CSED211: Lab. 4

AttackLab

조승혁

shhj1998@postech.ac.kr

POSTECH

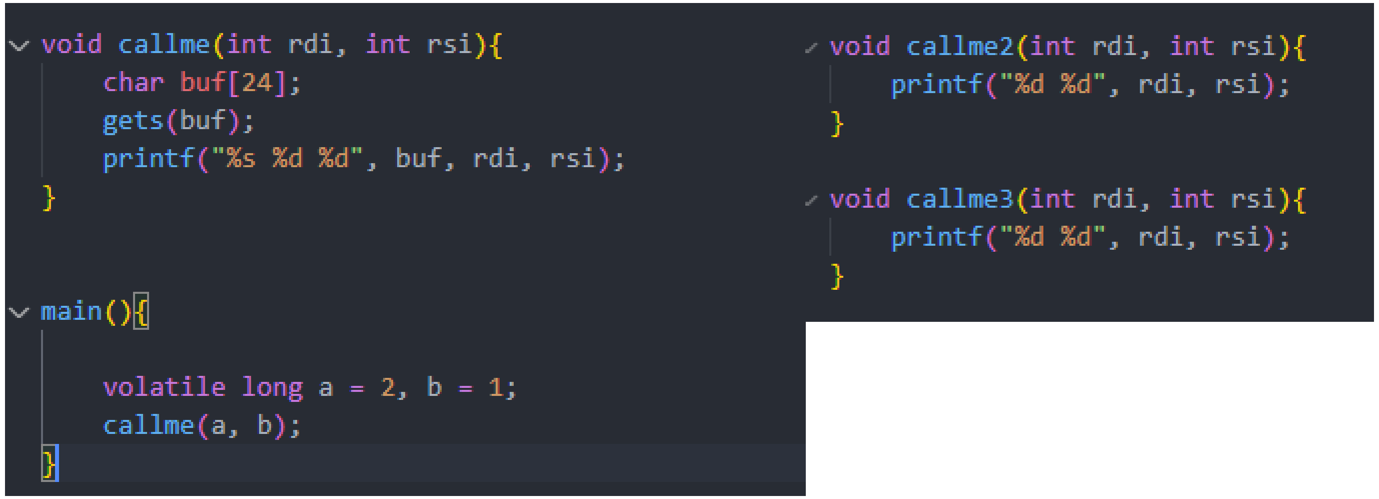
2023.10.30

Table of Contents

- Return Oriented Programming
- Defense Methods
- Homework: Attack Lab

Buffer Overflow: Example

- From the following code, we will attempt to call the functions `callme2` and `callme3` subsequently.

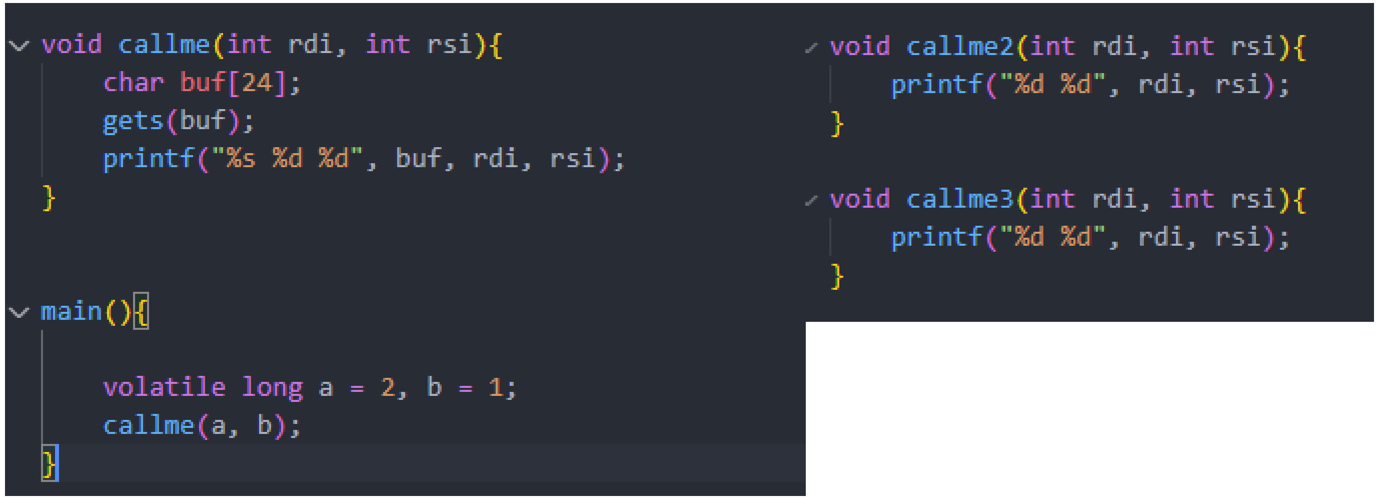


```c
void callme(int rdi, int rsi){
    char buf[24];
    gets(buf);
    printf("%s %d %d", buf, rdi, rsi);
}


main(){

    volatile long a = 2, b = 1;
    callme(a, b);
}
```

```c
void callme2(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}

void callme3(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}
```

