Lab #3. ROP & Challenges

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

General Information

- **Check "Lab #3" in *Assignment* tab of *Cyber Campus***
  - Skeleton code (**`Lab3.tgz`**) is attached in the post
  - Deadline: **11/15 Friday 23:59**
  - Submission will be accepted in that post, too
  - Late submission due: **11/17 Sunday 23:59 (-20% penalty)**
  - Delay penalty is applied uniformly **(not problem by problem)**
- **Please read the instructions in this slide carefully**
  - This slide is a 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
- **You must submit a report as well**
  - More instructions are provided at the end of this slide

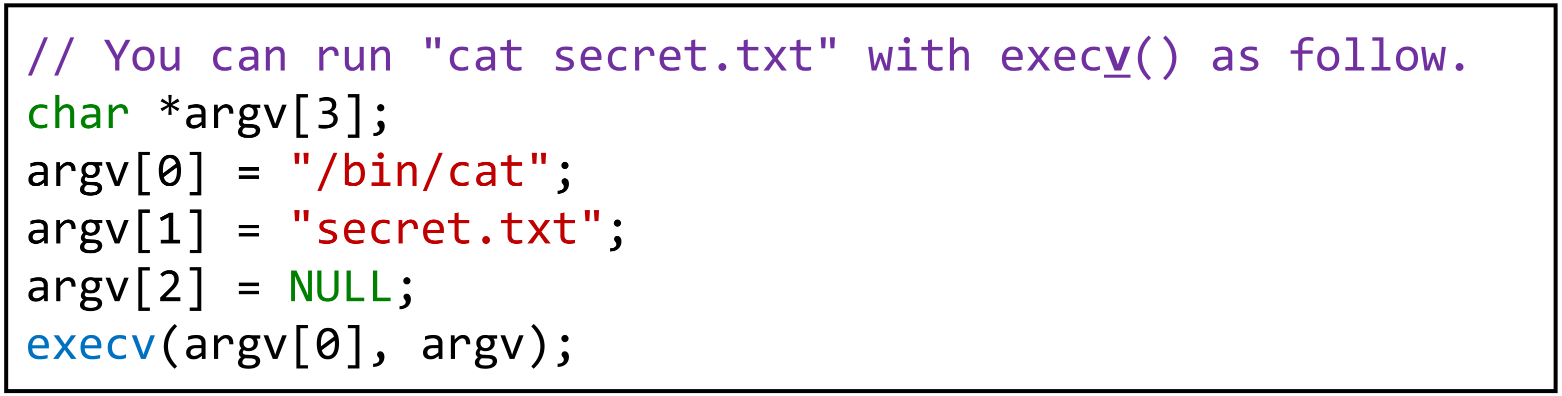
Skeleton Code Structure

- **Copy `Lab3.tgz` into CSPRO server and decompress it**
  - You **must connect to** `csproN.sogang.ac.kr` **(N = 2, 3, or 7)**
- **Skeleton code has similar structure to the previous lab**
  - **`3-1/ ... 3-4/`** : Problems that you have to solve
  - **`3-5/`** : *Bonus problem* for practice **(not included in grading)**
    - But this one cab be important when preparing the lab exam
  - **`check.py`**, **`config`**: Files for self-grading
- **This slide will provide a guide on writing ROP exploit**
  - It also provides a detailed tutorial for solving **`3-1`**

```
jschoi@cspro2:~$ tar -xzf Lab3.tgz
jschoi@cspro2:~$ ls Lab3
3-1  3-2  3-3  3-4  3-5  check.py  config
```

Reading `serect.txt`

- **In the lecture slide, we talked about `execve()` function**
  - But there are other variants of like **`execv()`**, **`execle()`**, …
- **In this lab, you must run the following code* with ROP**
  - There are other ways to read **`secret.txt`**, but **don’t use them**
    - Ex) Using **`system()`** instead, or spawning a shell with **`execv()`**
  - They may not work and you can even **get 0 point** in such cases
    - For instance, **`system()`** does not work properly with SUID



```
// You can run "cat secret.txt" with execv() as follow.
char *argv[3];
argv[0] = "/bin/cat";
argv[1] = "secret.txt";
argv[2] = NULL;
execv(argv[0], argv);
```

