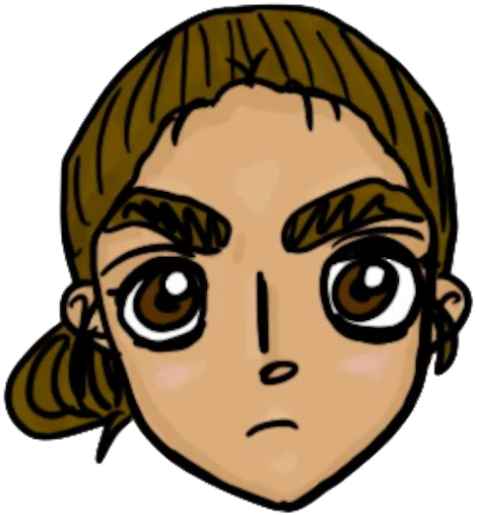
Introducción a las técnicas de exploiting en Linux

Álvaro Fernández
Lucas Tomé

tail -2 /etc/passwd

t_lucas:x:1001:1001::/home/t_lucas:/bin/fish



Hacker trabajando actualmente en Red Team.
Participo en CTFs y otras competiciones de hacking o de programación.

Interesado en la seguridad ofensiva, la privacidad, la electrónica y en destripar software.

f_alvaro:x:1002:1002::/home/f_alvaro:/bin/fish

Pequeño entusiasta del mundo de la informática y en concreto de la ciberseguridad, que gasta su tiempo haciendo CTFs y trabajando en un laboratorio de pentesting.

cd /sconf/introduccion/

cat README.md

Vulnerabilidades y sus protecciones

Vamos a ver:

* Stack overflow
* Format string
* Escritura arbitraria
* Heap overflow

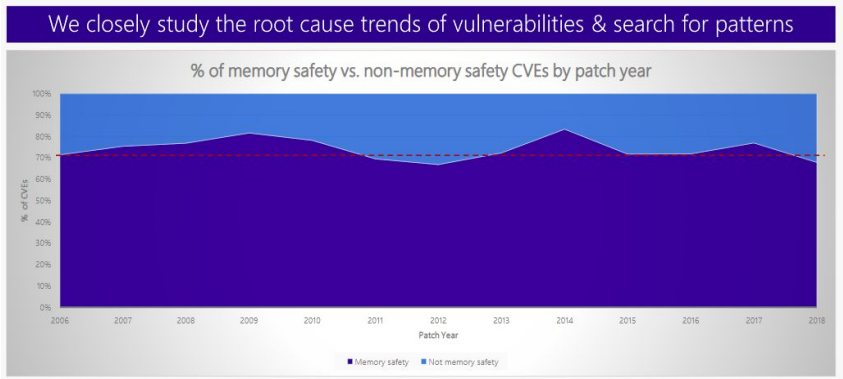
Con ejemplos para aprender practicando

Pero hay muchas más

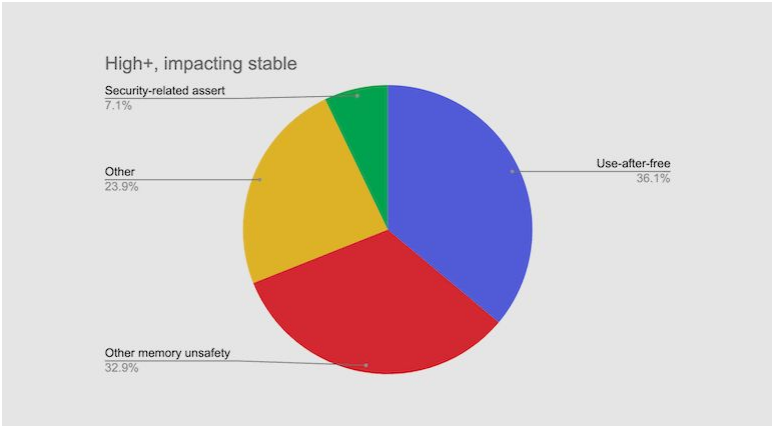
pip install -r requeriments.txt

- Conceptos básicos de programación
- Familiarizado con la terminal UNIX
- El resto se va a explicar desde 0

cat motivacion



Matt Miller (MSRC)



