Práctica 5 - Seguridad - Parte 1

A - Introducción 1

1. Defina política y mecanismo.

Una *política* define *qué* se quiere hacer, los objetivos del sistema. Un *mecanismo* define *cómo* se implementa esa política. Separarlos da flexibilidad al sistema.

2. Defina objeto, dominio y right.

- *Objeto:* recurso de hardware o software (ej: archivo, impresora)
- *Dominio:* conjunto de pares (objeto, derecho)
- *Right:* derecho para realizar operaciones sobre un objeto (ej: read, write).

3. Defina POLA (Principle of least authority).

Principio que indica que los procesos deben acceder *solo* a los objetos y derechos necesarios para completar su tarea.

4. ¿Qué valores definen el dominio en UNIX?

El *UID* y el *GID*. Dos procesos con el mismo (UID, GID) pertenecen al mismo dominio y acceden 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 un mecanismo que randomiza la ubicación de áreas de memoria como stack, heap, text y bibliotecas. Linux provee ASLR para procesos de usuario (*ASLR*) y para el kernel (*KASLR*).

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

Editando el archivo `/proc/sys/kernel/randomize_va_space`:

- `0`: deshabilitado
- `1` o `2`: activado con distintos niveles de randomización.

B - Ejercicio introductorio: Buffer Overflow simple

1. Compilar usando el makefile provisto el ejemplo 00-stack-overflow.c provisto en el repositorio de la cátedra.

Acá primero pullee y luego:
make 00-stack-overflow

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

root@so:/home/so/codigo-para-practicas/practica5# ./00-stack-overflow access pointer: 0x7ffc84c3158f, password pointer: 0x7ffc84c31570, distance: 31 Write password: s

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

Con big secret funciona.

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.

```
root@so:/home/so/codigo-para-practicas/practica5# python3 -c 'print("A"*31 + "\x01")' | ./00-stack-overflow
access pointer: 0x7fff9ef25f1f, password pointer: 0x7fff9ef25f00, distance: 31
Write password: Now you know the secret
```

Comando para generar una cadena de 31 bytes:
`python3 -c 'print("A"*31 + "\x01")' | ./00-stack-overflow`

5. Después de realizar la explotación, reflexiona sobre las siguientes preguntas:

a. ¿Por qué el uso de gets() es peligroso?

Porque no verifica la longitud del input. Puede sobrescribir variables o direcciones de memoria importantes

b. ¿Cómo se puede prevenir este tipo de vulnerabilidad?

Evitar `gets()`. Usar funciones como `fgets()` o `memcpy()` con verificación de tamaño

c. ¿Qué medidas de seguridad ofrecen los compiladores modernos para evitar estas vulnerabilidades?

- **ASLR**: randomiza direcciones, haciendo difícil predecir dónde escribir.
- **NX bit**: marca como no ejecutables ciertas regiones de memoria.
- **KASLR**: aplica ASLR también al kernel.
- *Stack Canaries (protector de pila):* Inserta valores especiales (canarios) antes de la dirección de retorno en la pila. Si un overflow sobrescribe el canario, el programa detecta la corrupción y aborta.
- *Advertencias y desuso de funciones inseguras:* Los compiladores emiten advertencias o errores cuando se usan funciones peligrosas como gets().

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.

make y eso

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

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

`cat /proc/sys/kernel/randomize_va_space` y eso debe retornar 2.

