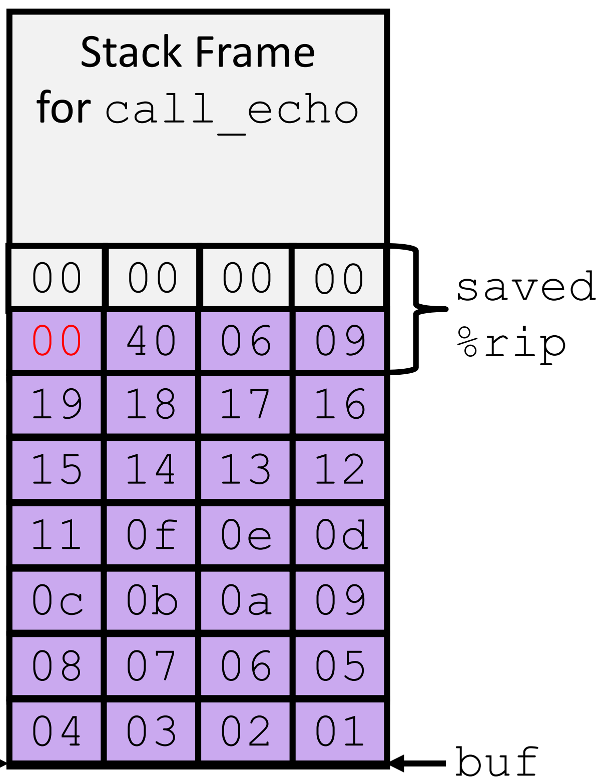
Lecture 10: Buffer Overflows (cont'd)

CS 105 Spring 2024

Review: Buffer Overflow Attack

- Idea: overwrite return address with address of instruction you want to execute next
  - If a string: use padding to fill up space between array and saved rip

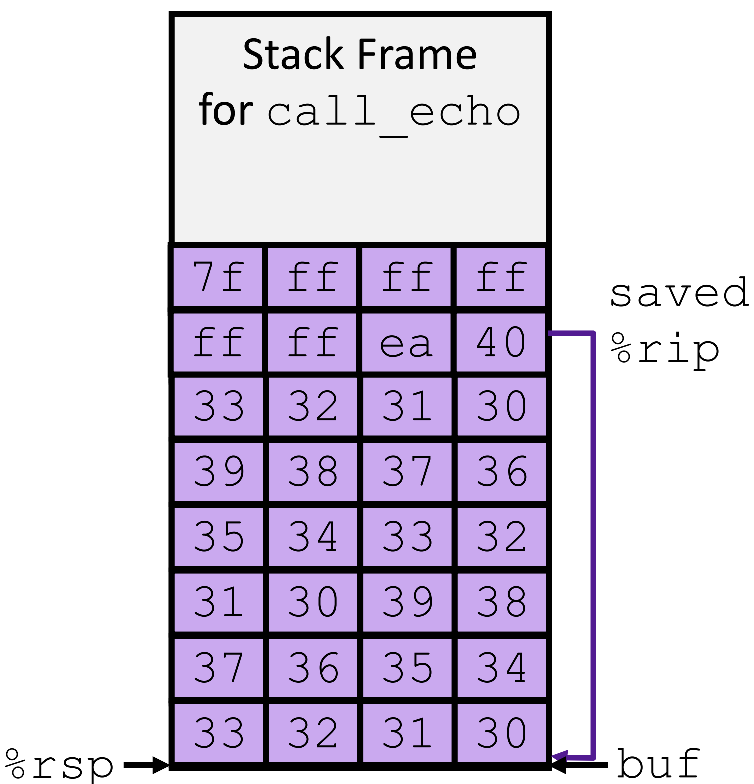



```
/* Echo Line */
void echo()
{
    char buf[4];
    gets(buf);
    puts(buf);
}
```

```
echo:
  subq  $0x18, %rsp
  movq  %rsp, %rdi
  call  gets
  call puts
  addq  $0x18, %rsp
  ret
```

Review: Stack Smashing

- Idea: fill the buffer with bytes that will be interpreted as code
- Overwrite the return address with address of the beginning of the buffer




```
/* Echo Line */
void echo()
{
    char buf[4];
    gets(buf);
    puts(buf);
}
```

```
echo:
  subq   $18, %rsp
  movq   %rsp, %rdi
  call   gets
  call  puts
  addq   $18, %rsp
  ret
```

