[Chap.3-7] Machine-level Representation of Programs

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

SKK UNIVERSITY

Contents

- ...
- Procedures
- Compound data structures
- Pointers
- GDB debugger
- Buffer overflow
- Floating-point codes

Buffer Overflow

Buffer overflow

- C does not perform any bounds checking for array references
- Local variables are stored on the stack along with state information (such as saved register values and return addresses)

- These lead to serious program errors, where the state stored on the stack gets corrupted by a write to an out-of-bounds array element

- **Buffer overflow**

Buffer Overflow

Vulnerable codes

```
/* Echo line */
void echo()
{
    // Way too small!
    char buf[8];
    gets(buf);
    puts(buf);
}

int main()
{
    printf("Type: ");
    echo();
    return 0;
}
```

```
$ ./bufdemo
Type: 123
123

$ ./bufdemo
Type: 1234567
1234567

$ ./bufdemo
Type: 123456789abcdef
Segmentation Fault
```

Buffer Overflow

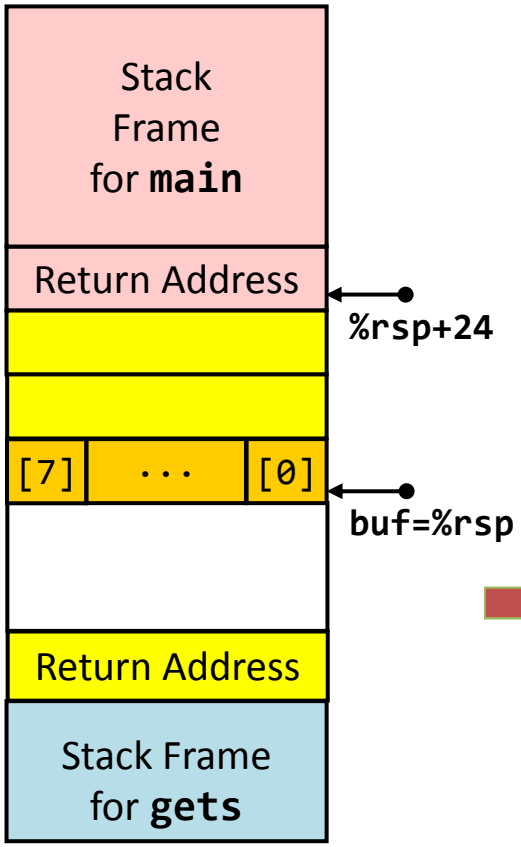
Vulnerable codes

- Unix implementation of gets()
  - No way to specify limit on # of characters to read

```
/* Implementation of library function gets() */
char *gets(char *s)
{
   int c;
   char *p = s;
   while ((c = getchar()) != '\n' && c != EOF)
      *p++ = c;
   if (c == EOF && p == s)
      /* No characters read */
      return NULL;
   *p++ = '\0';
   return s;
}
```

Buffer Overflow

■ Vulnerable codes




```
/* Echo Line */
void echo()
{
    char buf[8];        /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
    subq $24,%rsp       Allocates 24 bytes on stack
    movq %rsp,%rdi      Compute buf as %rsp
    call gets           Call gets
    movq %rsp,%rdi      Compute buf as %rsp
    call puts           Call gets
    addq $24,%rsp       Deallocate stack space
    ret                 Return
```