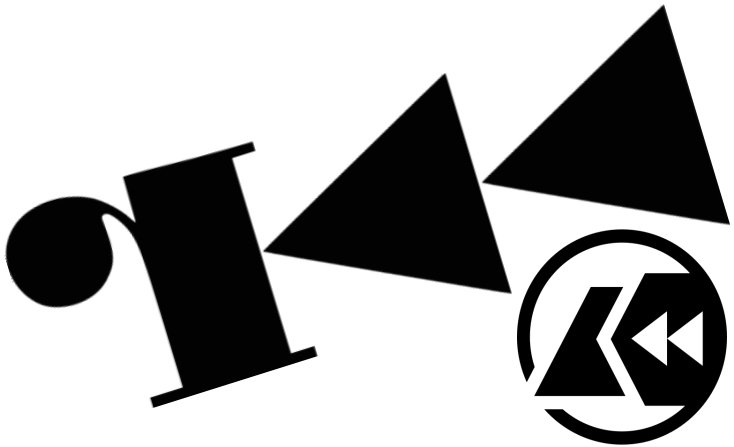
Google

curl -O http://sconf/exploiting.ova

- 3 GB debería estar descargada ya :D
- ArcoLinux XFCE con 8 GB de disco
- Usuario: exploit
- Contraseña: exploit
- 2GB RAM y 4 cores. Adaptar a cada ordenador antes de encender
- Contiene:
  - Herramientas necesarias para seguir el taller
  - Problemas planteados en el taller con sus soluciones

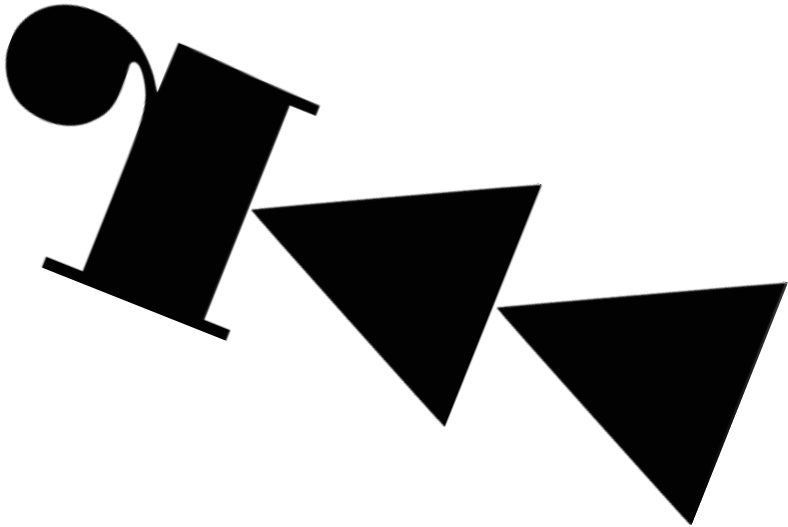
cd /sconf/tools

man desensabladores



¿Decompilador?

man debuggers



https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

man pwntools

PWNTOOLS

Pwntools es un framework para CTF y una biblioteca de desarrollo de exploits. Escrito en Python, está diseñado para la creación y el desarrollo rápidos de prototipos, y tiene la intención de hacer que la escritura de exploits sea lo más simple posible.

http://docs.pwntools.com/en/stable/

man pwntools

Clases y funciones que vamos a utilizar:

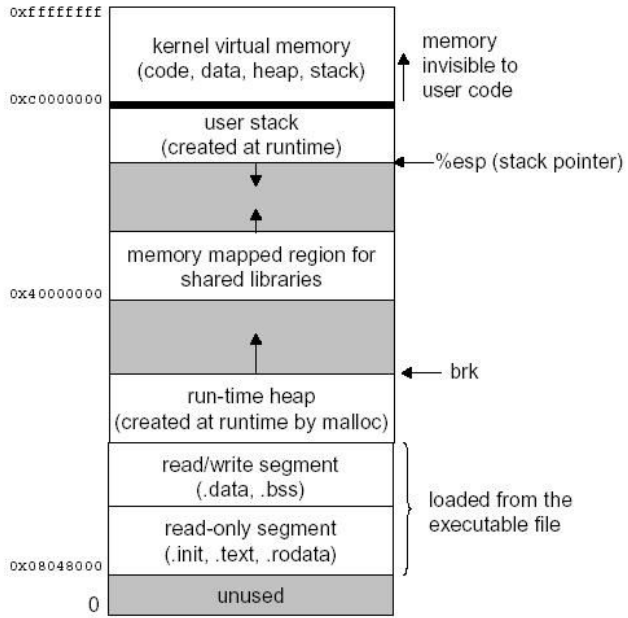
- process(*path_program*)
  - sendline(*line*): Escribir una línea en el stdin del programa
  - recvline(): Leer una línea en el stdout del programa
  - interactive(): Pasar a modo interactivo
- ELF(path_program)
  - sym/symbols: array con los símbolos de un fichero ELF
  - path: ruta del fichero ELF
  - search: buscar una cadena en el fichero ELF
- ROP(*path_program*)
  - find_gadget(*instructions*): devuelve un array con la dirección de memoria donde se encuentran las instrucciones.
- flat(*args*): codifica los argumentos en una cadena

cd /sconf/teoría

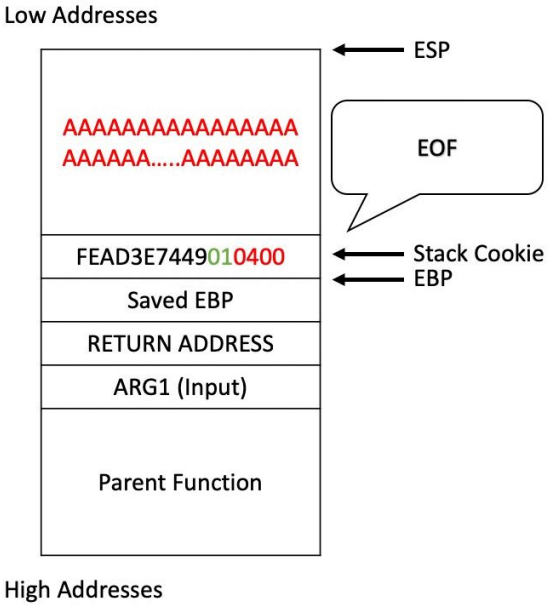
cat info/exe_struct

Algunas secciones son:

- .data
- .bss
- .text
- stack
- heap
- .got
- .plt



cat info/stack



Estructura de memoria FIFO utilizada en un programa para:

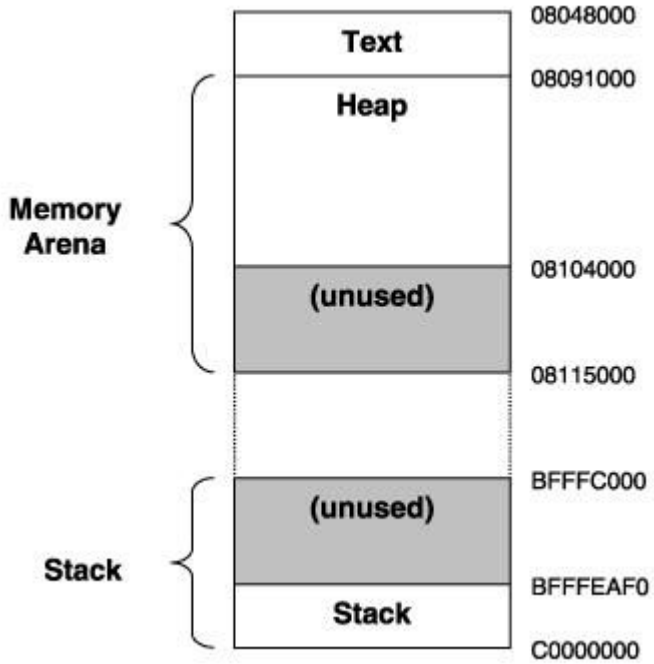
- Variables locales
- Stack frame
- (x86) parámetros de una función.

cat info/heap (en Linux)

Estructura de memoria utilizada en un programa para variables.