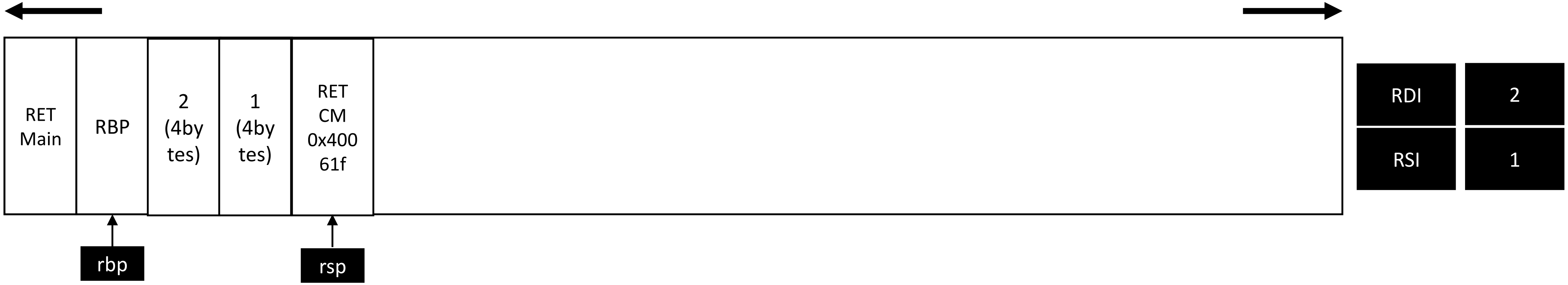
Buffer Overflow: Example

- The memory layout when we call `main`.



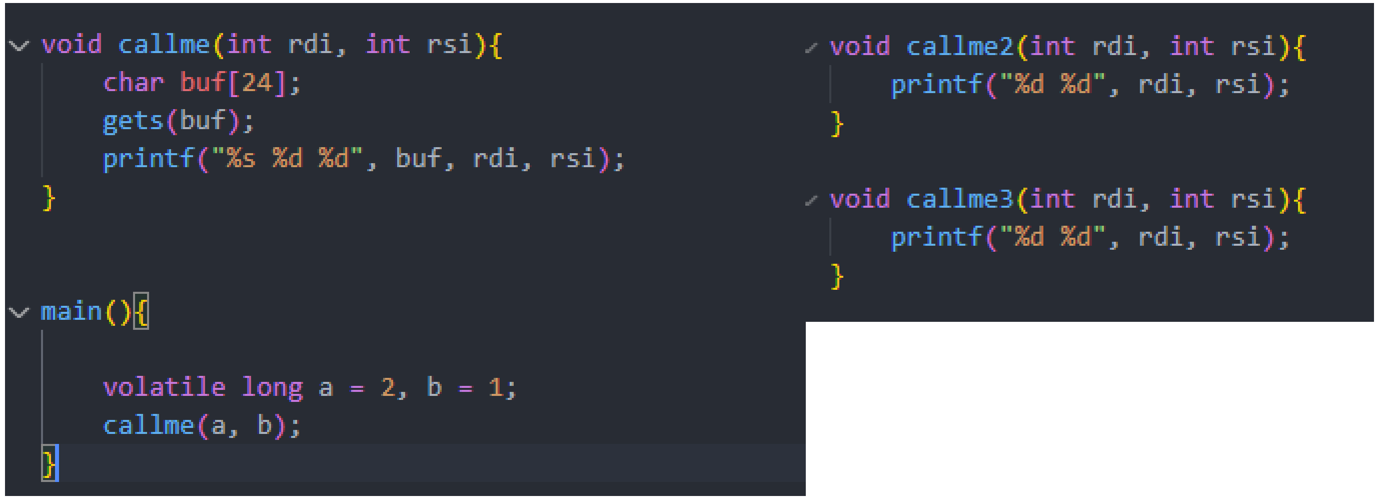
```
void callme(int rdi, int rsi){
    char buf[24];
    gets(buf);
    printf("%s %d %d", buf, rdi, rsi);
}


main(){

    volatile long a = 2, b = 1;
    callme(a, b);
}
```

```
void callme2(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}

void callme3(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}
```