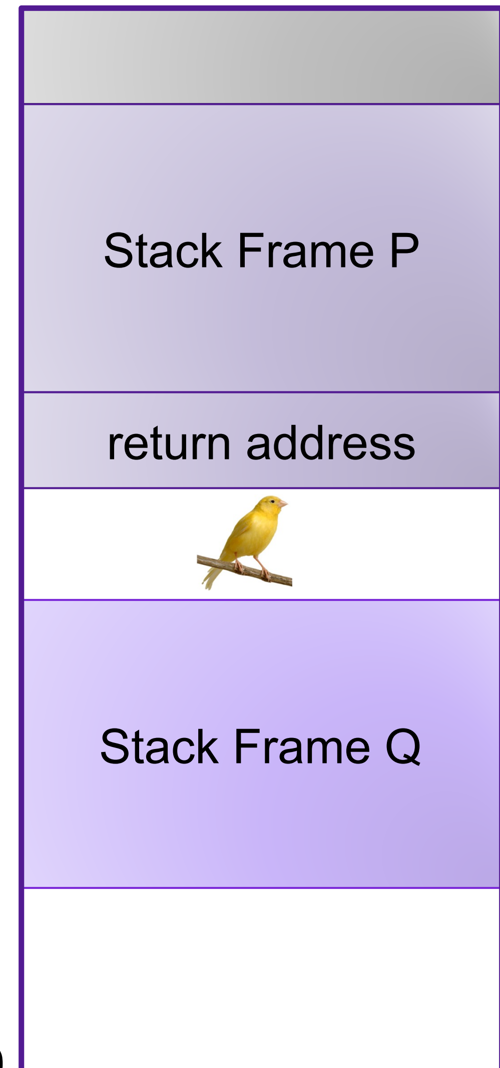
Defense #1: Bounds Checks

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

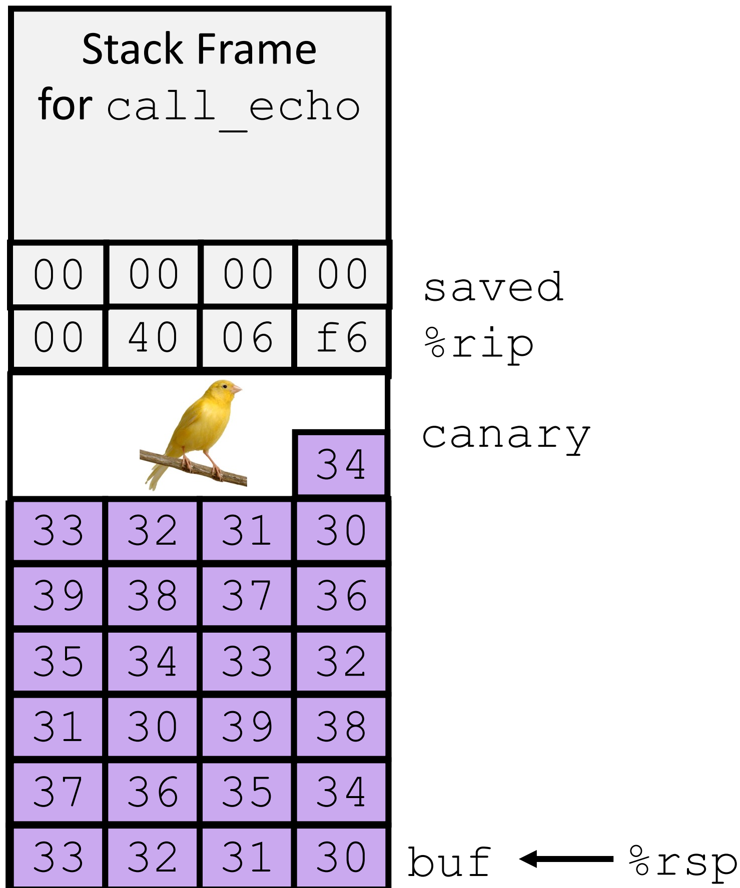
- For example, use library routines that limit string lengths
  - **`fgets`** instead of **`gets`**
  - **`strncpy`** instead of **`strcpy`**
  - Don't use **`scanf`** with **`%s`** conversion specification (use **`fgets`** to read the string or use **`%ns`** where **`n`** is a suitable integer)
- Or use a high-level language

Defense #2: Compiler checks

- Idea
  - Place special value ("canary") on stack just beyond buffer
  - Check for corruption before exiting function
- GCC Implementation
  - `-fstack-protector`
  - Now the default (disabled on Monday)