Buffer Overflow

Vulnerable codes

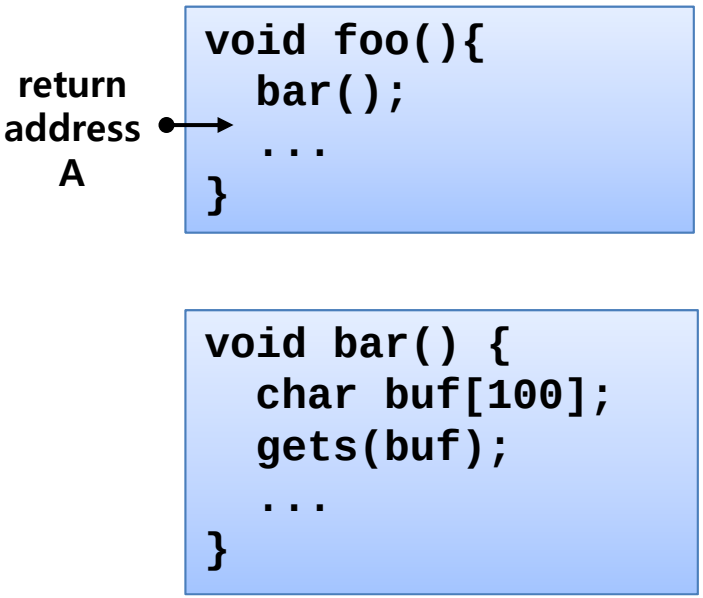
- As long as the user types at most seven characters, the string returned by **gets** (including NULL) will fit within the space allocated for **buf**
- A longer string will cause **gets** to overwrite some of the information stored on the stack

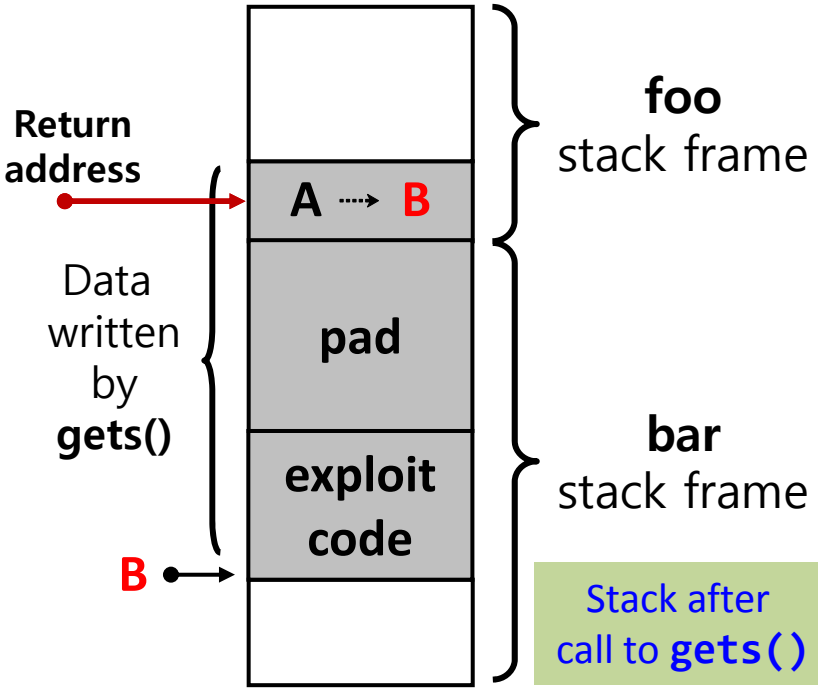
# characters typed	Additional corrupted state
0-7	None
8-23	Unused stack space
24-31	Return address
32+	Saved state in caller

Buffer Overflow

Malicious use of buffer overflow

- Input string contains byte representation of executable code
- Overwrite the return address with a pointer to the exploit code
- When **bar()** executes **ret**, it will jump to the exploit code





