Introduction to
Binary Exploitation

Plan

I. **PART 1**
   A. What is a binary ?
   B. What is binary exploitation ?
II. **PART 2**
   A. What is a process ?
   B. Process memory layout (Memory sections)
   C. What is the stack area and how does it work?
III. **PART 3**
   A. Common Vulnerabilities in PWN
      1. Integer overflow
      2. Buffer Overflow
   B. Useful tools
IV. **Ressources**

What is a Binary ?

In terms of Appearance: A file containing mostly non-readable characters, it's a succession of bytes, that couldn't be interpreted only by sight.

What is it really?: A file that is meant to be interpreted by a program, in order to do something specific.

example:

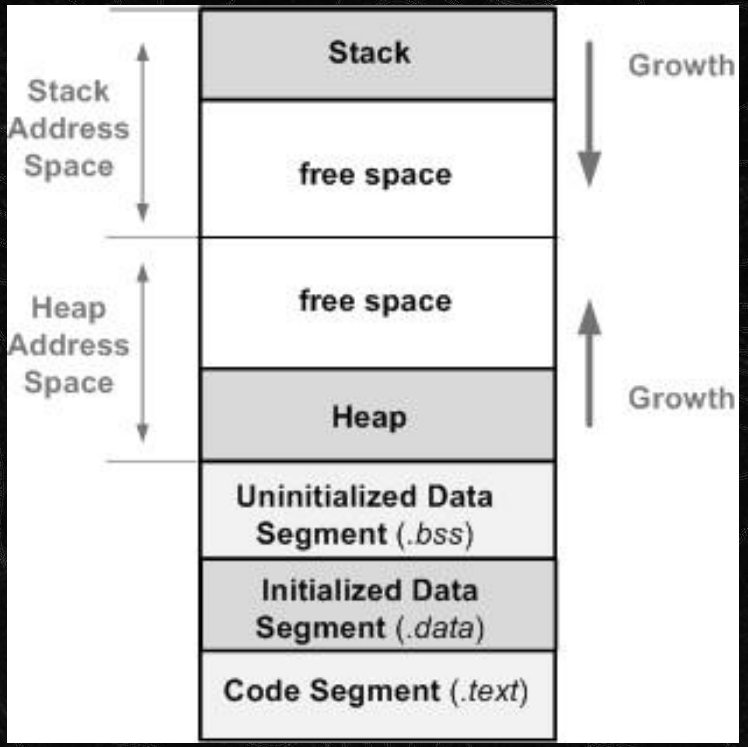
- Images are interpreted by A program (Image viewer… etc) to be seen as images
- ELF files, are executed by the system that understands the set of bytes in this format.

What is Binary exploitation ?

Also Called “PWN” , it consists on exploiting executables (ELF, PE, …etc).
Its main purpose is to find bugs (problems) in these, that could allow a hacker to steal important data or to take over a system

PWN VS REVERSE

PROCESSES


Stack
Growth
Stack Address Space
free space
free space
Heap Address Space
Heap
Growth
Uninitialized Data Segment (.bss)
Initialized Data Segment (.data)
Code Segment (.text)

THE STACK

A memory section used for storing dynamic data related to the execution of the program. As seen in the previous slide, it grows going from the higher to the lower address. it is manipulated using two instructions: **push && pop** .

We are mainly (in this workshop) manipulate:

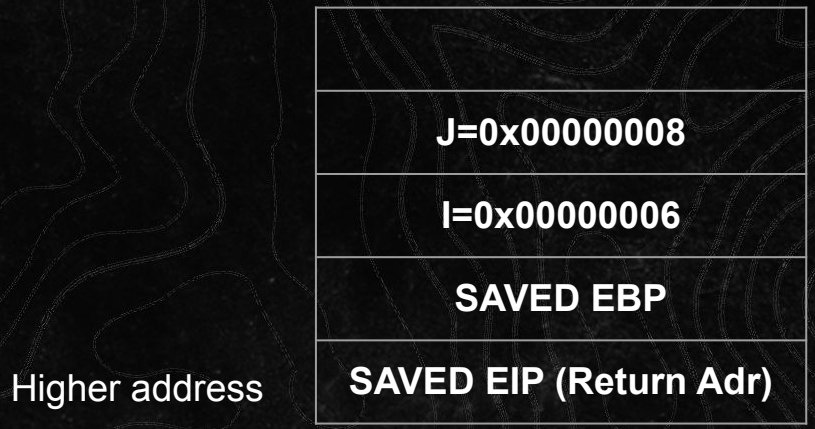
- Local variables && arguments
- return addresses

