

SEGURIDAD EN SISTEMAS OPERATIVOS
4º Curso - Grado Ingeniería Informática
Defensa de Prácticas (20/12/2018)

Apellidos y nombre: García Carrera, Víctor

DNI: 51473495R

Cuestiones a responder sobre las prácticas de la Asignatura:

- 1) [40%] **En relación con AppArmor**, Sesión 3 de la Práctica 1, en el Ejercicio 4.2 donde se ha definido un perfil para una aplicación, explicar que acciones permite dicho perfil dicho perfil.
- 2) [40%] Sobre las protecciones frente a explotaciones del formato ELF, Ejercicio 1 de la Sesión 1 de la Práctica 2, explicar cómo se ha obtenido la información relativa a los todos los posibles mecanismos de protección de dicho formato de ejecutable para la distribución en la que ha realizado la práctica.
- 3) [20%] Sobre el análisis forense de discos duros, Ejercicio 3 de la Sesión 1 de la Práctica 3, indicar cuál ha sido el procedimiento y acciones seguidas para encontrar las evidencias necesarias para responder a las cuestiones planteadas.

Respuesta 1)

La aplicación elegida para desarrollar un perfil de AppArmor es *gedit*. A pesar de ser “un simple editor de texto”, vamos a ver que AppArmor permite desarrollar un perfil con multitud de configuraciones posibles. Los perfiles de AppArmor se localizan en el directorio */etc/apparmor.d/* en forma de simples ficheros de texto. Nuestro perfil se almacenará en esta ruta en forma de un fichero de texto de nombre *usr.bin.gedit*. A través del comando *sudo aa-genprof /usr/bin/gedit* entramos en el creador del perfil, que establece que hagamos un uso normal de la aplicación en otra ventana aparte mientras escanea las posibles acciones de seguridad que puede llevar a cabo. Leerá las entradas del log del sistema *syslog* y presentará diversas configuraciones posibles para la aplicación.

```
victorevictor@portatil: /etc/apparmor.d $ sudo aa-genprof /usr/bin/gedit
Writing updated profile for /usr/bin/gedit.
Setting /usr/bin/gedit to complain mode.

Before you begin, you may wish to check if a profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in order to scan the system logs for AppArmor events.
For each AppArmor event, you will be given the opportunity to choose whether the access should be
allowed or denied.

Profiling: /usr/bin/gedit
[[S]can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:
```

Tras finalizar este escaneo, presenta las diversas configuraciones de seguridad que se pueden llevar a cabo con distintas opciones (Allow, Deny, Ignore). Las acciones que nos permite llevar a cabo son muchas. En primer lugar, permite configurar los permisos del socket de tipo raw de la familia netlink que utiliza la aplicación, si permitimos o no su utilización. Cabe mencionar que en mi caso probé a denegar el utilizar ese socket netlink raw y la aplicación presentaba problemas para abrirse, en concreto mostraba el mensaje de la siguiente imagen:

```
Profile: /usr/bin/gedit
Network Family: netlink
Socket Type: raw

[1 - #include <abstractions/nameservice>]
[2 - network netlink raw,
{Allow / (Deny) / (Ignore / Audit / Abort / Finish
Adding deny network netlink raw, to profile.
```

```
victor@victor-portatil ~ $ gedit
Failed to connect to Mir: Failed to connect to server socket: No such file or directory
Unable to init server: Could not connect: Connection refused
```

Además, nos permite configurar muchas de las librerías que utiliza la aplicación, como por ejemplo los diccionarios utilizados para la corrección del lenguaje y sugerencia de palabras. En mi caso particular en el que mi sistema tiene como idioma principal el inglés, me permite configurar la utilización o no de las librerías que contienen los diccionarios de inglés británico(GB) y el inglés americano (US) tal y como muestra esta imagen:

```
1 - /home/victor/.config/enchant/en_GB.exc
[2 - /home/*/.config/enchant/en_GB.exc]
{(A)llow / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)inish / (M)ore

Profile: /usr/bin/gedit
Path: /home/victor/.config/enchant/en_US.dic
Mode: rw
Severity: 6

1 - /home/victor/.config/enchant/en_US.dic
[2 - /home/*/.config/enchant/en_US.dic]
{(A)llow / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)inish / (M)ore

Profile: /usr/bin/gedit
Path: /home/victor/.config/enchant/en_US.exc
Mode: rw
Severity: 6

1 - /home/victor/.config/enchant/en_US.exc
[2 - /home/*/.config/enchant/en_US.exc]
{(A)llow / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)inish / (M)ore
```