* Except for 3-3 (this one can be solved without control hijack)

Example: Problem 3-1

- **Target program (`twice.c` / `twice.bin`) is given**

```c
void run_cat(char *filepath) {
  char *argv[3];
  argv[0] = "/bin/cat";
  argv[1] = filepath;
  argv[2] = NULL;
  execv(argv[0], argv);
}
...
void vuln(void) {
  char buf[20];
  printf("Input your message in stack buffer: ");
  read(0, buf, 64);
}
```

Your goal is to execute this function with "**`secret.txt`**"

You can see that BOF occurs here

Finding ROP Gadgets

- **In principle, you must disassemble all the addresses in the code section, which contains assembly instructions**
- **Pwntools offers `ROP()` API that does this automatically**
  - **`print(rop.rdi)`**: Print gadgets that can affect **`%rdi`** register
- **Tip: You can use `p64()` function to write concise code**
  - FYI, **`u64()`** function performs conversion in opposite direction

```
p = process("./twice.bin")
rop = ROP("./twice.bin")
# You can print the gadget information as follow.
print(rop.rdi)

# The following two lines have the same meaning.
rdi_gadget = b"\xb3\x12\x40\x00\x00\x00\x00\x00"
rdi_gadget = p64(0x4012b3) # More concise
```

Attaching GDB to Process

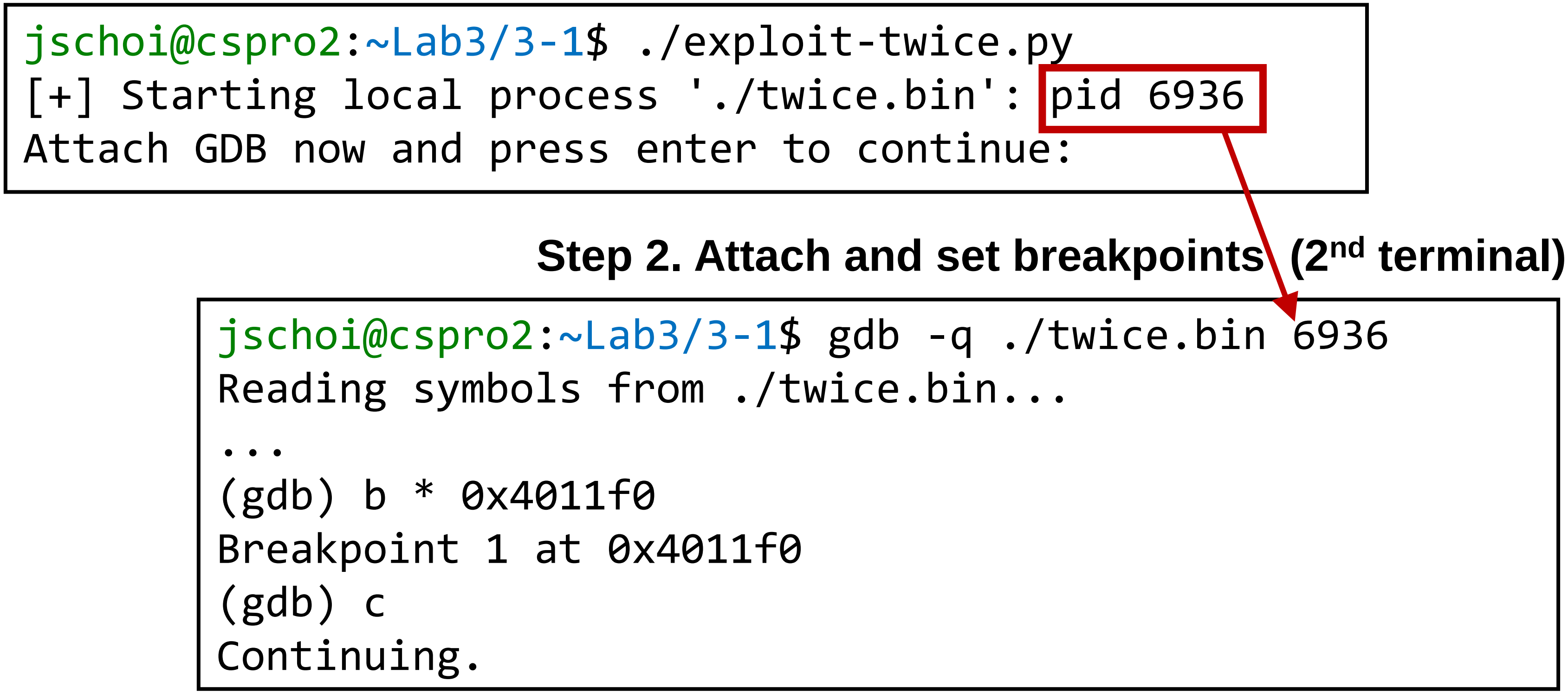
- **Assume that you wrote the exploit code below**
  - It uses ROP gadget to change the value of **`%rdi`** into **`0x4142`**
- **Let's use gdb to check if this works as expected**
  - Previously, we launched **`gdb`** and started a process from there
  - This time, let's run the script and attach to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

```
p = process("./twice.bin")

# You can use this line to pause the script for a while.
input("Attach GDB now and press enter to continue: ")
...
print(p.recvuntil(b"stack buffer: "))
rdi_gadget = p64(0x4012b3)
p.send(b"a" * 0x28 + rdi_gadget + p64(0x4142))

input("Done, but let me wait for a while...")
```