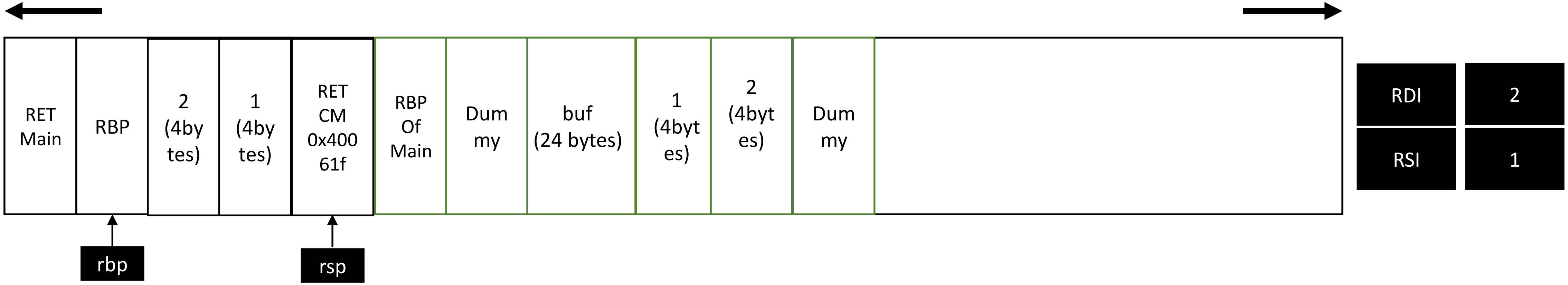
Buffer Overflow: Example

- The memory layout when we call `callme`.



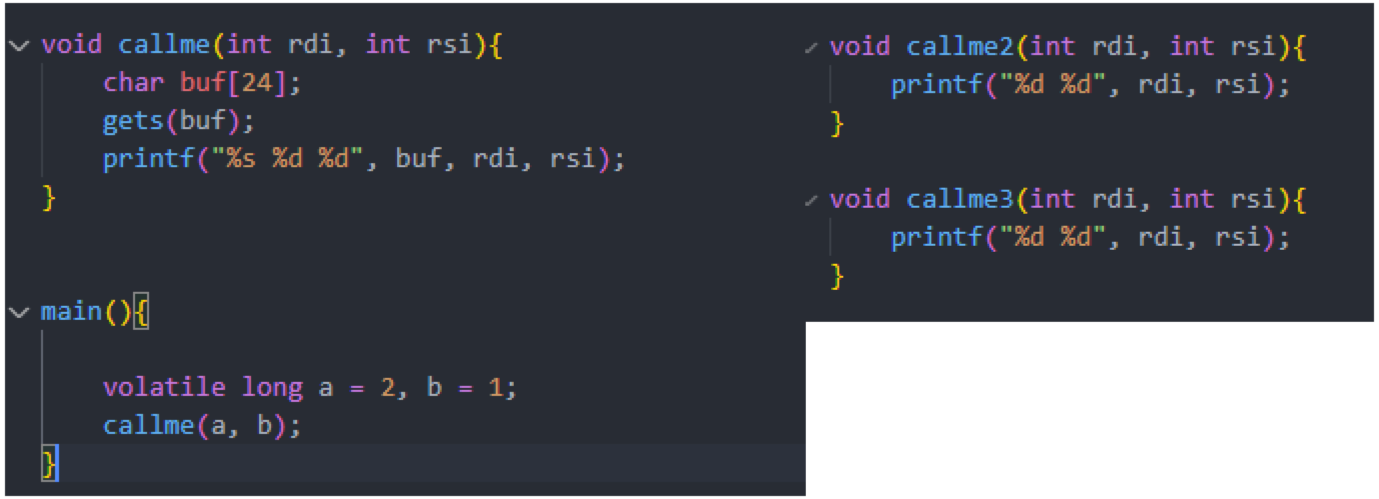
```
void callme(int rdi, int rsi){
    char buf[24];
    gets(buf);
    printf("%s %d %d", buf, rdi, rsi);
}


main(){

    volatile long a = 2, b = 1;
    callme(a, b);
}
```

```
void callme2(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}

void callme3(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}
```