Buffer Overflow

Avoiding buffer overflows

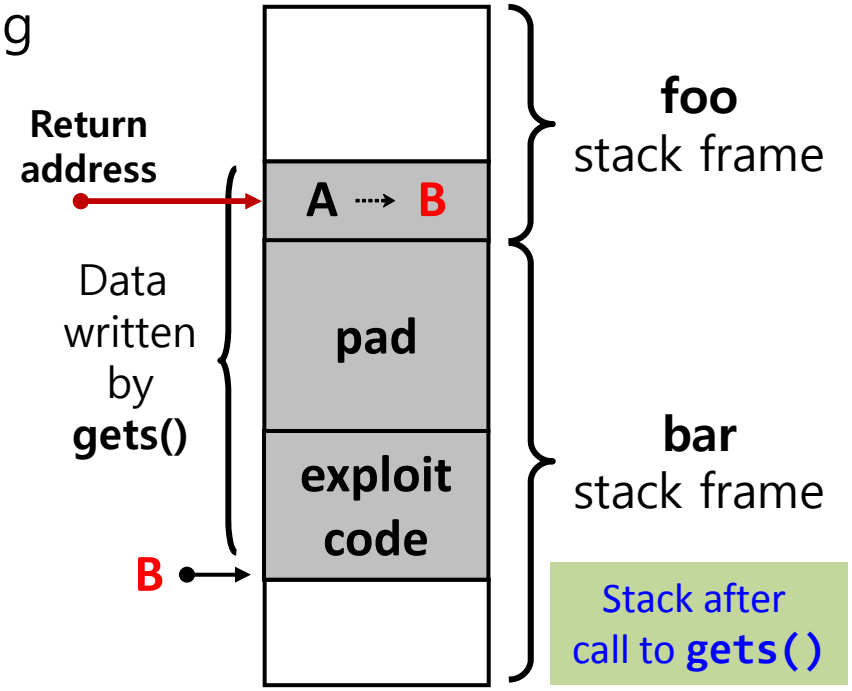
- Use library routines that limit string lengths
  - **fgets()** instead of **gets()**
    - ✓ Includes as an argument a count on the max # of bytes to read
  - **strncpy()** instead of **strcpy()**
  - Don't use **scanf()** with **%s** conversion specification
    - ✓ Use **fgets()** to read the string
    - ✓ Use **%ns** where **n** is a suitable integer

```
/* Echo Line */
void echo()
{
   char buf[4];  /* too small! */
   fgets(buf, 4, stdin);
   puts(buf);
}
```

Buffer Overflow

System-level protection

- Stack randomization
  - In order to insert exploit code into a system, the attacker needs to inject both the code as well as a pointer to the code as parts of the attack string



Buffer Overflow

- **System-level protection**
  - Stack randomization
    - Makes the position of the stack vary from one run of a program to another
      - ✓ Implemented by allocating a random amount of space on the stack at the start of a program
    - Makes it difficult for hacker to predict the stack address that can be used for the inserted code
    - Now standard practice in Linux systems

Buffer Overflow

System-level protection

- Stack randomization
  - Example) Guessing stack addresses
    - ✓ Running on Linux machine in 32-bit mode
      - Address range: **0xff7fc59c** ~ **0xffffd09c** (range of 2^{23})
    - ✓ Running on older Linux system
      - Address range: same address every time
    - ✓ Running in 64-bit mode
      - Address range: **0x7fff0001b698** ~ **0x7fffffffaa4a8** (range of 2^{32})

```
int main()
{
   long local;
   printf("local at %p\n", &local);
   return 0;
}
```

Buffer Overflow

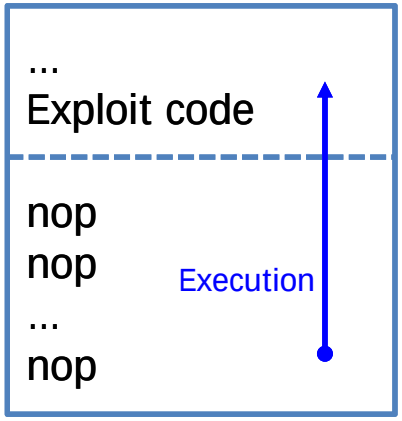
- **System-level protection**
  - ASLR (Address Space Layout Randomization)
    - Generalization of the stack randomization
    - Different parts of a program, including program code, library code, stack, data, and heap, are loaded into different regions of memory each time a program is run

Buffer Overflow

System-level protection

- ASLR (Address Space Layout Randomization)

[**nop sled**] A persistent attacker can overcome randomization
by brute force attacks (with a trick to include a long sequence of **nop**'s)