Se reserva de forma dinámica mediante las funciones:

- malloc
- calloc



cat info/x86 (aka Ensamblador)

Conditional jumps		
Instruction	**Description**	**Condition**
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JG	Jump if Greater	ZF = 0 and SF = OF
JGE	Jump if Greater of Equal	SF = OF
JL	Jump if Less	SF <> OF
JLE	Jump if Less or Equal	ZF = 1 or SF <> OF
JO	Jump if Overflow	OF = 1
JP	Jump if Parity	PF = 1
JPE	Jump if Parity Even	PF = 1
JS	Jump if Sign	SF = 1
JZ	Jump if Zero	ZF = 1

Loops	
LOOP	Decrements CX and jumps to label if CX <> 0
LOOPE LOOPZ	Decrements CX and jumps to label if CX <> 0 and ZF = 1
LOOPNE LOOPNZ	Decrements CX and jumps to label if CX <> 0 and ZF = 0
JCXZ	Jumps to label if CX = 0

Remember: Loops uses CX and ZF registers

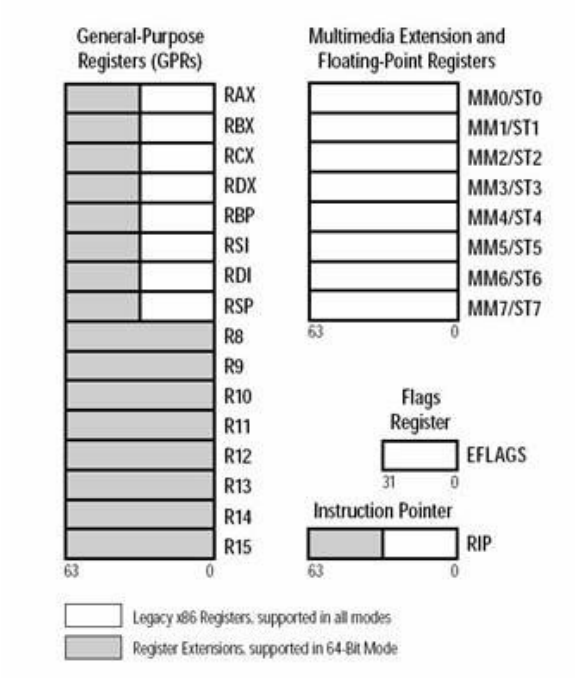
Instructions (cont)		
MUL / IMUL / DIV / IDIV	Multiplies or divides two numbers and stores it to AX (+DX for most significant bit)	MOV AL, 0001Ah MUL 00002h
JMP	Jumps to a label	JMP calc
CMP	Performs a comparison	CMP AL, 01234b
PUSH / PUSHA / PUSHF	Pushes a data to SS: [SP] / all registers / FLAGS	PUSH 10h
POP / POPA / POPF	Restores fatas form SS: [SP] / all registers (except SP) / FLAGS	POP AX

Instructions		
Instruction	**Usage**	**Example**
MOV	Assigns value to register	MOV AX, 15h
ADD	Adds value to register	ADD AX, BX
SUB	Substracts value to register	SUB AX, 1
AND	Executes binary AND operation	AND AL, 1101111b
OR	Executes binary OR operation	OR AL, BX
NOT	Executes binary NOT operation	NOT AL
XOR	Executes binary XOR operation	XOR AL, 01010101b

SHL/SAL /SHR/SAR	Shifts to the left or to the right. When arithmetic, the sign bit doesn't shift. Explused bit is stocked in CF	SHL AL, 1 / SAL AL, 2
ROL/RCL/ROR/RCR	Rotates to the left or to the right, using or not CF	ROL AL, 1 / RCR AL, 2
INC / DEC	Increments or decrements a value	INC AX / DEC BX
ADC / SBB	Does an addition or substraction of >16bits numbers, by storing restraint in CF	ADC AX, CX
STC / CLS / CMC	Sets CF to 1, 0 or inverts it	STC
STD / CLD	Sets DF to 1 or 0	STD
STI / CLI	Sets IF to 1 or 0	CLI

https://cheatography.com//mika56/cheat-sheets/asm-8086/pdf/

cat info/registros64



Registros importantes:

- EFLAGS: bits con el estado
- RIP: puntero a la instrucción
- RAX: return de una función.

cat info/C_ABI (en x86_64)

Una ABI es una asignación del modelo de ejecución del lenguaje a una combinación particular de máquina / sistema operativo / compilador.