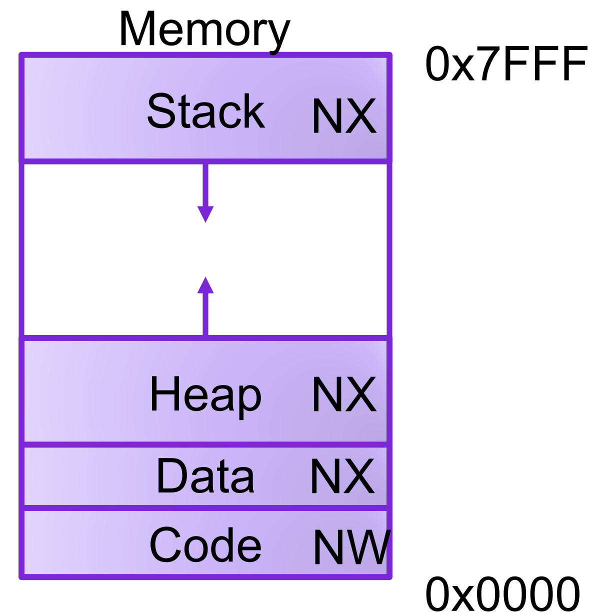


Stack Canaries




```
authenticate:
  pushq    %rbx
  subq     $16, %rsp
  movq     %rdi, %rbx
  movq     %fs:40, %rax
  movq     %rax, 8(%rsp)
  xorl     %eax, %eax
  movq     %rsp, %rdi
  call     gets
  movq     %rsp, %rsi
  movq     %rbx, %rdi
  call     strcmp
  testl    %eax, %eax
  sete     %al
  movq     8(%rsp), %rdx
  xorq     %fs:40, %rdx
  je       .L2
  call     __stack_chk_fail
.L2:
  movzbl   %al, %eax
  addq     $16, %rsp
  popq     %rbx
  ret
```