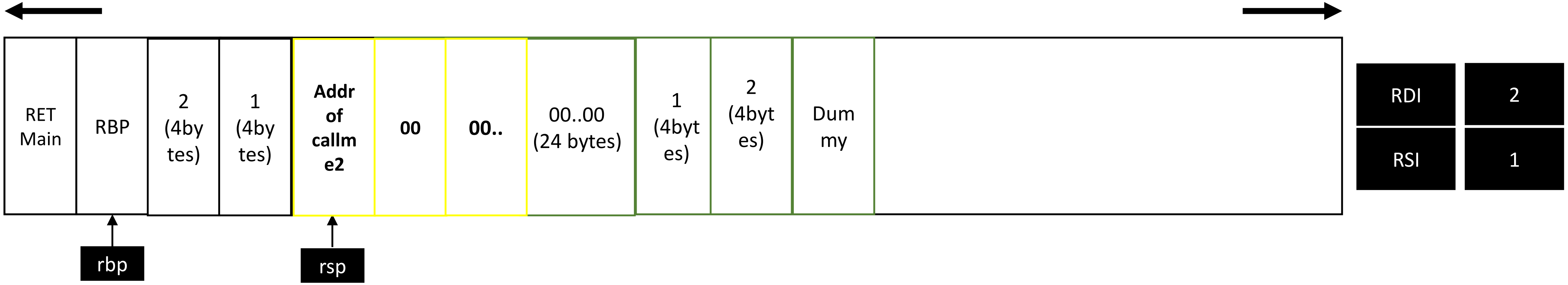
Buffer Overflow: Example

- We can call `callme2` by putting the address of `callme2` to rsp.
  - Exploit string = “0x00” x 24 + “0x00” x 8 + “0x00” x 8 + <address of `callme2`>.



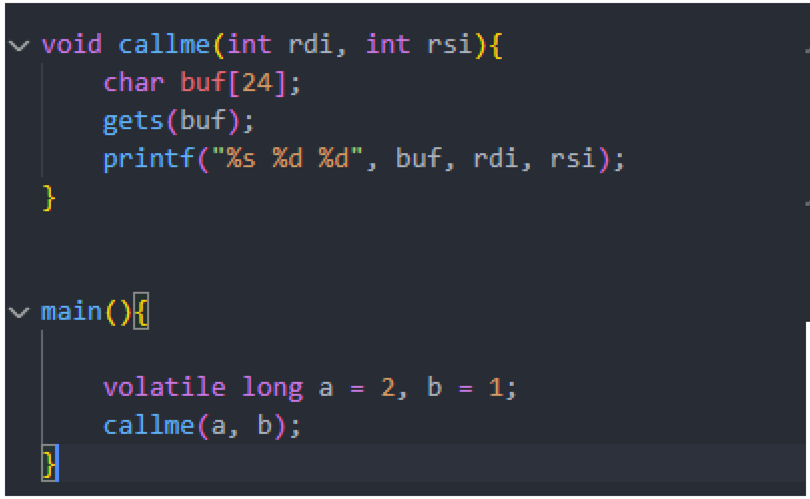
```
void callme(int rdi, int rsi){
    char buf[24];
    gets(buf);
    printf("%s %d %d", buf, rdi, rsi);
}


main(){

    volatile long a = 2, b = 1;
    callme(a, b);
}
```

```
void callme2(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}

void callme3(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}
```

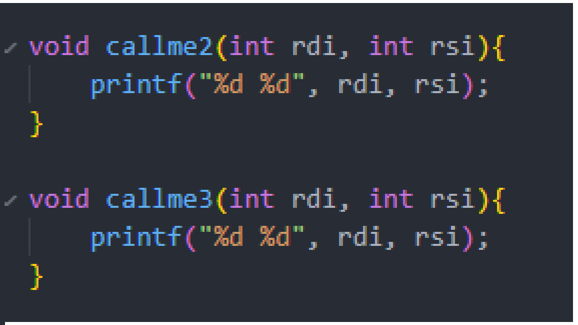


Buffer Overflow: Example

- However, it is hard to pass the original arguments which are stored in rdi and rsi.
  - Q) How can we call `callme2` with the arguments?



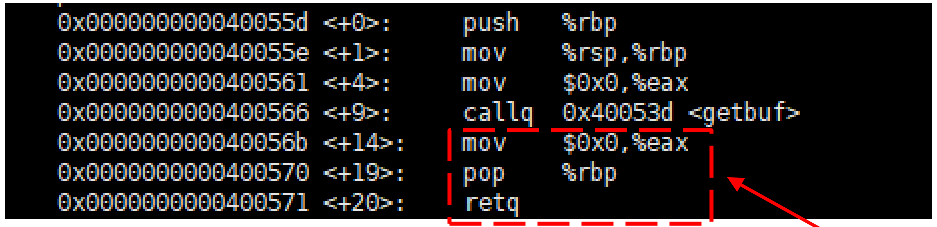
```
void callme(int rdi, int rsi){
    char buf[24];
    gets(buf);
    printf("%s %d %d", buf, rdi, rsi);
}


main(){

    volatile long a = 2, b = 1;
    callme(a, b);
}
```



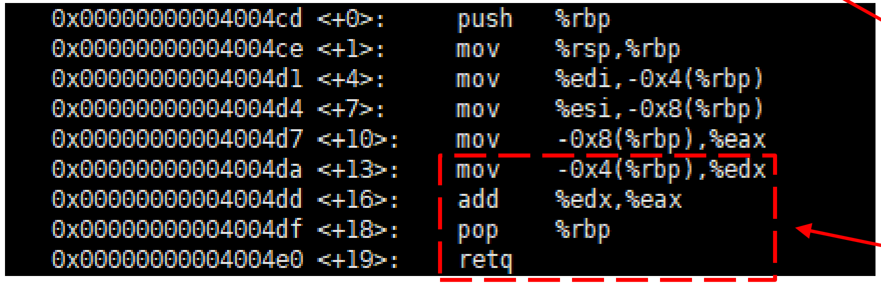
```
void callme2(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}

void callme3(int rdi, int rsi){
    printf("%d %d", rdi, rsi);
}
```