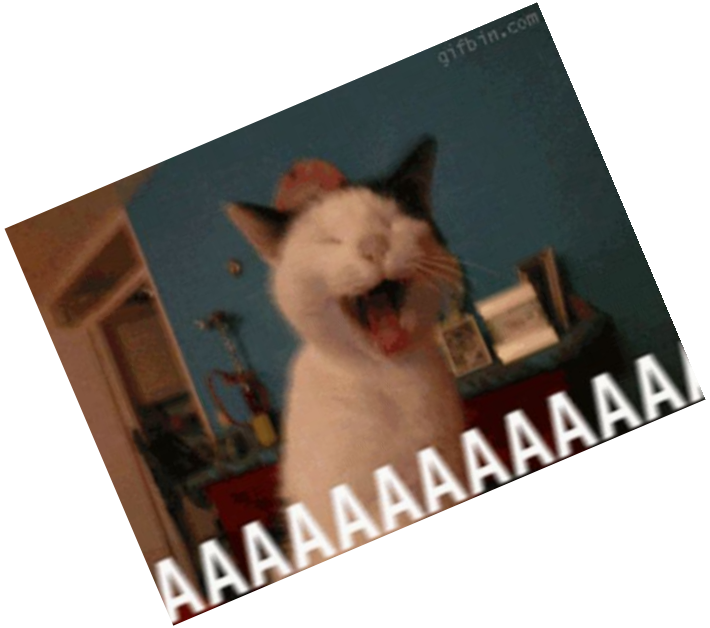
Algunas especificaciones del ABI en un linux de 64bits son:

- La secuencia de llamada a una función tiene las siguientes características:
  - Para pasar parámetros a la subrutina, colocamos hasta seis de ellos en registros (en orden: rdi, rsi, rdx, rcx, r8, r9). Si hay más de seis parámetros en la subrutina se inserta en la pila.
  - La instrucción call guarda la dirección de retorno y la dirección base del marco actual de la pila en el stack, y se ramifica al código de subrutina.
  - Cuando se realiza la función, el valor de retorno de la función debe colocarse en RAX
- Alineado a 64 bits
- Las llamadas al sistema operativo se realizan mediante interrupciones

./hello_world

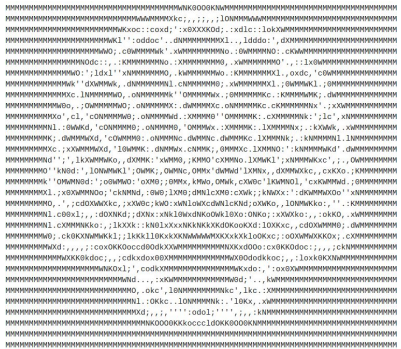
cd /sconf/practica

wget https://stackoverflow.com/



Escribir en una variable información que ocupa más espacio que el reservado por esa variable y por tanto, sobreescribir otras zonas de memoria

cat info/shellcode



Porción de código en binario que se inserta en un buffer del ejecutable. Esto permite que si se redirige la ejecución del código al buffer ejecutamos el código deseado.

cat info/NOP

0x90

Instrucción **N**o **OP**eration

./exploit1.py

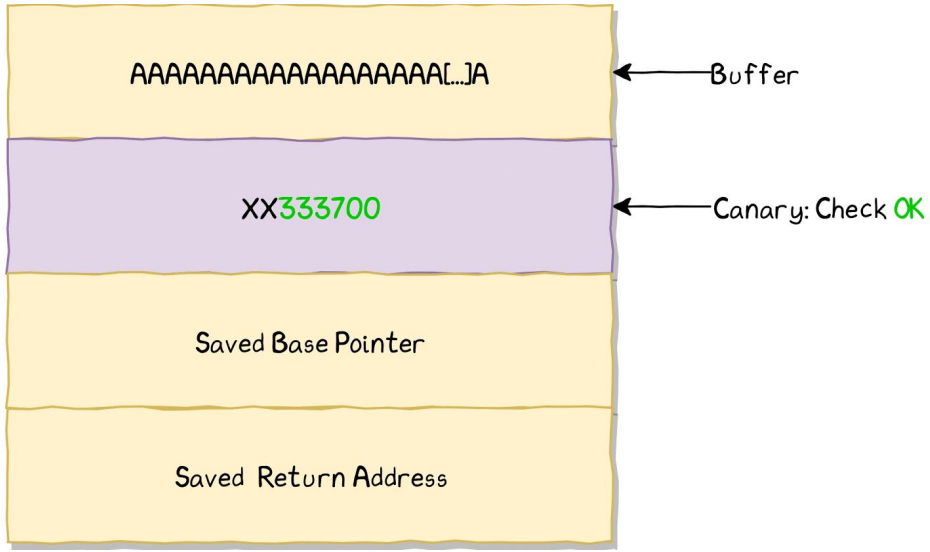
Fases:

1. Desactivar ASLR
2. Encontrar la vulnerabilidad.
3. Calcular la longitud hasta el RIP
4. Buscar la dirección de inicio del buffer
5. Buscar el shellcode (http://shell-storm.org/shellcode/)
6. Preparar el payload
7. Proceder a su explotación

cat defense/canary

Brute-forcing a 32-Bit Stack Canary

(simplified :))



cat info/format_string

Vulnerabilidad producida cuando se utiliza directamente una cadena de texto para ser interpretada con operadores de formato.

```c
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[]) {
    char buf[80];

    strcpy (buf, argv[1]);

    printf (buf);

    return 0;
}
```

Permite la inserción de operaciones de formato y por tanto de la lectura de las siguientes direcciones de memoria y escritura en una dirección de memoria en concreto.

./exploit2.py (Canary)

¡Pista!
El operador %p permite mostrar el valor de una palabra en memoria

Fases:

1. Desactivar ASLR
2. Encontrar las dos vulnerabilidades del código
3. Conocer el valor del canario a partir de la vulnerabilidad del string format
4. Buscar la dirección y el tamaño del buffer del stack overflow
5. Buscar el shellcode
6. Crear el payload
7. Proceder a su explotación

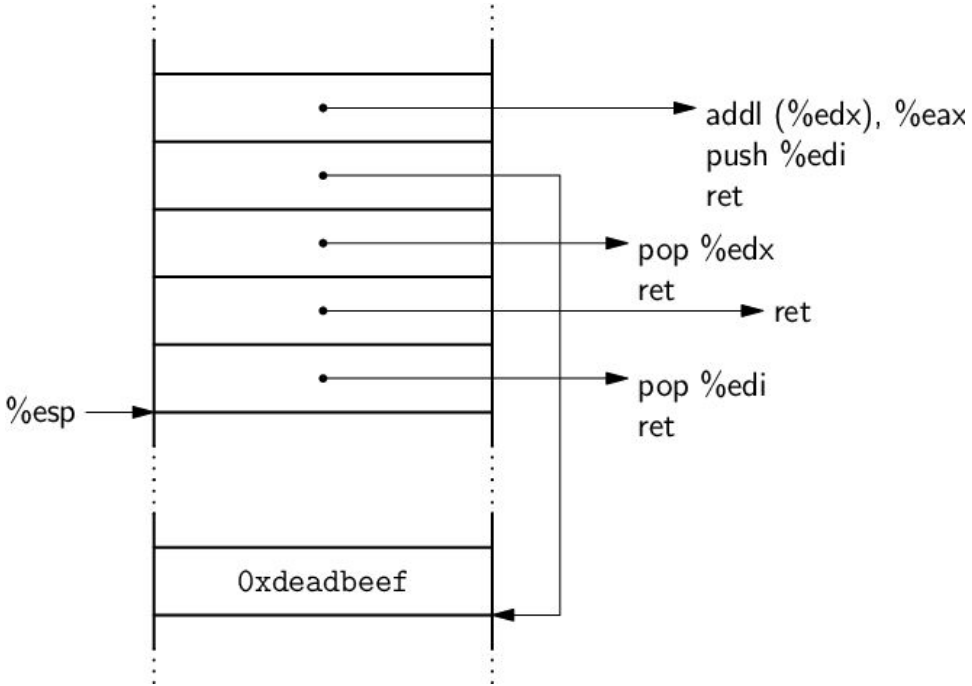
cat defense/NX (aka DEP)

```
root@localhost:~# checksec ret2text
[*] '/root/ret2text'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

Protección de la memoria de un proceso que no permite que exista zonas de memoria con los permisos de escritura y de ejecución.

cat info/ROP

pop ret



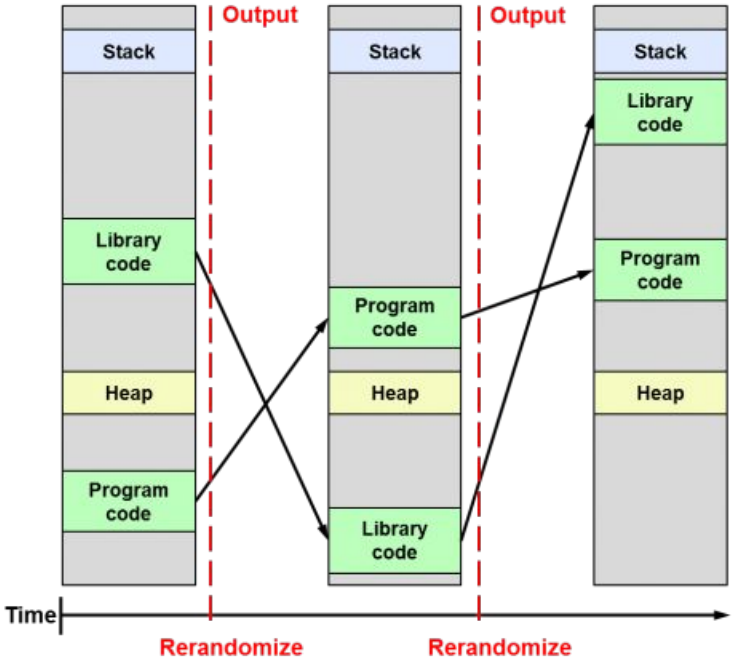
Buscar porciones de código en el binario para realizar la función que se quiera ejecutar.

./exploit3.py (NX)

Fases:

1. Desactivar ASLR
2. Encontrar la vulnerabilidad
3. Calcular la longitud hasta el RIP
4. Buscar el gadget ROP
5. Buscar en la librería libc la función deseada