Buffer Overflow

System-level protection

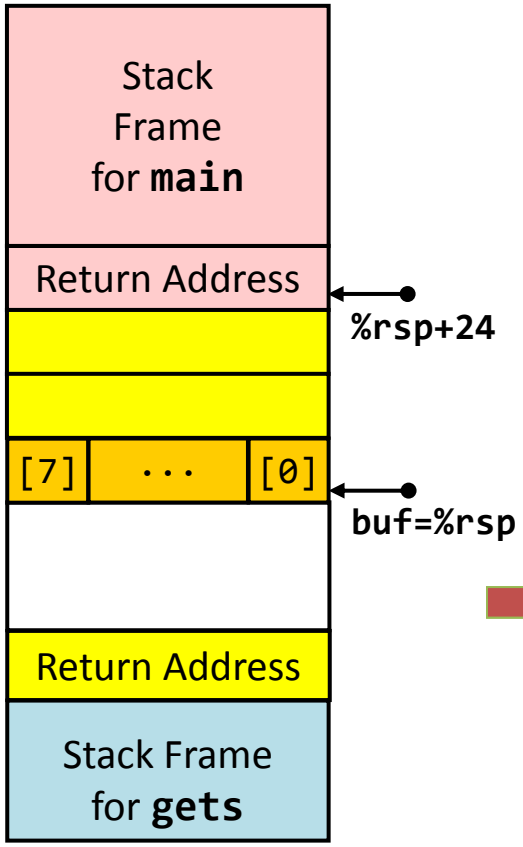
- Stack corruption detection
  - Detects when a stack is corrupted
    - ✓ **Stack protector** in recent versions of gcc (used automatically)
  - Stores a special **canary value** (**guard value**) in the stack frame between any local buffer and the rest of the stack state
    - ✓ Generated randomly each time a program is run
  - Checks if the canary value has been altered, before restoring the register state and returning from the function

  - **But, there are other ways to corrupt the state of an executing program**

Buffer Overflow

System-level protection

- Review: Vulnerable codes



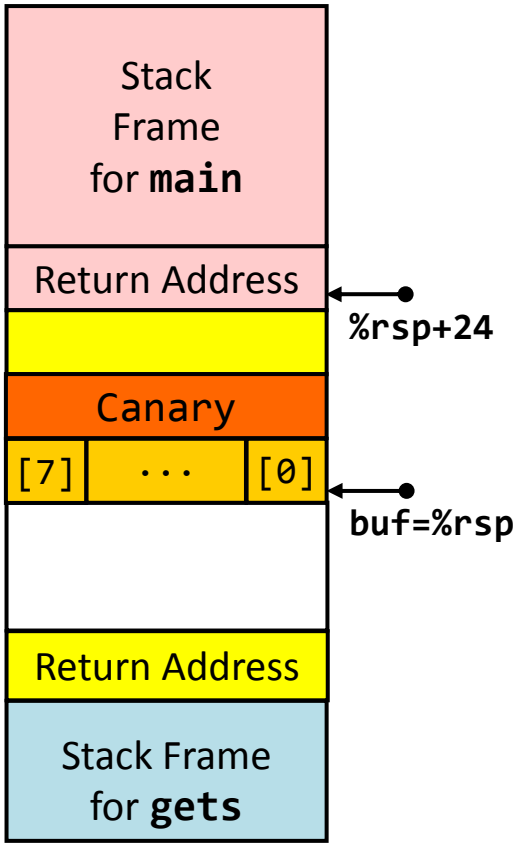

```
/* Echo Line */
void echo()
{
    char buf[8];        /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
    subq $24,%rsp       Allocates 24 bytes on stack
    movq %rsp,%rdi      Compute buf as %rsp
    call gets           Call gets
    movq %rsp,%rdi      Compute buf as %rsp
    call puts           Call gets
    addq $24,%rsp       Deallocate stack space
    ret                 Return
```

Buffer Overflow

System-level protection

- Stack corruption detection



- Segmented addressing
  - Storing the canary in a special read-only segment

```
echo:
    subq $24,%rsp          Allocate 24 bytes on stack
    movq %fs:40,%rax       Retrieve canary
    movq %rax,8(%rsp)      Store on stack
    xorl %eax,%eax         Zero out register
    movq %rsp,%rdi         Compute buf as %rsp
    call gets              Call gets
    movq %rsp,%rdi         Compute buf as %rsp
    call puts              Call puts
    movq 8(%rsp),%rax      Retrieve canary
    xorq %fs:40,%rax       Compare to stored value
    je .L9                 If =, goto ok
    call __fail            Stack corrupted!
.L9:                    ok:
    addq $24,%rsp          Deallocate stack space
    ret                    Return
```

