a1-milestone
Weicheng Wang
w257wang

sploit1.c: off-by-one

Sploit1.c performs a exploit to do a “off-by-one” buffer over flow that overflows the snprintf function in print_usage function by one byte.

In order to get into that function, we need no command argument been passed into submit.c. So I set args[1] to args[3] to null. and snprintf takes cmd as one of its argument which is args[0]. Thanks to execve(), we got to change arg[0] to what we want. I passed in a string that first byte is NOP had rest are 4 byte address which should points to the address in env[0]. where is stored a list of NOP and shell code.
since I put a NOP in the front of arg[0]. The char string will get off by one to the frame pointer, therefore the LSB is changed to point forward. when print_usage return, because of the wrong frame pointer set by us in snprintf, function will return to where our shell code locate and run our shell code, therefore we get root.

sploit2.c: smash the stack

Sploit2.c performs a exploit to do a “smash the stack” buffer over flow, which overwrites the return address in check_for_viruses.

I noticed there is a buffer at the first line of the function. That means the function stack should be looks like this:

```
[              ] [    ] [    ] |
   buf            fp    ret  |
```

Therefore if we can overflow the buf over the ret and change it to our shell code, we can get the root from this. In this function, everything in the file, which should be submit, is put into the buf by character(by byte) until a ‘\0’ is reached. I use this and write the shell code into the file followed by an address, which checked using gdb, which points into the buf. In order to pass the virus test, which is a regular expression check checking if “bin/sh” is in the file, we need to let bin/sh not completely in the buf.