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

Por si solo no cambio nada tuve que modificar el makefile:

```
SOURCES = $(wildcard *.c)
ASSEMBLY_SOURCES = $(SOURCES:.c=.s)
PREPROC_FILES = $(SOURCES:.c=.i)
OBJECTS = $(SOURCES:.c=.o)
EXECUTABLES = $(SOURCES:.c=)
```

```
# -fno-stack-protector: Disables stack protection, which helps prevent buffer
overflows.
# -z execstack: Allows execution of code on the stack, which can lead to code injection
attacks.
# -no-pie: Disables position-independent executables, which can help prevent code
injection attacks.
# -fcf-protection=none: Disables control flow protection, which helps prevent code
injection attacks.
# -fno-asynchronous-unwind-tables: Disables unwind tables, which can help prevent stack
corruption attacks.
# -O0: Disables optimizations, which can make it easier to analyze the code and find
vulnerabilities.
INSECURE_FLAGS = \
    -fno-stack-protector\
    -z execstack\
    -fcf-protection=none\
    -O0 \
    -fPIE


CFLAGS = -save-temps -g $(INSECURE_FLAGS)
LDFLAGS = -pie

all: $(EXECUTABLES)
    $(CC) $(CFLAGS) $(LDFLAGS) $@.c -o $@

clean:
    rm -f $(OBJECTS) $(EXECUTABLES) $(ASSEMBLY_SOURCES) $(PREPROC_FILES)
```

Lo compile devuelta y anduvo.

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().

```
echo "0" > /proc/sys/kernel/randomize_va_space
```

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.

25 bytes
Según el layout de stack que diste:

- 00–15 → `password` (16 bytes)
- 16–23 → old RBP (8 bytes)
- 24–31 → dirección de retorno (8 bytes)

Para alcanzar la dirección de retorno y sobreescribirla

- Necesitás **16 + 8 = 24 bytes** de relleno.
- El byte 25 y en adelante pisan la dirección de retorno.

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

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

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

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

```
root@so:/home/so/codigo-para-practicas/practica5# python3 payload_pointer.py --padding 24 --pointer "0x5555555551c9" --pointer-size 8 --endianness little --program "./01-stack-overflow-ret"
```

Yo lo termine haciendo todo junto con ese comando.
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**:
  - Cuando un programa con **SETUID (chmod u+s)** es ejecutado por cualquier usuario, el proceso se ejecuta con los **privilegios del propietario** (root en este caso).
  - Esto permite a usuarios normales realizar acciones privilegiadas temporalmente (ej: cambiar contraseñas con `passwd`).
  - **En tu caso**: El exploit funcionó porque `01-stack-overflow-ret` tenía **SETUID root**, permitiéndote obtener un shell con `uid=0(root)`.
- **Si el propietario es nobody**:

```
- El proceso se ejecutaría con los permisos de `nobody` (usuario sin privilegios).
- No tendría ningún beneficio de seguridad, ya que `nobody` no tiene acceso
especial.
```

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?

1. **Funcionamiento del payload**:
   - Sobrescribimos la dirección de retorno con `0x5555555551c9` (`privileged_fn`).
   - Al retornar, el flujo del programa saltó a `privileged_fn`, que ejecutaba una shell (`/bin/sh`).
   - Como el programa tenía **SETUID root**, la shell heredó permisos de root (`uid=0`).
2. **Comandos ejecutados**:
   - `ls`: Mostró archivos en el directorio actual (confirmando acceso al filesystem).
   - `id`: Verificó que el UID era `0` (root).
   - `pwd`: Confirmó el directorio de trabajo.

- **SETUID root** es poderoso pero peligroso: Si un programa con este permiso es vulnerable (como en este caso), permite escalar privilegios.
- **Little-endian** afecta cómo se escriben direcciones en memoria: Siempre debes invertir los bytes al generar exploits para x86_64.
- **El exploit fue exitoso**: Obtuviste una shell con privilegios de root gracias al desbordamiento de buffer + SETUID.
  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?
  ASLR ayuda porque al randomizar las direcciones que toma el programa es mucho más dificil que uno haga una payload que rompa el programa.
  d. ¿Cómo podría evitar este tipo de ataques en un módulo del kernel de Linux? ¿Qué mecanismo debería estar habilitado?
  1. SMAP (Supervisor Mode Access Prevention)
- **Bloquea el acceso del kernel a memoria de usuario**.
- Evita que exploits en el kernel usen datos manipulados desde user-space.