Buffer Overflow

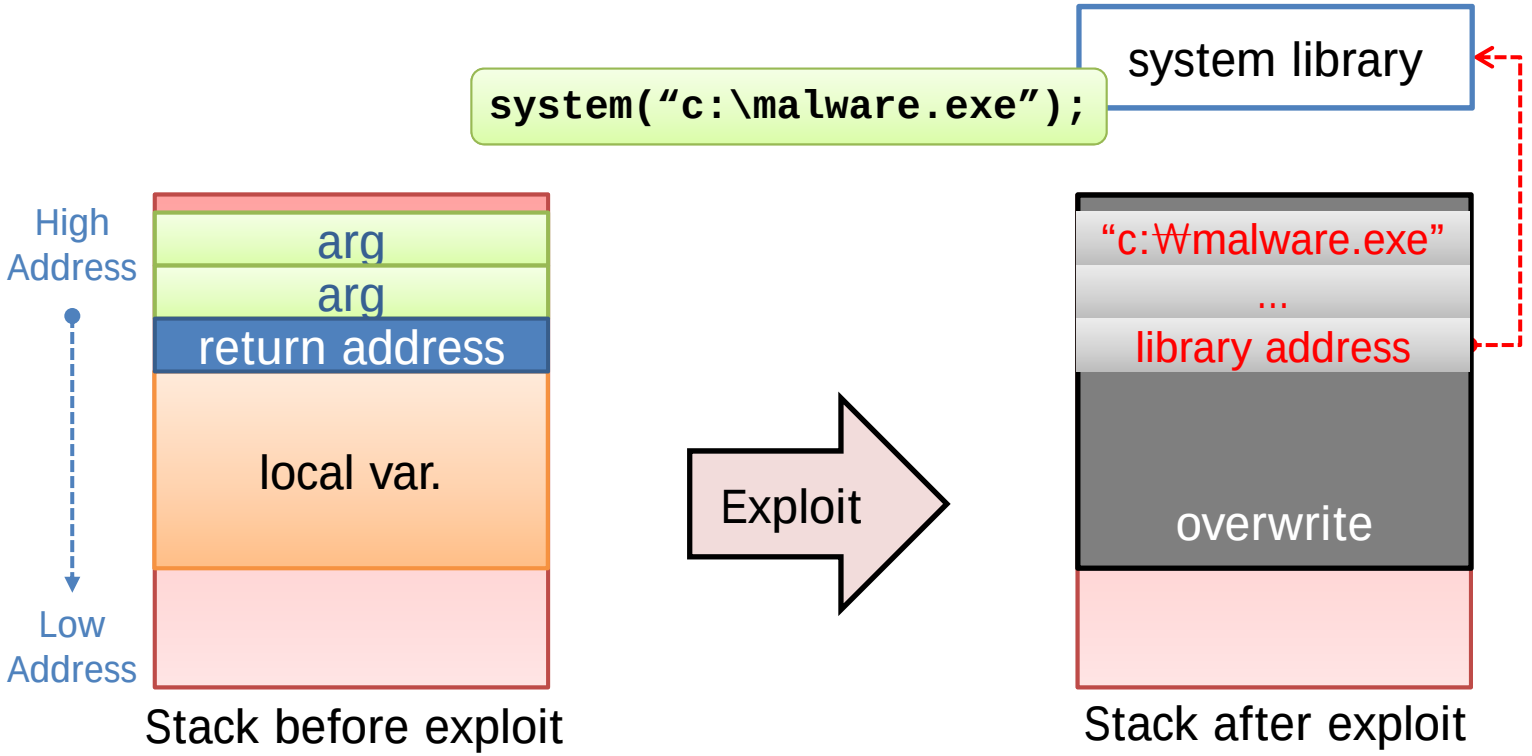
System-level protection

- DEP (Data Execution Prevention)
  - Typical access control (in most systems)
    - ✓ 3 types of accesses (**read**, **write**, **execute**)
  - Historically,
    - ✓ The x86 architecture merged the **read** and **execute** access controls into a single 1-bit flag
    - ✓ So the readable stack is also executable
  - Limits the stack pages to being readable but not executable (checking by hardware, which provides efficiency)
    - ✓ AMD NX (No-eXecute page protection)
    - ✓ Intel XD (eXecute-Disable bit)
    - ✓ ARM XN (eXecute-Never bit)