Defense #3: Memory Tagging



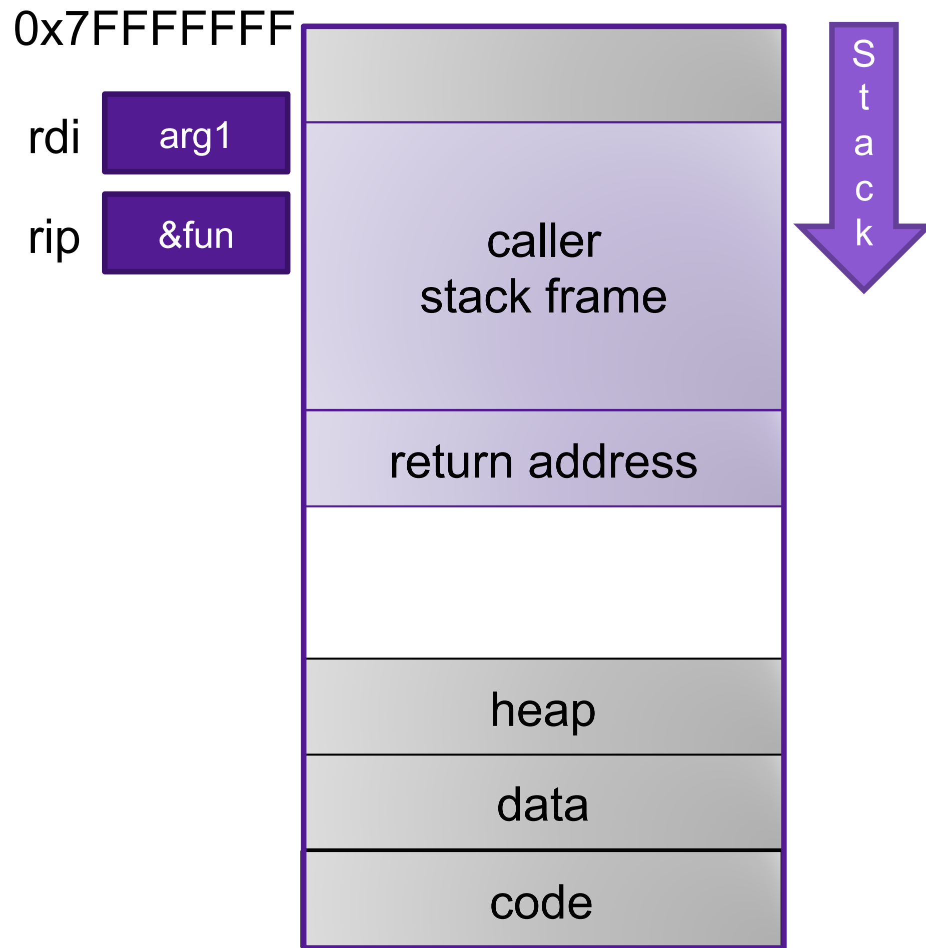
GCC Implementation

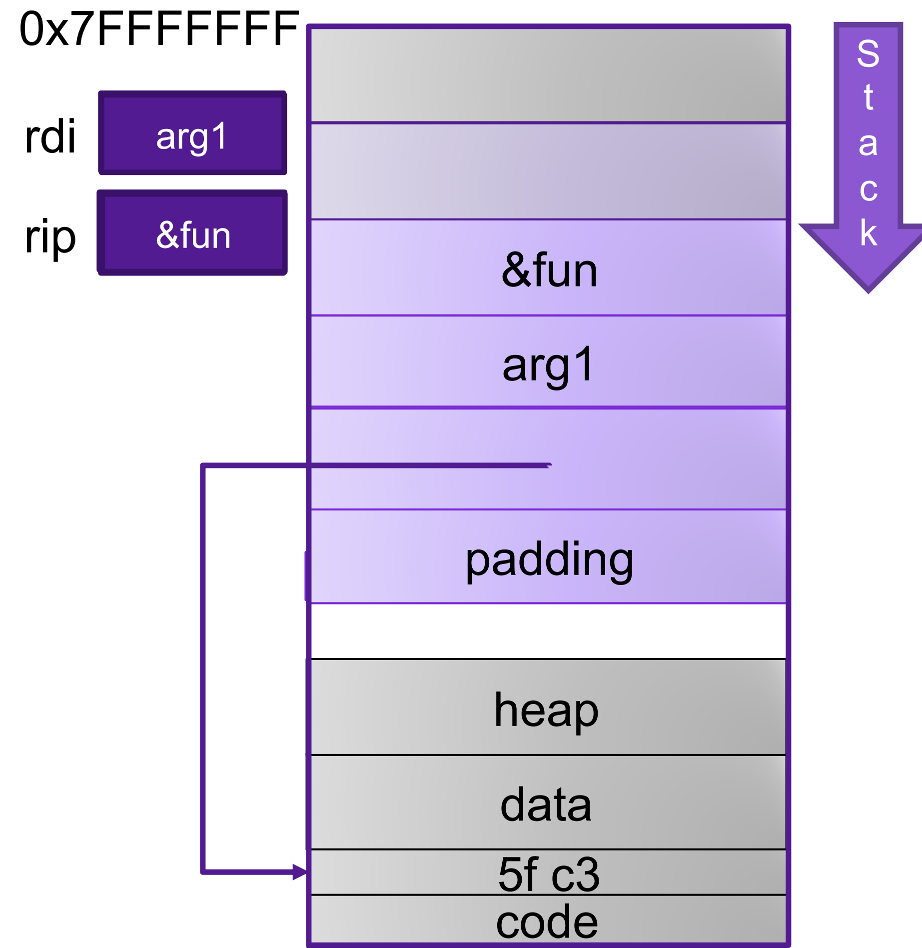
- Now the default
- Can disable with `-z execstack`

Code Reuse Attacks

- Key idea: execute instructions that already exist
- Defeats memory tagging defenses
- Examples:
  1. return to a function or line in the current program
  2. return to a library function (e.g., return-into-`libc`)
  3. return to some other instruction (return-oriented programming)

Handling Arguments





Return-into-libc

Sr.No.	Function & Description
1	**double atof(const char *str)** Converts the string pointed to, by the argument *str* to a floating-point number (type double).
2	**int atoi(const char *str)** Converts the string pointed to, by the argument *str* to an integer (type int).
3	**long int atol(const char *str)** Converts the string pointed to, by the argument *str* to a long integer (type long int).
8	**void free(void *ptr** Deallocates the memory previously allocated by a call to *calloc, malloc,* or *realloc*.
9	**void *malloc(size_t size)** Allocates the requested memory and returns a pointer to it.
10	**void *realloc(void *ptr, size_t size)** Attempts to resize the memory block pointed to by ptr that was previously allocated with a call to *malloc* or *calloc*.

15	**int system(const char *string)** The command specified by string is passed to the host environment to be executed by the command processor.
16	**void *bsearch(const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const void *, const void *))** Performs a binary search.
17	**void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))** Sorts an array.
18	**int abs(int x)** Returns the absolute value of x.
22	**int rand(void)** Returns a pseudo-random number in the range of 0 to *RAND_MAX*.
23	**void srand(unsigned int seed)** This function seeds the random number generator used by the function **rand**.