Attaching GDB to Process

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

Step 1. Start the exploit script (1st terminal)



```
jschoi@cspro2:~Lab3/3-1$ ./exploit-twice.py
[+] Starting local process './twice.bin': pid 6936
Attach GDB now and press enter to continue:
```

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

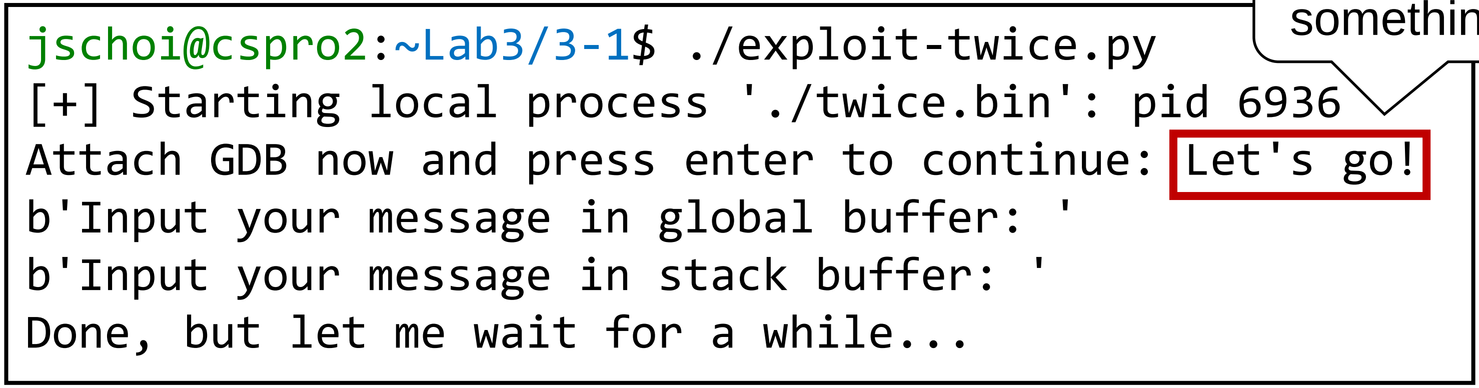
```
jschoi@cspro2:~Lab3/3-1$ gdb -q ./twice.bin 6936
Reading symbols from ./twice.bin...
...
(gdb) b * 0x4011f0
Breakpoint 1 at 0x4011f0
(gdb) c
Continuing.
```