Return Oriented Programming (ROP)

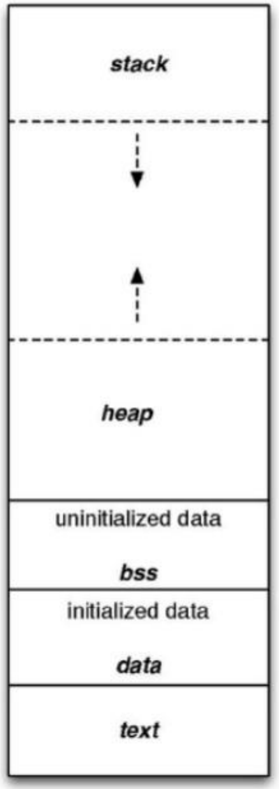
- Exploit parts of existing codes (usually codes in library) which include **RET** instruction.
    - Recall that RET pops stack.
- Gadget: A small code (code snippet) which ends with RET instruction.



```
0x000000000040055d <+0>:     push   %rbp
0x000000000040055e <+1>:     mov    %rsp,%rbp
0x0000000000400561 <+4>:     mov    $0x0,%eax
0x0000000000400566 <+9>:     callq  0x40053d <getbuf>
0x000000000040056b <+14>:    mov    $0x0,%eax
0x0000000000400570 <+19>:    pop    %rbp
0x0000000000400571 <+20>:    retq
```



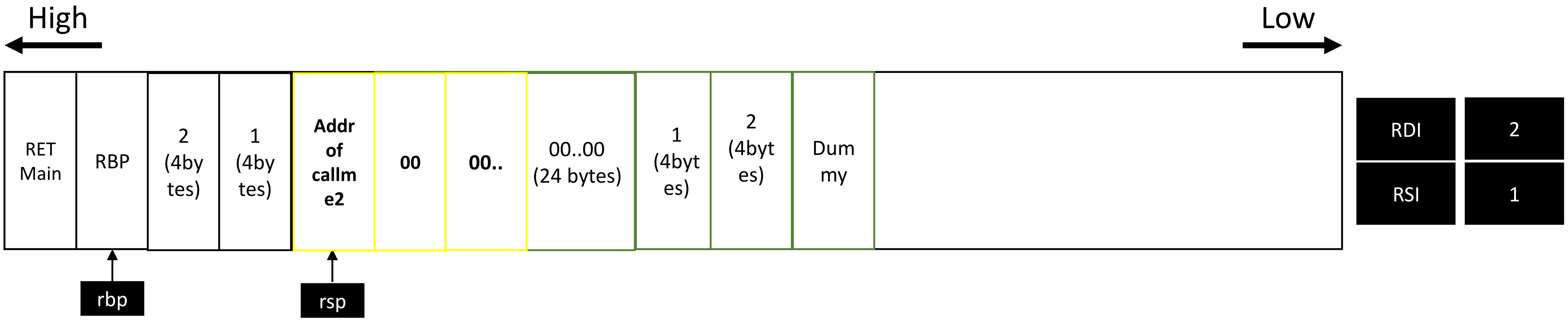
```
0x00000000004004cd <+0>:     push   %rbp
0x00000000004004ce <+1>:     mov    %rsp,%rbp
0x00000000004004d1 <+4>:     mov    %edi,-0x4(%rbp)
0x00000000004004d4 <+7>:     mov    %esi,-0x8(%rbp)
0x00000000004004d7 <+10>:    mov    -0x8(%rbp),%eax
0x00000000004004da <+13>:    mov    -0x4(%rbp),%edx
0x00000000004004dd <+16>:    add    %edx,%eax
0x00000000004004df <+18>:    pop    %rbp
0x00000000004004e0 <+19>:    retq
```