Properties of x86 Assembly

- lots of instructions
- variable length instructions
- not word aligned
- dense instruction set

Gadgets

```
void setval(unsigned* p) {
  *p = 3347663060u;
}
```

```
<setval>:
4004d9: c7 07 d4 48 89 c7   movl $0xc78948d4,(%rdi)
4004df: c3                  ret
```

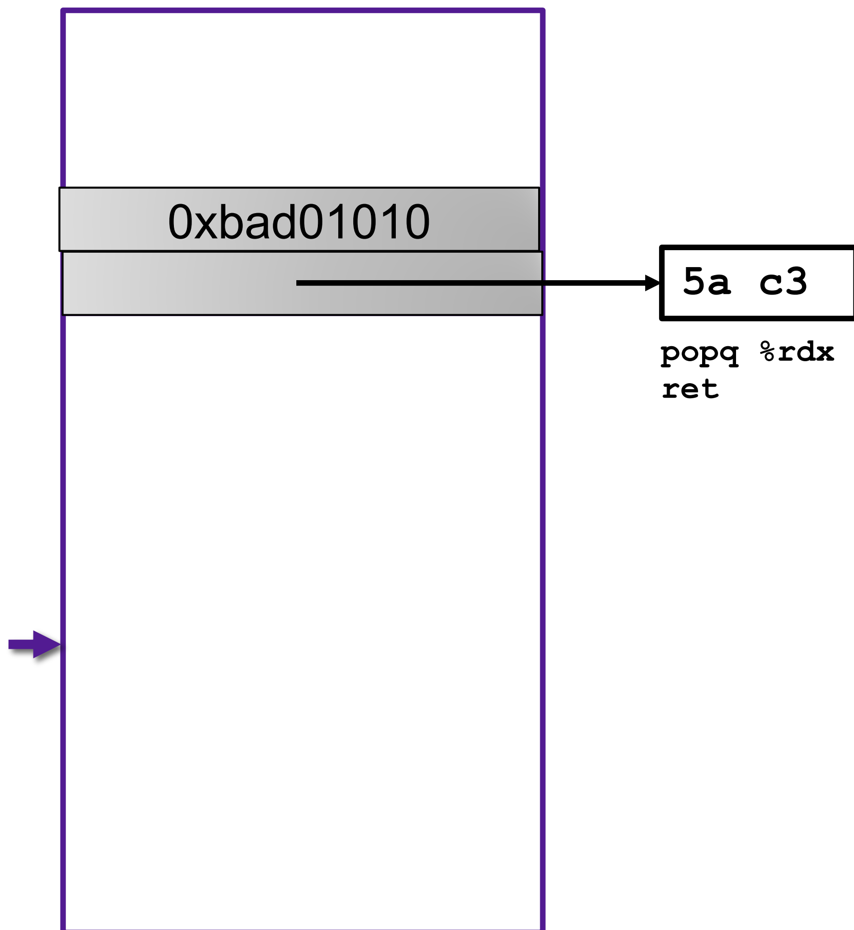
gadget address: `0x4004dc`

encodes:
```
movq %rax, %rdi
ret
```

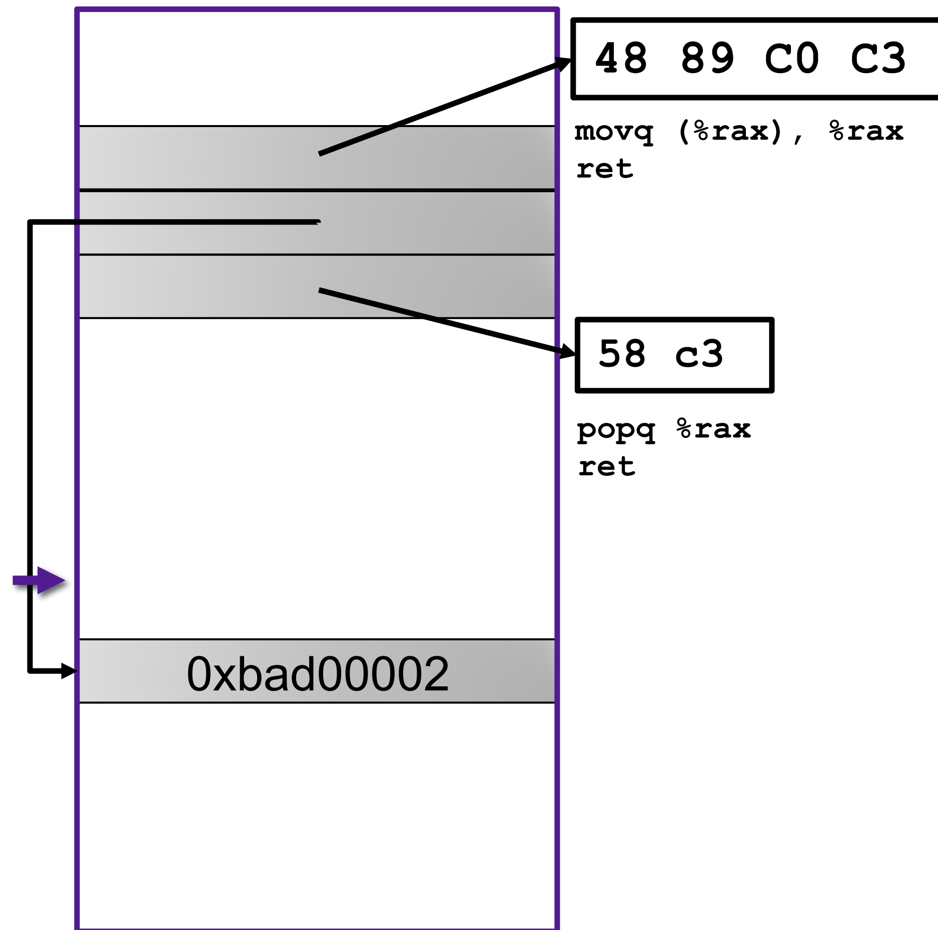
executes: `%rdi <- %rax`

Example Gadgets

Load Constant



Load from memory