Finalmente, otra acción destacada que permitía llevar a cabo era configurar los permisos de lectura y escritura del archivo *pruebagedit* creado para probar la aplicación mientras AppArmor escaneaba los logs.

```
Profile: /usr/bin/gedit
Path: /home/victor/pruebagedit
Mode: rw
Severity: 6

1 - /home/victor/pruebagedit
[2 - /home/*pruebagedit]
{(A)llow / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)inish / (M)ore
```

Tras crear el perfil, ya se encuentra activado en modo enforce como comprobamos con una nueva llamada a *aa-status*:

```
victor@victor-portatil /etc/apparmor.d $ sudo aa-status
[sudo] password for victor:
apparmor module is loaded.
53 profiles are loaded.
16 profiles are in enforce mode.
/usr/bin/dhclient
/usr/bin/gedit
/usr/lib/NetworkManager/nm-dhcp-client.action
/usr/lib/NetworkManager/nm-dhcp-helper
```

Respuesta 2)

Para conocer los mecanismos de protección de un binario ELF presentes en nuestro sistema, comenzamos recopilando diversa información acerca de nuestro sistema como su distribución, arquitectura y el compilador utilizado.

Distribución: Linux Mint 18.1 Serena (comando `cat /etc/issue`)

Arquitectura: 64-bit, x86_64 (podemos verlo en la información de la CPU que podemos visualizar mediante el comando `lscpu`)

Kernel Release: 4.4.0-53-generic (`uname -r`)

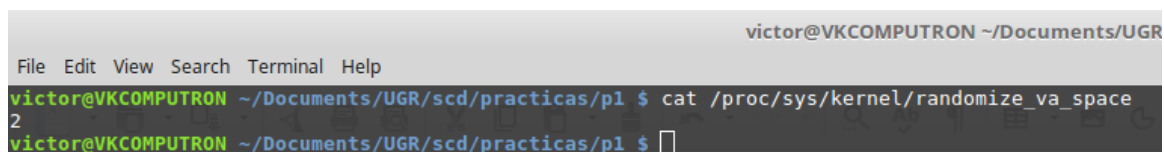
Versión del compilador gcc (comando `cat /proc/version`): 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4)

Antes de continuar, toda la información que se muestra a continuación ha sido obtenida tanto del guión de la práctica, como de los enlaces que había en la misma relativos a los diversos mecanismos de protección y una búsqueda exhaustiva de diversas fuentes acerca de cada mecanismo para obtener las versiones del compilador a partir de las cuales están presentes y demás información necesaria.

Una vez conocemos las versiones de nuestro sistema, procedemos a analizar los diversos mecanismos existentes para protegerse frente a ataques maliciosos de *buffer overflow* o desbordamiento de buffer.

El primer mecanismo observado es el *ALSR* (*Aleatorización de la disposición del espacio de direcciones*). Este mecanismo pertenece al kernel, estando presente en el kernel de Linux a partir de la versión 3.14, por lo que nuestro sistema dispone de ella. El kernel junto con el cargador de programas aleatoriza la localización dentro del espacio de direcciones de las áreas de datos tales como la base del ejecutable, las posiciones de la pila, el heap y las librerías con el objetivo de dificultar la predicción de un atacante acerca de las direcciones de memoria utilizadas por el programa para evitar que pueda alterarlas.

Este mecanismo necesita de la opción de compilación *-fPIE* para activar el mecanismo de *ejecutables independientes de la posición (PIE)*, el cual también presente en nuestro compilador. Podemos controlar ASLR mediante el valor de `/proc/sys/kernel/randomize_va_space`. De forma predeterminada, su valor es 2, indicando que realiza lo descrito previamente junto con la aleatorización de los segmentos de datos.



```
victor@VKCOMPUTRON ~/Documents/UGR
File Edit View Search Terminal Help
victor@VKCOMPUTRON ~/Documents/UGR/scd/practicas/p1 $ cat /proc/sys/kernel/randomize_va_space
2
victor@VKCOMPUTRON ~/Documents/UGR/scd/practicas/p1 $
```

El siguiente mecanismo a escrutar es el de *protección de rotura de pila*, que va ligado al de *fuerza fortificada* o *fortify source*. Se trata de una protección en tiempo de compilación que activa una serie de protecciones en la *glibc* para detectar *buffer overflow* en diversas funciones que realizan operaciones con memoria y strings. No detecta todos los tipos de *buffer overflow* pero si que proporciona un nivel mayor de seguridad en aquellas funciones que son más propensas a causar fugas de memoria. Se activa con la directiva de compilación `-D_FORTIFY_SOURCE=x`, donde *x* puede valer 1, en cuyo caso es necesario también utilizar la opción de optimización del compilador de nivel 1 (`gcc -O1`) u otros niveles en adelante, y no afecta al comportamiento del programa, o *x* puede valer 2, donde se realizan más comprobaciones pero puede modificar el comportamiento esperado del programa. La utilización de este mecanismo está disponible en versiones de GCC posteriores a la 4.0, por lo que disponemos del mismo.