Attaching GDB to Process

- **You must open two terminals and switch between them**
  - In the 2nd terminal, you can use the **gdb** commands to debug

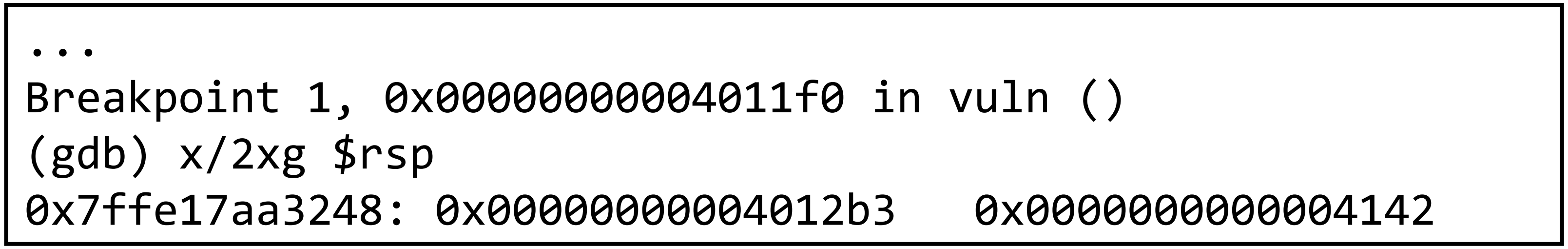
Step 3. Resume the exploit script (1st terminal)

You must type in something like this



```
jschoi@cspro2:~Lab3/3-1$ ./exploit-twice.py
[+] Starting local process './twice.bin': pid 6936
Attach GDB now and press enter to continue: Let's go!
b'Input your message in global buffer: '
b'Input your message in stack buffer: '
Done, but let me wait for a while...
```

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



```
...
Breakpoint 1, 0x00000000004011f0 in vuln ()
(gdb) x/2xg $rsp
0x7ffe17aa3248: 0x00000000004012b3   0x0000000000004142
```

Demonstration

Obtaining Function Offset

- **For problem 3-2, you will have to obtain the offset of a function within the `libc` library**
  - Recall that you need this information to figure out the address of **`execv()`** function, using memory disclosure
- **You can do this easily by using the `pwntools` API**
  - Then you don’t have to hard-code constants in your script

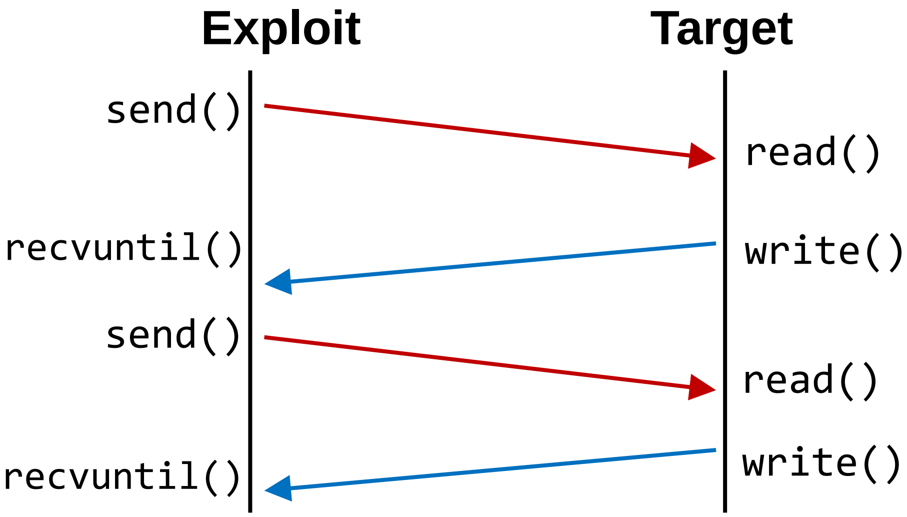
```python
# You can investigate the offset of libc functions as follow.
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
read_offset = libc.symbols['read']
execv_offset = libc.symbols['execv']
print("Offset of read() within library: 0x%x" % read_offset)
print("Offset of execv() within library: 0x%x" % execv_offset)
```