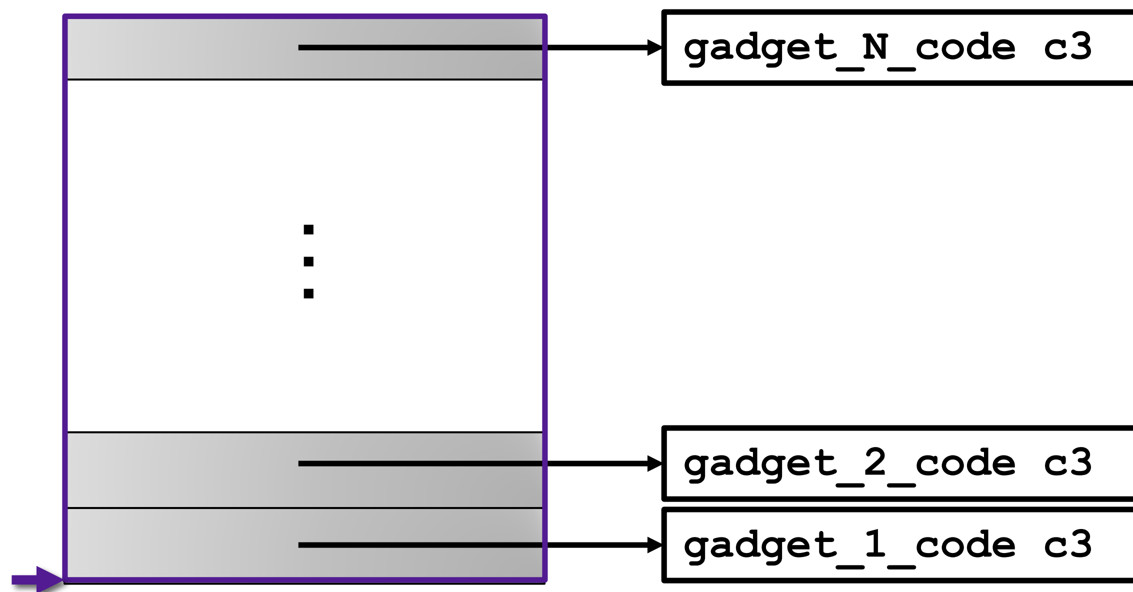


Return-oriented Programming

Return-oriented Programming

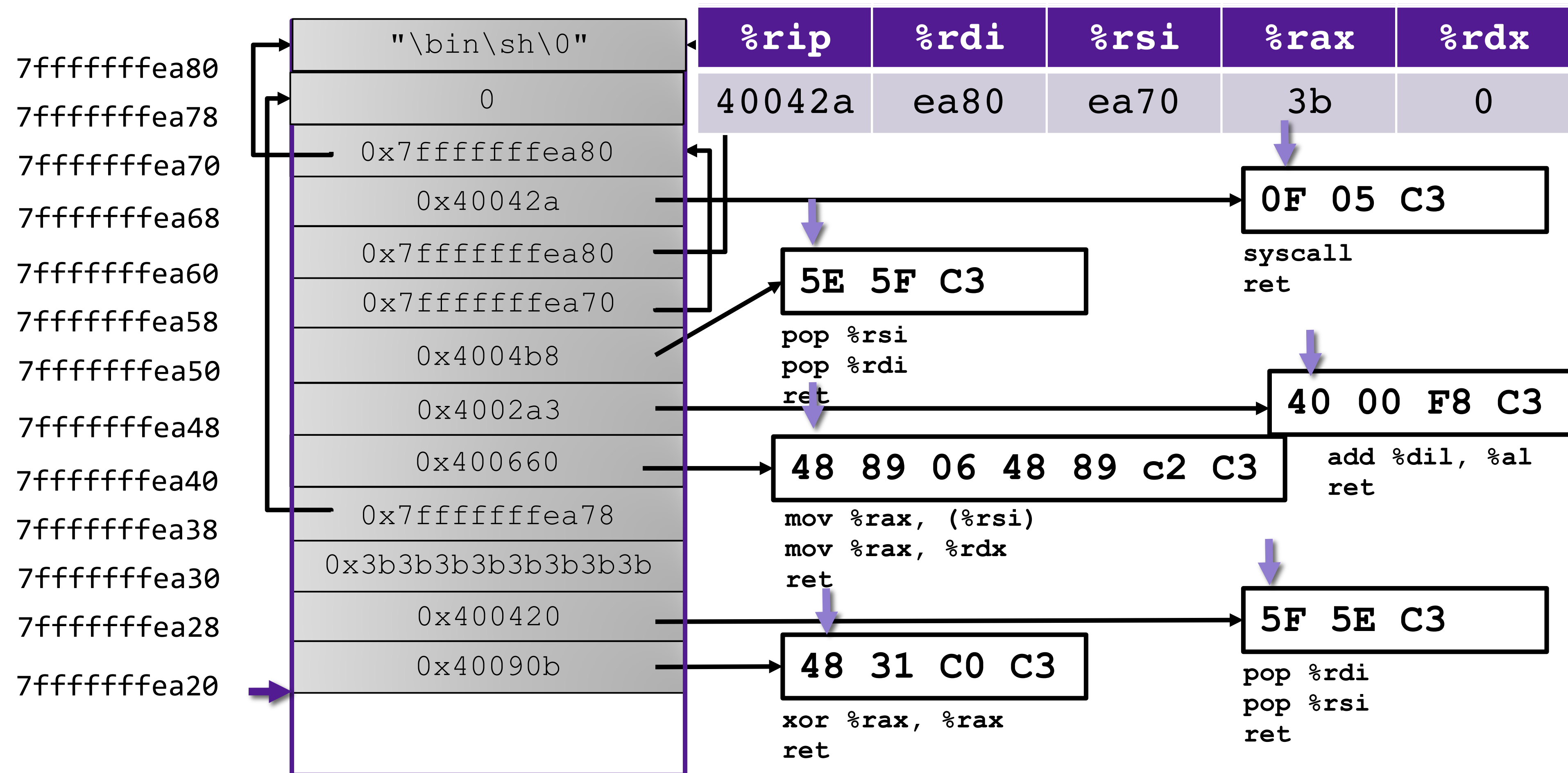
is a lot like a ransom note, but instead of cutting cut letters from magazines, you are cutting out instructions from .text segments

