y **`sealert`**. En el resto de distribuciones, podemos encontrar la utilidad **`audit2why`** (que es una especie de `audit2allow -w`) que nos da cierta realimentación de las denegaciones. Por ejemplo;

```
# ausearch -m avc -ts today | audit2why
type=AVC msg=audit(1371204434.608:475): avc:  denied { getattr }
for  pid=1376 comm="httpd" path="/var/www/html/infocenter" dev="dm-1"
ino=1183070 scontext=system_u:system_r:httpd_t:s0 tcontext=unconfined_u:o
bject_r:user_home_t:s0 tclass=dir

Was caused by:
The boolean httpd_read_user_content was set incorrectly.
Description:
Determine whether httpd can read generic user home content files.

Allow access by executing:
# setsebool -P httpd_read_user_content 1
```

La utilidad **audit2why** aquí no considera que el contexto de la ubicación objetivo este equivocado, y sugiere que se habiliten los permisos en el servidor web para leer contenidos de usuario.

Del examen del ejemplo anterior, podemos ver de la denegación que su etiqueta es **user_home_t**, es decir, para manipular archivos del directorio home del usuario, no para archivos de sistema dentro de /var. Una forma de ver que el contexto del recurso objetivo es correcto es comprobarlo con `matchpathcon` que nos devuelve el contexto que debería estar de acuerdo con la política de SELinux.

```
# matchpathcon /var/www/html/infocenter
/var/www/html/infocenter        system_u:object_r:httpd_sys_content_t:s0
```

Realizando esto sobre denegaciones relacionadas con archivos y directorios obtendremos ayuda para encotrar la solución de forma rápida. Además, muchos dominios tienen páginas de manual específicas que informan sobre los tipos que utilizan comúnmente en el dominio, así la forma de tratar con el dominio en más detalle:

```
# man ftpd_selinux
```

9.- Escribiendo un nuevo módulo de política

Probablemente no editaremos la política base salvo que seamos mantenedores de una distribución Linux, pero posiblemente deseemos que la política se comporte de forma diferente. En este caso, quizás debamos reconstruir la política base sin la parte responsable de cierto comportamiento y compilarla como un módulo.

Si necesitamos reconstruir completamente un nuevo módulo, también haremos uso de la herramienta `audit2allow`. En el ejemplo que sigue, creamos un módulo nuevo de política.

```
root@SEStrict ~/userkmsg # audit2allow -laRm userkmsg -o userkmsg.te
root@SEStrict ~/userkmsg # bzcat /usr/share/doc/selinux-base-policy
-20081210/ Makefile.example.bz2 > Makefile
```

```
root@SEStrict ~/userkmsg # ls
Makefile userkmsg.te
root@SEStrict ~/userkmsg # #check userkmsg.te if it really does what
you want
root@SEStrict ~/userkmsg # make
Compiling strict userkmsg module
/usr/bin/checkmodule: loading policy configuration from tmp/userkmsg.
tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 8) to tmp
/userkmsg.mod
Creating strict userkmsg.pp policy package
rm tmp/userkmsg.mod.fc tmp/userkmsg.mod
root@SEStrict ~/userkmsg # ls
Makefile tmp userkmsg.fc userkmsg.if userkmsg.pp userkmsg.te
root@SEStrict ~/userkmsg # semodule -i userkmsg.pp
root@SEStrict ~/userkmsg # semodule -l|grep userkmsg
userkmsg 1.0
root@SEStrict ~/userkmsg #
```

Después de que la herramienta genere el archivo .te, debemos asegurarnos manualmente si esta hace lo que deseamos.

10.- Herramientas:

En Fedora, para utilizar algunas de las órdenes comentadas debemos asegurarnos tener instalados los paquetes:

```
policycoreutils
policycoreutils-gui
selinux-policy
selinux-policy-policy
setroubleshoot-server
setools, setools-gui, and setools-console
libselinux-utils
mcstrans
```

Ver https://docs-old.fedoraproject.org/en-US/Fedora/12/html/Security-Enhanced_Linux/chap-Security-Enhanced_Linux-Working_with_SELinux.html#sect-Security-Enhanced_Linux-Working_with_SELinux-SELinux_Packages.