Hints

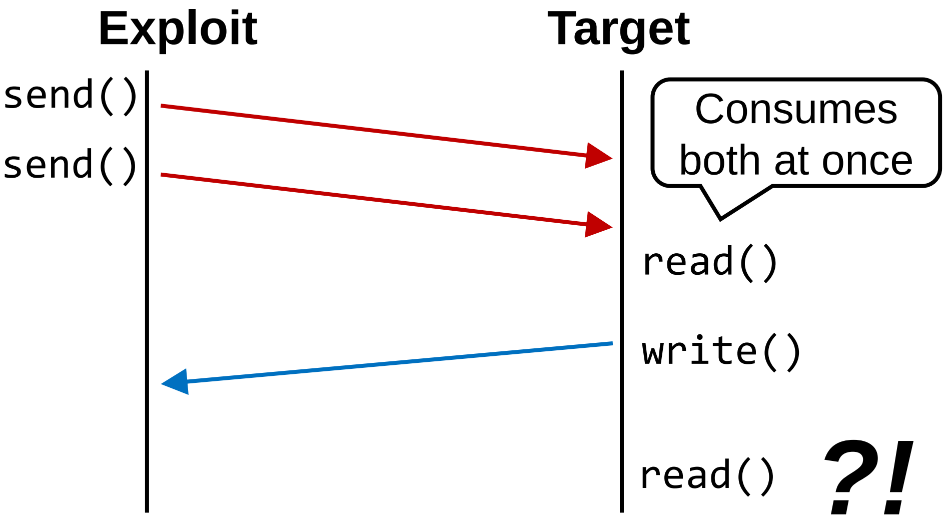
- **In 3-1, be careful in handling newline (`\n`) and null (`\0`)**
  - Recall that **`pwntools`**' **`sendline()`** implicitly appends **`'\n'`**
- **In 3-2, you must leak the addresses of `libc` functions**
  - Try to disclose the library addresses stored in **GOT**
  - A function's **GOT** entry is filled in when it's *called for the first time*
- **In 3-3, you will have to exploit a format string bug to disclose the memory content of an arbitrary address**
- **In 3-4, you must exploit a use-after-free vulnerability**
  - First, examine the behavior of memory allocator, by writing a simple program with **`malloc()`** and **`free()`** sequence
  - In other words, think about how to make the allocator return the freed block that you want

Caution: Reliability of Exploit

- **In this lab, your script has to be especially careful in interacting with the target program carefully**
  - Make sure that you send and receive message **step by step**
  - If not, your exploit code may not work reliably (if it doesn't work during the actual grading for this reason, I will deduct point)