2. SMEP (Supervisor Mode Execution Prevention)

- **Impide que el kernel ejecute código en páginas de usuario**.
- Mitiga ataques que inyectan shellcode en memoria de usuario.

3. KASLR (Kernel ASLR)

- **Aleatoriza la ubicación del código del kernel**.
- Hace impredecibles direcciones de funciones críticas.

4. Stack Canaries

- **Valores aleatorios ("canarios") entre variables y la dirección de retorno**.
- Si un desbordamiento modifica el canario, el kernel detecta la corrupción y aborta.

5. `CONFIG_STRICT_DEVMEM`

- **Restringe acceso a** `/dev/mem`, evitando que usuarios lean/escriban memoria del kernel directamente.

6. Deshabilitar módulos peligrosos

- Ejemplo: `sysctl kernel.modules_disabled=1` (solo para entornos ultra-restringidos).

D - Ejercicio SystemD

1. Investigue los comandos:

a. systemctl enable

Propósito: Activar el inicio automático de un servicio al arrancar el sistema.

b. systemctl disable

Propósito: Desactivar el inicio automático de un servicio.

c. systemctl daemon-reload

Propósito: Recargar la configuración de SystemD después de modificar archivos de unidad (`.service`, `.timer`, etc.).

d. systemctl start

Propósito: Iniciar un servicio inmediatamente.

e. systemctl stop

Propósito: Detener un servicio en ejecución.

f. systemctl status

Propósito: Ver el estado de un servicio (activo, inactivo, fallido).

g. systemd-cgls

Propósito: Mostrar la jerarquía de control de grupos (cgroups) de SystemD.

h. `journalctl -u [unit]`

Propósito: Mostrar logs específicos de una unidad (servicio).

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

a. IPAddressDeny e IPAddressAllow

IPAddressDeny: Bloquea conexiones desde direcciones IP específicas.
IPAddressAllow: Permite conexiones solo desde direcciones IP específicas (anula `IPAddressDeny`).

b. User y Group

User: Ejecuta el servicio con un usuario específico (no root).
Group: Ejecuta el servicio con un grupo específico.

c. ProtectHome

Protege directorios como `/home`, `/root`, y `/run/user` (solo lectura o inaccesibles).
Opciones:

- `true` (solo lectura).
- `read-only` (solo lectura).
- `tmpfs` (monta como tmpfs vacío).

d. PrivateTmp

Crea un directorio `/tmp` privado para el servicio, aislado del sistema.

e. ProtectProc

Restringe acceso a `/proc` para evitar fugas de información.
Opciones:

- `noaccess` (oculta procesos de otros usuarios).
- `invisible` (oculta todos los procesos excepto los del servicio).

- `ptraceable` (permite debugging).
  Ejemplo:

f. MemoryAccounting, MemoryHigh y MemoryMax

MemoryAccounting: Habilita el monitoreo de uso de memoria.
MemoryHigh: Límite "blando" de memoria (el servicio puede superarlo, pero el kernel intentará reducir su consumo).
MemoryMax: Límite "duro" de memoria (el servicio será terminado si lo supera).

Nota: Todas estas opciones se colocan en la sección `[Service]` del archivo `.service`.

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 ii. systemctl restart insecure_service.service

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.

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

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# systemctl status
insecure_service.service
● insecure_service.service - Insecure service
     Loaded: loaded (/etc/systemd/system/insecure_service.service; enabled; preset:
enabled)
     Active: active (running) since Mon 2025-06-02 12:48:28 -03; 17min ago
   Main PID: 2755 (insecure_servic)
      Tasks: 5 (limit: 2306)
     Memory: 2.4M
        CPU: 10ms
     CGroup: /system.slice/insecure_service.service
             └─2755 /opt/sistemasoperativos/insecure_service

lines 1-10
```

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

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# ps aux | grep
insecure_service
root        2755  0.0  0.4 1232112 8244 ?        Ssl  12:48   0:00
/opt/sistemasoperativos/insecure_service
root        3079  0.0  0.1   6484  2092 pts/0    S+   13:06   0:00 grep
insecure_service
```