6. Buscar en la librería libc la cadena “/bin/sh”
7. Crear el payload
8. Proceder a su explotación

cat defense/ASLR_PIE



Protección de memoria que aleatoriza el espacio de memoria de un proceso

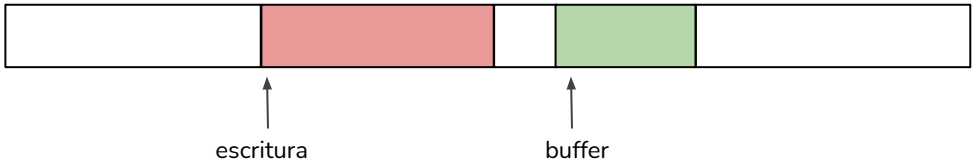
- ASLR. Protección en el kernel del sistema operativo. Modifica la dirección base del programa
- PIE. Protección opcional que se establece al compilar un programa. Modifica la dirección base del código

./exploit4.py (NX +ASLR)

Fases:

1. Encontrar las dos vulnerabilidades
2. Conocer el tamaño real del buffer
3. Conocer la dirección base del espacio de direcciones a partir del modo debug.
4. Buscar el codigo “pop rdi ret” en la librería libc (direccion real)
5. Buscar en la librería libc la función deseada (direccion real)
6. Buscar en la librería libc la cadena “/bin/sh” (direccion real)
7. Crear el payload
8. Proceder a su explotación

cat info/WoB



Error lógico de un programa que permite a un atacante hacer una escritura en la dirección de memoria deseada.

./exploit5.py (WoB)

Fases:

1. Encontrar la vulnerabilidad
2. Conocer dónde vamos a redirigir la ejecución del código
3. Conocer que vamos a sobreescribir
4. Conocer la diferencia entre la dirección base a escribir y la dirección deseada a escribir.
5. Crear el payload
6. Proceder a su explotación

cat info/heap_overflow

Vulnerabilidad de corrupción de memoria similar al Stack Overflow pero más limitada en lo que puede hacer. Permite modificar estructuras en memoria.

./exploit6.py (Heap Overflow)

Fases:

1. Analizar el binario para encontrar la vulnerabilidad.
2. Preparar el exploit.
3. Obtener la dirección base del binario.
4. Calcular el buffer hasta la dirección que nos interesa.
5. Obtener la dirección base de la libc.
6. Modificar la GOT para saltar a system.

FIN

¿Dudas?