El siguiente mecanismo es bastante conocido, el de *protección de pila* (*Stack Smashing Protection*). Está disponible en versiones de GCC posteriores a la 3.2-7 (disponemos de ella). Las posibles opciones de compilación para implementarla son las siguientes: `-fno-stack-protector` deshabilita este mecanismo, `-fstack-protector` es la opción más utilizada y activa la técnica de canario de pila o *Stack Canary* sobre un conjunto de funciones potenciales de generar un *buffer overflow*, `-fstack-protector-all` cumple la misma función que la anterior pero sobre todas las funciones, y finalmente `-fstack-protector-strong`, que incluye funciones adicionales a proteger.

El último mecanismo es el de *protección de pila no ejecutable*. Creamos un ejecutable ELF llamado *prodcons_exe* y con la instrucción `readelf -l prodcons_exe` podemos observar si la pila es o no ejecutable. La siguiente salida muestra como, en la línea referente a GNU_STACK, los permisos son solo de lectura y escritura, por lo que la pila no es ejecutable. Podemos cambiar los permisos de la pila con el programa `exestack -s [binario]` o con la opción de compilación `-z execstack`.

```
victor@VKCOMPUTRON ~/Documents/UGR/scd/practicas/p1 $ readelf -l prodcons_exe
Elf file type is EXEC (Executable file)
Entry point 0x4018e0
There are 9 program headers, starting at offset 64

Program Headers:
Type           Offset             VirtAddr           PhysAddr
FileSiz        MemSiz              Flags             Align
PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
PHDR           0x00000000000001f8 0x00000000000001f8 R E 0
INTERP         0x0000000000000238 0x0000000000000238 R E 0
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
LOAD           0x000000000000b7b8 0x000000000000b7b8 R E 200000
LOAD           0x000000000000bdc0 0x000000000000bdc0 0x000000000000bdc0
DYNAMIC        0x0000000000003e8 0x000000000000928 RW 200000
NOTE           0x000000000000bdf8 0x000000000000bdf8 0x000000000000bdf8
NOTE           0x000000000000200 0x000000000000200 RW 8
NOTE           0x000000000000254 0x000000000000254 0x000000000000254
NOTE           0x000000000000e44 0x000000000000e44 R 4
GNU_EH_FRAME   0x0000000000007e58 0x0000000000007e58 0x0000000000007e58
GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
GNU_RELRO      0x0000000000000000 0x0000000000000000 RW 10
GNU_RELRO      0x000000000000bdc0 0x000000000000bdc0 0x000000000000bdc0
                0x000000000000230 0x000000000000230 R 1

Section to Segment mapping:
Segment Sections...
```

Respuesta 3)

En este ejercicio utilizamos la herramienta Autopsy para analizar una imagen forense. En primer lugar queremos buscar en la imagen forense del pendrive utilizado en los ejercicios anteriores de esa práctica información relativa al fichero borrado que conteniza la amenaza. En segundo lugar, utilizamos dos imágenes forenses descargadas de un caso real en el que se produjo una filtración de documentos corporativos desde un ordenador de un alto ejecutivo de la empresa M57 y se publicaron en un foro del sitio web de la competencia.

Para la búsqueda de estas evidencias, una vez creamos el caso en Autopsy y cargamos las imágenes forenses, la herramienta cuenta con un buscador de palabras (keyword search) que resulta nuestra piedra angular para encontrar las evidencias. Para el primer caso del pendrive, buscamos por la palabra clave *amenaza* (esta palabra estaba presente en el fichero de texto borrado). Para el caso M57, buscamos por nombres de empleados de la empresa (Alison, Jean, Bob, Gina, Harris, Indy...). También buscamos los correos de Alison y Jean: alison@m57.biz ; password: "ab=8989, jean@m57.biz ; password: gick*1212. Esto nos permite ver los usuarios implicados en la filtración.

Además, contamos con apartados en Autopsy para analizar los ficheros, ver su tipo y algunos metadatos, información que utilizamos para ver otras evidencias como cuándo se creó la hoja de cálculo.