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

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# curl
http://localhost:8080

<!DOCTYPE html>
<html lang="en" data-theme="light">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Insecure Service</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bulma@1.0.4/css/bulma.min.css">
</head>
<body>


  <header class="has-background-primary p-2">
    <h1 class="title has-text-white">Home</h1>
  </header>


  <div class="columns m-0" class="has-background-primary" style="min-height: calc(100vh
- 128px);">


    <aside class="column menu has-background-primary is-2 p-2 is-flex is-align-content-
center">
      <ul class="menu-list">

        <li><a class="has-background-primary has-text-white" href="/">Home</a></li>

        <li><a class="has-background-primary has-text-white"
href="/resources/">Recursos</a></li>
```

```
        <li><a class="has-background-primary has-text-white" href="/mem/">Memoria</a>
</li>

      </ul>
    </aside>


    <main class="column p-4 has-background-white">

    </main>
        </div>
  </body>
</html>
```

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.

Estos es jijo porque no estarían estos grupos

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# ps aux | grep
insecure_service
root        4134  0.0  0.1   6484  2096 pts/0    S+   13:16   0:00 grep
insecure_service
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# systemctl status
insecure_service.service -l
× insecure_service.service - Insecure service
     Loaded: loaded (/etc/systemd/system/insecure_service.service; enabled; preset:
enabled)
     Active: failed (Result: exit-code) since Mon 2025-06-02 13:16:28 -03; 1min 16s ago
   Duration: 5ms
    Process: 4093 ExecStart=/opt/sistemasoperativos/insecure_service (code=exited,
status=217/USER)
   Main PID: 4093 (code=exited, status=217/USER)
        CPU: 621us

jun 02 13:16:28 so systemd[1]: Started insecure_service.service - Insecure service.
jun 02 13:16:28 so (_service)[4093]: insecure_service.service: Failed to determine user
credentials: No such process
jun 02 13:16:28 so (_service)[4093]: insecure_service.service: Failed at step USER
spawning /opt/sistemasoperativos/insecure_service: No such process
jun 02 13:16:28 so systemd[1]: insecure_service.service: Main process exited,
code=exited, status=217/USER
jun 02 13:16:28 so systemd[1]: insecure_service.service: Failed with result 'exit-
code'.
```

Y no puedo crear usuarios porque no esta los comandos. Así que se queda acá la práctica.

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).

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.

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

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

Todos los cambios juntos son:

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# cat
/etc/systemd/system/insecure_service.service
# SystemD unit to handle insecure_service service
[Unit]
Description=Insecure service
After=network.target

[Service]
Type=simple
Restart=Always
ExecStart=/opt/sistemasoperativos/insecure_service
# 5. Ejecutar como usuario y grupo no privilegiados
User=nobody
Group=nogroup

# 6. Limitar acceso solo a localhost (si el servicio lo permite)
# Esto depende del binario; esta línea puede ser ignorada si no funciona
# BindAddress=127.0.0.1

# 7a. No permitir acceso a /home y crear /tmp privado
ProtectHome=yes
PrivateTmp=yes
```

```
# 8. Limitar acceso a otros procesos
ProtectProc=invisible

# 9. Límite de memoria
MemoryMax=16M

[Install]
WantedBy=multi-user.target
```

```
root@so:/home/so/codigo-para-practicas/practica5/insecure_service# systemctl daemon-reexec
systemctl daemon-reload
systemctl restart insecure_service
```

```
root@so:/home/so# ps aux | grep insecure_service
nobody      1056  0.0  0.3 1232112 7340 ?        Ssl  10:50   0:00 /opt/sistemasoperativos/insecure_service
root        1170  0.0  0.1   6484  2076 pts/4    S+   10:50   0:00 grep insecure_service
```