Return-oriented Programming



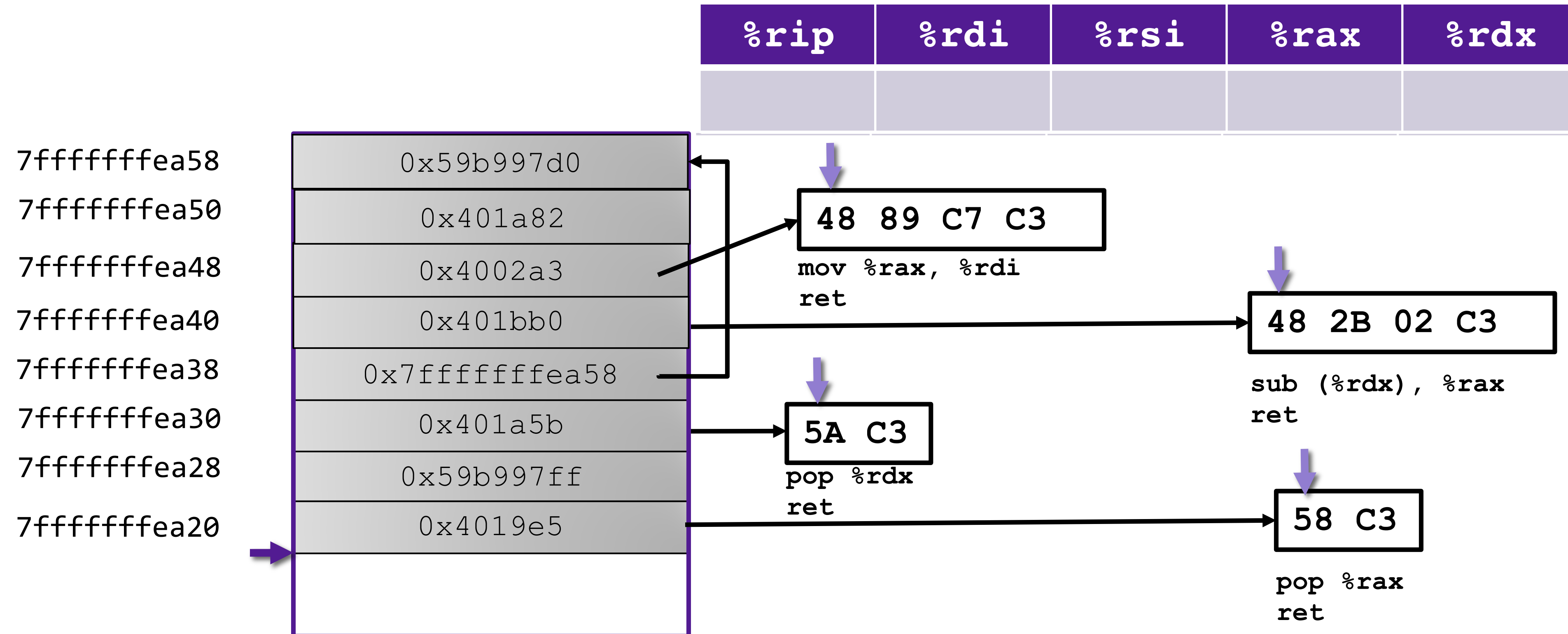
Final ret in each gadget sets pc (%rip) to beginning of next gadget code

Return-Oriented Shellcode



Exercise: ROP

- What are the values in the registers when the function at address 0x401a82 starts executing?

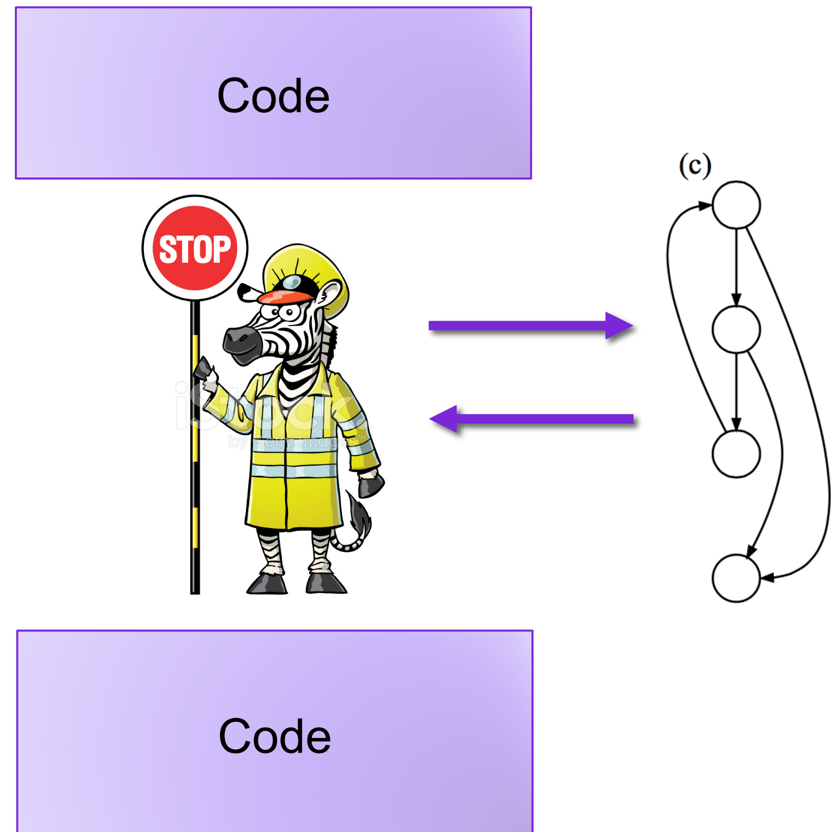


Defense #5: Address Space Layout Randomization

Other defenses

Gadget Elimination

Control Flow Integrity



The state of the world

Defenses:

- high-level languages
- Stack Canaries
- Memory tagging
- ASLR
- continuing research and development…

But all they aren't perfect!

The state of the world

2,500
2,000
1,500
1,000
500
0
2014 2015 2016 2017 2018 2019

Linux™

Canon