Buffer Overflow: ROP

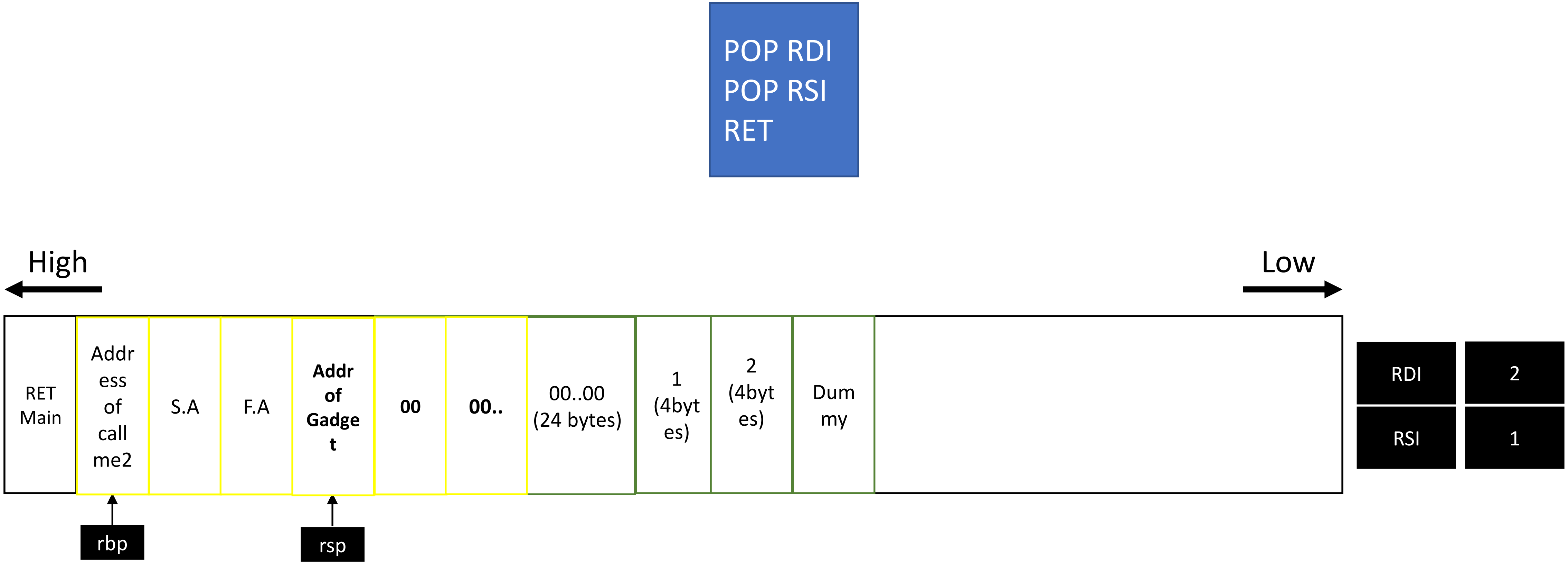
- We need to pass two arguments using `rdi` and `rsi` register.
  - Find the address of the gadget with the following code in the code section (`.text`)!

```
POP RDI
POP RSI
RET
```



Buffer Overflow: ROP

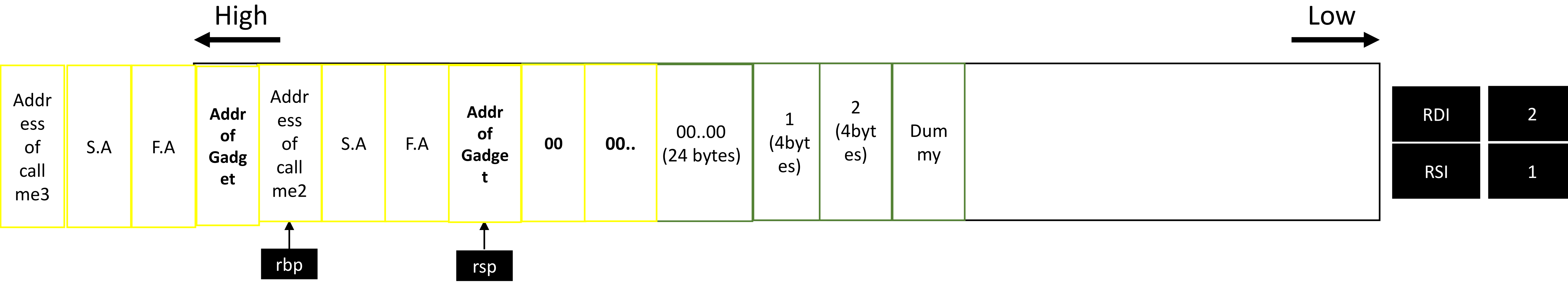
- The exploit string calling `callme2` with the two arguments becomes:
  - “0x00” x 24 + “0x00” x 8 + “0x00” x 8 + <address of gadget> + <first argument> + <second argument> + <address of `callme2`>.