- **There are still other ways to attack computers !!!**

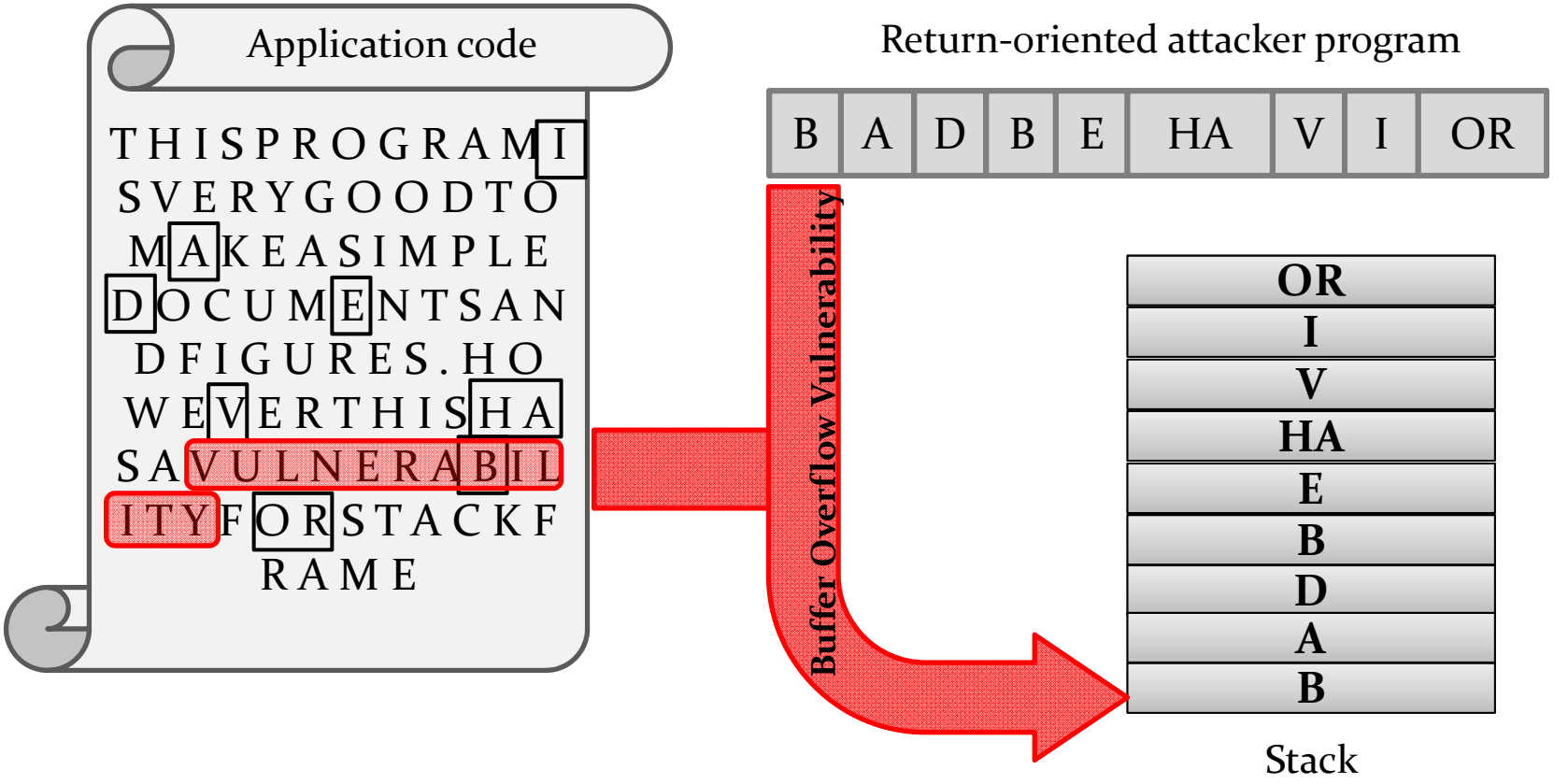
Buffer Overflow

- **Another techniques for stack smashing**
  - RTL (Return-to-Libc)