THE STACK

Let’s see how the stack grows in this case !

```
1   void func() {
2       int i=6 ;
3       int j=8 ;
4       printf("%d\n",i+j);
5   }
6
7   void main(){
8       func() ;
9   }
```

THE STACK




```c
void func() {
    int i=6 ;
    int j=8 ;
    printf("%d\n",i+j);
}

int main(){
    func() ;
    /*
        Some code here
    */
    return 0 ;
}
```

THE STACK

***
****
****
***
******
SAVE EIP (Return ADR func) POP EIP (ret)
B
A
SAVED EBP
SAVED EIP (Return ADR main)

```c
void func() {
    char buf[16] ;
    fgets(buf,16,stdin) ;
    printf("%s", buf) ;
}

int main(){
    int a, b;
    func() ;
    /*
        Some code here
    */
    return 0 ;
}
```

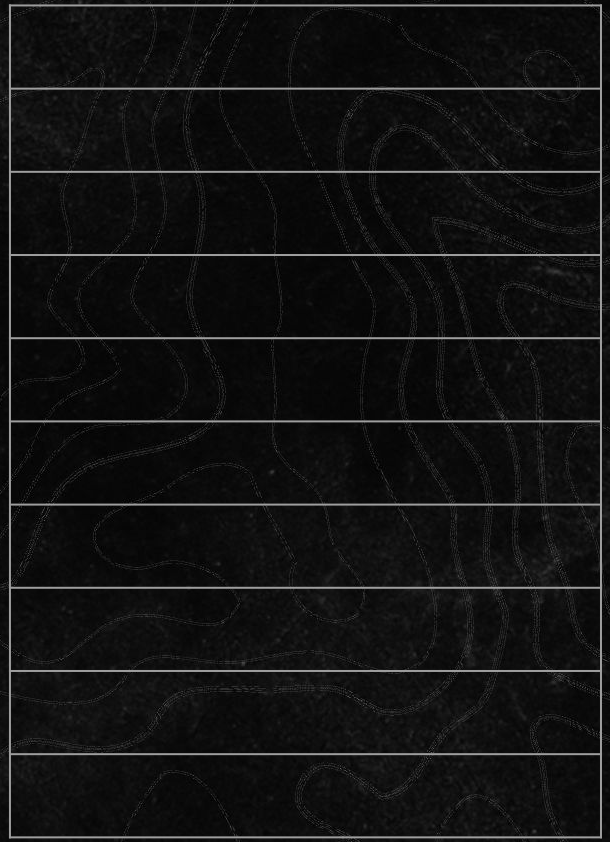
Common vulnerabilities in PWN

BUFFER OVERFLOW

Happens when the program doesn’t check the amount of data the user provides. SO, if he enters more data than expected, it may lead to dangerous data overwrite. And it could lead to:

- Unexpected program behaviour.
- Unexpected “program execution redirection”.

BUFFER OVERFLOW



```c
void func() {
    char buf[16] ;
    gets(buf);
    printf("%s", buf) ;
}

int main(){
    int a, b;
    func() ;
    /*
        Some code here
    */
    return 0 ;
}
```

BUFFER OVERFLOW

AAAA
AAAA
AAAA
AAAA
I = 0x41000000
save ebp
SAVE EIP

EXAMPLE 1

BUFFER OVERFLOW

AAAA
//AAAA
AAAA
AAAA
// ******//SAVE EBP
/********SAVE EIP (ret adr vuln)
SAVE EBP
SAVE EIP (ret adr main)

EXAMPLE 2

INTEGER OVERFLOW

When we surpass the max range of data representation in that format.

Example:

- Entering a number greater than 4294967295 for integers

Questions ?

Useful tools

Disassemblers:
- GDB (GNU DEBUGGER)
- radare2

Decompilers:
- GHIDRA
- IDA

Programming languages:
- Assembly && C && C++
- Python for scripting
- Some Linux knowledge

Other tools:
- readelf / objdump / strings
- checksec
- ltrace && strace

Ressources

- Binary exploitation Playlist by LiveOverflow (YouTube)
- PwnCollege Platform
- Nightmare
- PicoCTF
- RootME

Thank You