Buffer Overflow: ROP

- We can also call `callme3` subsequently in a similar way:
  - “0x00” x 24 + “0x00” x 8 + “0x00” x 8 + <address of gadget> + <first argument> + <second argument> + <address of `callme2`>.
  - + <address of gadget> + <first argument> + <second argument> + <address of `callme3`>.

```
POP RDI
POP RSI
RET
```



High ← → Low

Address of callme3	S.A	F.A	**Addr of Gadget**	Address of callme2	S.A	F.A	**Addr of Gadget**	**00**	**00..**	00..00 (24 bytes)	1 (4bytes)	2 (4bytes)	Dummy	
				rbp			rsp							

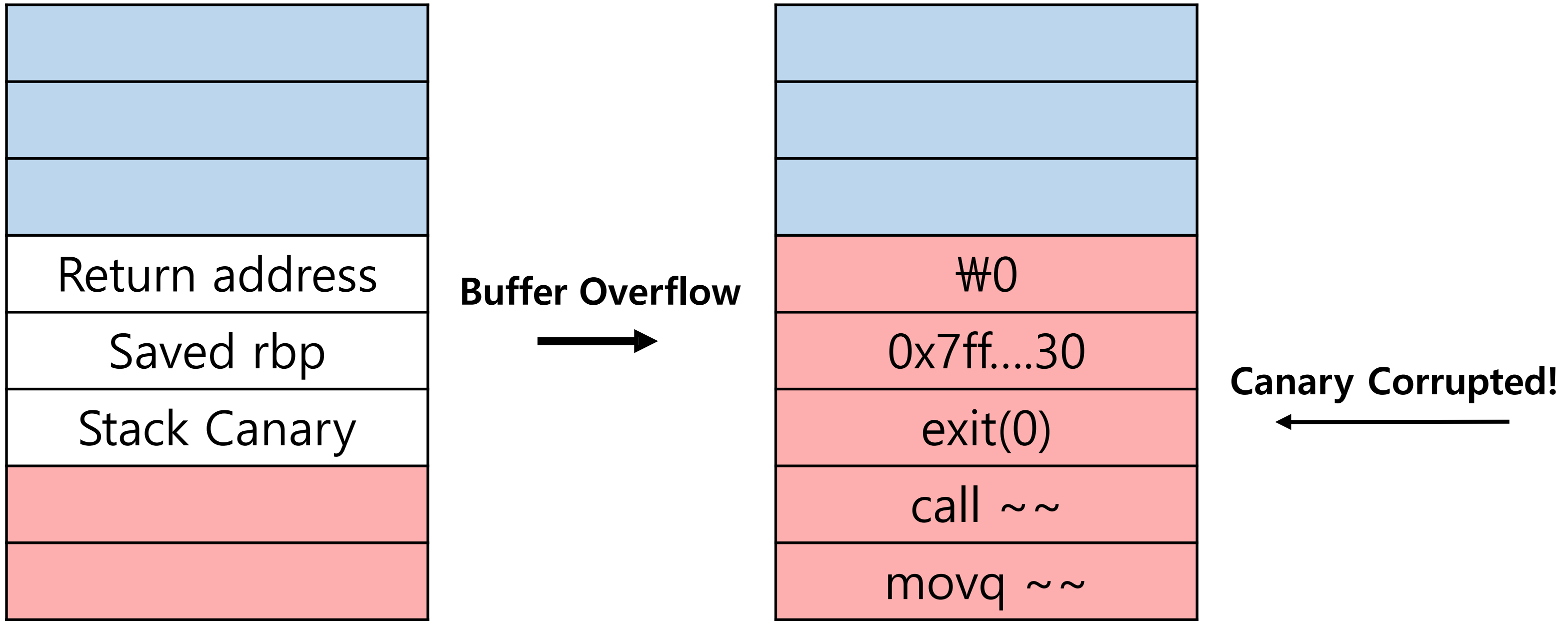
RDI	2
RSI	1

Defense

- There are three ways to defense buffer overflow attack.
  - Stack canary.
  - Data execution prevention (DEP) / No execute (NX) bit.
  - Address space layout randomization (ASLR).