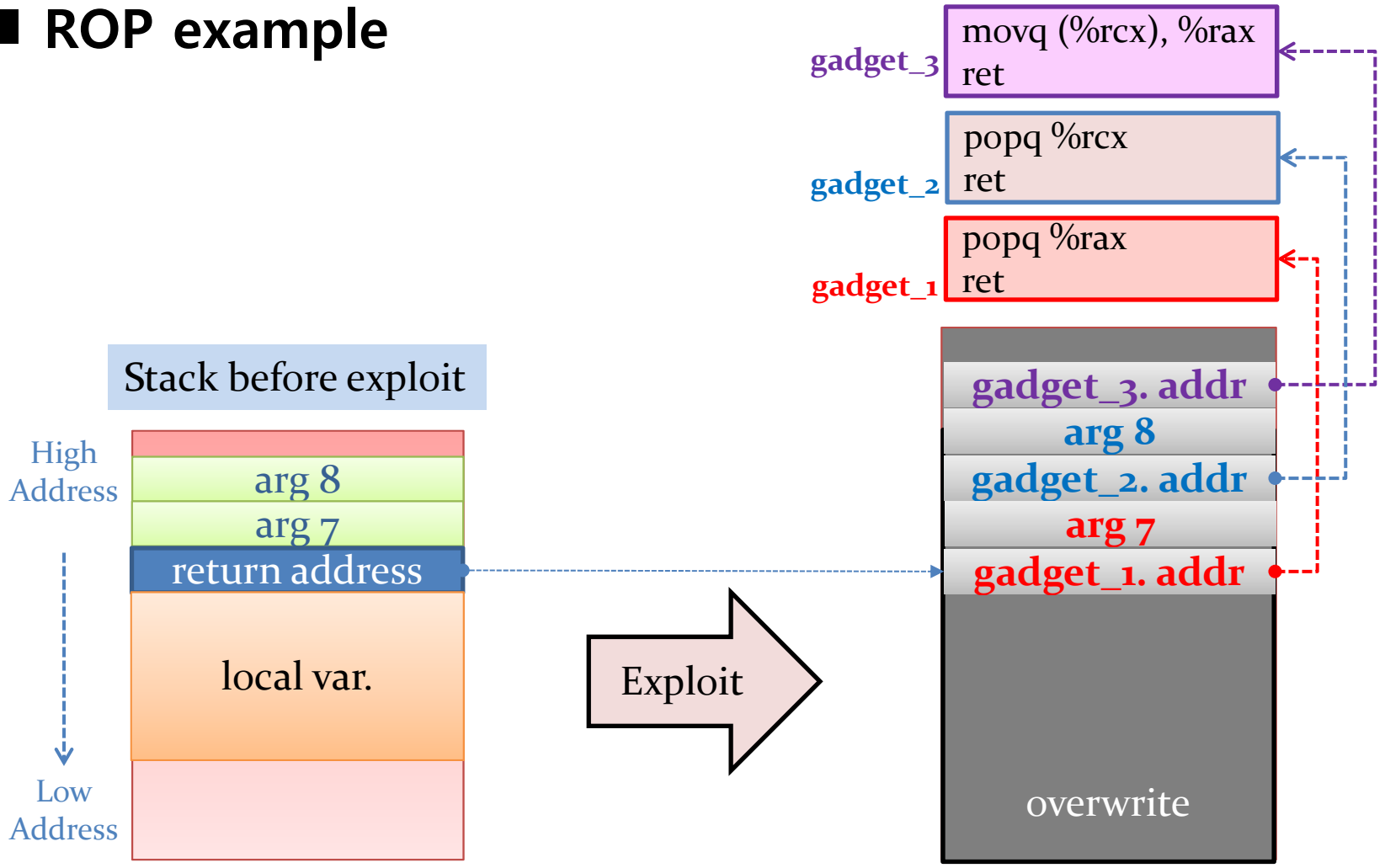
Buffer Overflow

- **Another techniques for stack smashing**
  - ROP (Return-Oriented Programming)



Buffer Overflow

- ROP example



Summary