This is what we want



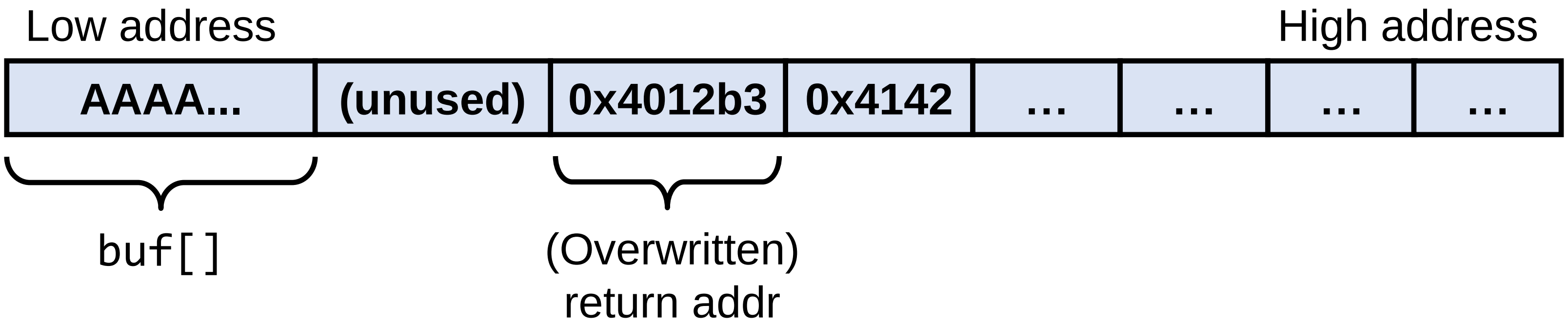
But this can happen

Report Guideline

- **Write report for 3-2 and 3-3 (not required for 3-1 and 3-4)**
  - 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 point; it is only used to deduct point
- **This time, I will provide a template for each problem**
  - Make sure that your report contains the requested content
- **If you used ChatGPT to write your exploit code, clearly describe it in your report (review the orientation slide)**

Report Template for 3-2

- **Draw the state of stack frame after your input overflows the buffer (see the example below)**
  - Draw it just as I did in the lecture slide of the ROP chapter
  - Explain the **meaning (role) of each memory** block in the figure
  - Ex) If it's a gadget address, explain what that gadget does
  - Ex) If it's an address of a function, explain what arguments you are trying to pass, and why you are doing that
    - If you are passing a pointer (memory address) as a function argument, explain what is stored in that address



Low address

High address

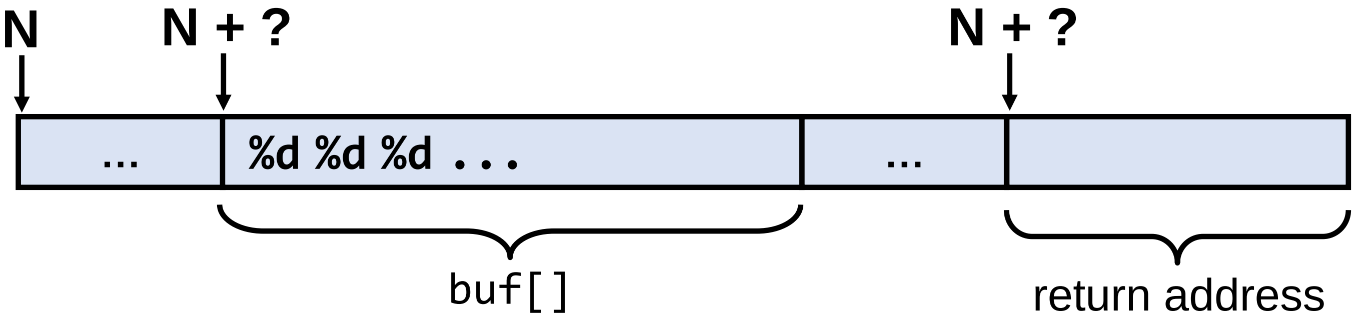
AAAA...	**(unused)**	**0x4012b3**	**0x4142**	**…**	**…**	**…**	**…**

buf[]

(Overwritten) return addr

Report Template for 3-3

- **Draw the state of `main()`'s stack frame immediately before `printf()` is called**
  - Clearly indicate the positions of **`buf[]`** and saved return address in the stack frame
  - **N** must be the value of **`%rsp`** at address **`0x4011e9`**
  - You will be entering format specifiers as input ("**`%d%d%d...`**"); so explain which stack position is consumed by each specifier
  - Justify why you repeat each format specifier for certain number of times



Problem Information

- **There are four problems you have to solve (25 pt. each)**
  - Problem 3-1: **`twice.bin` (★)**
  - Problem 3-2: **`substr.bin` (★★★)**
  - Problem 3-3: **`fsb.bin` (★★☆)**
  - Problem 3-4: **`item.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 how you analyzed and solved the problem, you will lose points**
  - You can write the report in Korean or English
  - Due to the limited time, I will randomly select a problem to grade

Submission Guideline

- **You should submit four exploit scripts and report**
  - Problem 3-1: **`exploit-twice.py`**
  - Problem 3-2: **`exploit-substr.py`**
  - Problem 3-3: **`exploit-fsb.py`**
  - Problem 3-4: **`exploit-item.py`**
  - **Don’t forget the report**: **report.pdf**
  - 3-5 is a bonus problem, so you don’t have to submit it
- **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**