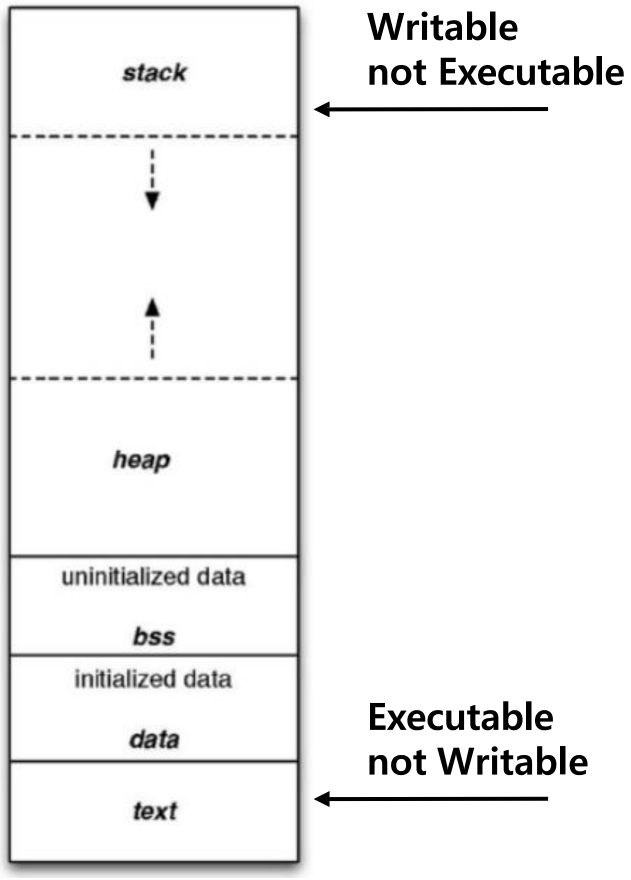
Stack Canary

- We can detect buffer overflow attack by observing the change of a value (canary).
- Canary is placed between a buffer and return address



Data execution prevention (DEP) / No execute (NX) bit

- Make injected instructions not executable.

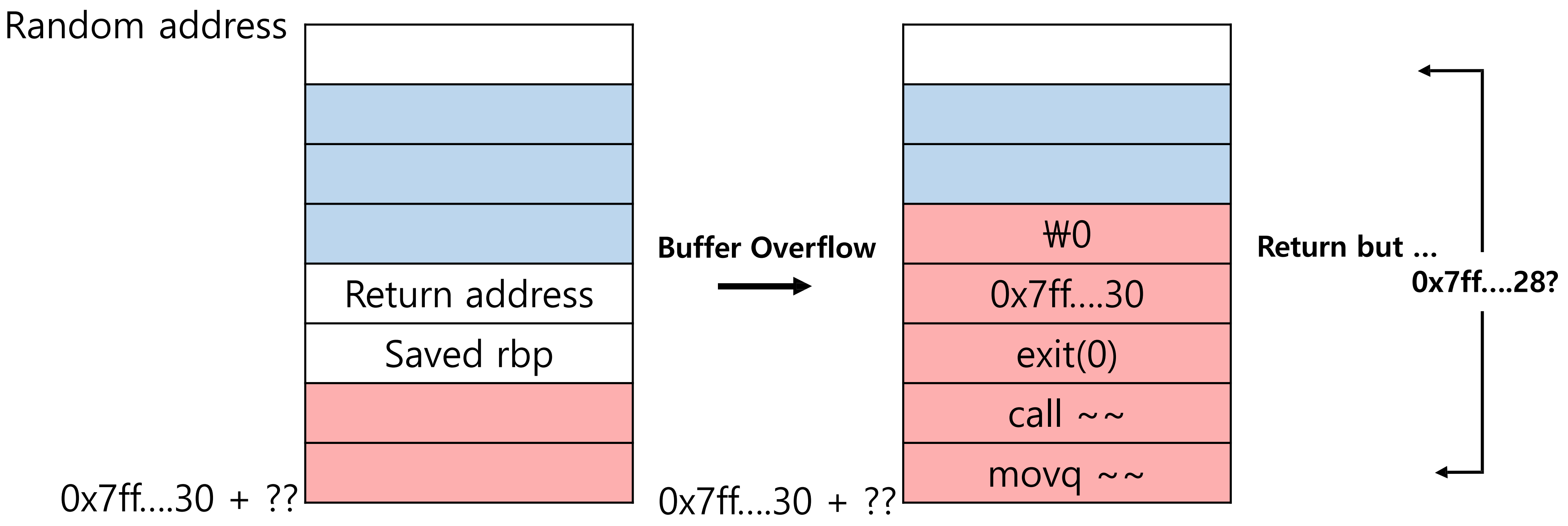


Writable
not Executable

Executable
not Writable

Address space layout randomization (ASLR).

- We can bother the buffer overflow attack by randomly initialize the start address of stack.



Homework (Attack Lab)

- Make sure that you enable local forwarding to access attack server.
- To download your target, go to http://127.0.0.1:15513.
  - Enter your information, student ID and school email.
  - Upload your target#.tar to the programming server.
- Your goal is to exploit the **five** targets:
  - ctarget.l1, ctarget.l2, ctarget.l3, rtarget.l2, rtarget.l3.
- Your score (corresponds to target #) will be automatically uploaded at:
  - http://127.0.0.1:15513/scoreboard.
  - Target can be exploited only in the programming server.
  - The score is not updated if you work in other machines.

Homework (Attack Lab)

- You can find more details in `writeup_attacklab.pdf`.

Homework (Attack Lab)

- Deadline: 11/6 23:59 (Mon)
- You need to upload a report in the PLMS.
  - Explain how did you exploit the target programs in the report.
  - Follow the file name format, [student#].pdf.
    - For example, 2020xxxx.pdf (No square brackets in the file name).
    - **No doc, No zip!**

Quiz