Lecture 9: Use and Abuse of the Stack (cont'd)

CS 105

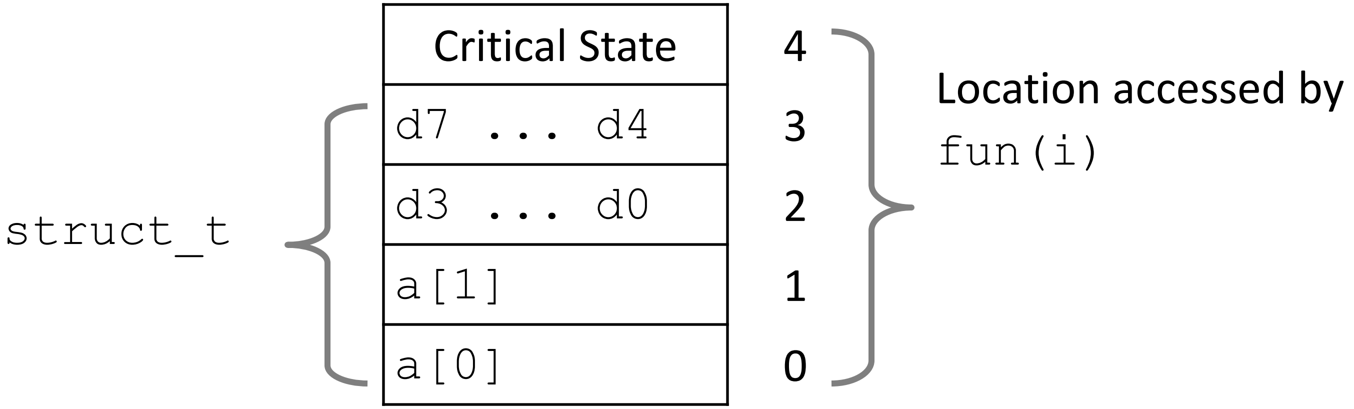
February 20, 2019

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →    3.14
fun(1)  →    3.14
fun(2)  →    3.140001
fun(3)  →    -2.000001
fun(4)  →    3.14
             Segmentation fault
```

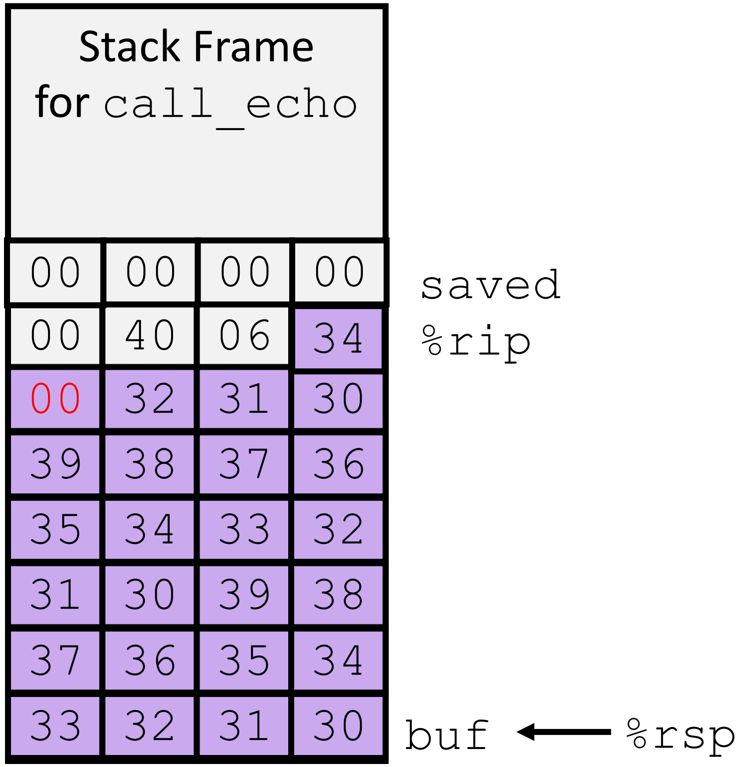
Explanation:



Critical State	4
d7 ... d4	3
d3 ... d0	2
a[1]	1
a[0]	0

Location accessed by `fun(i)`

Review: Buffer Overflow Stack



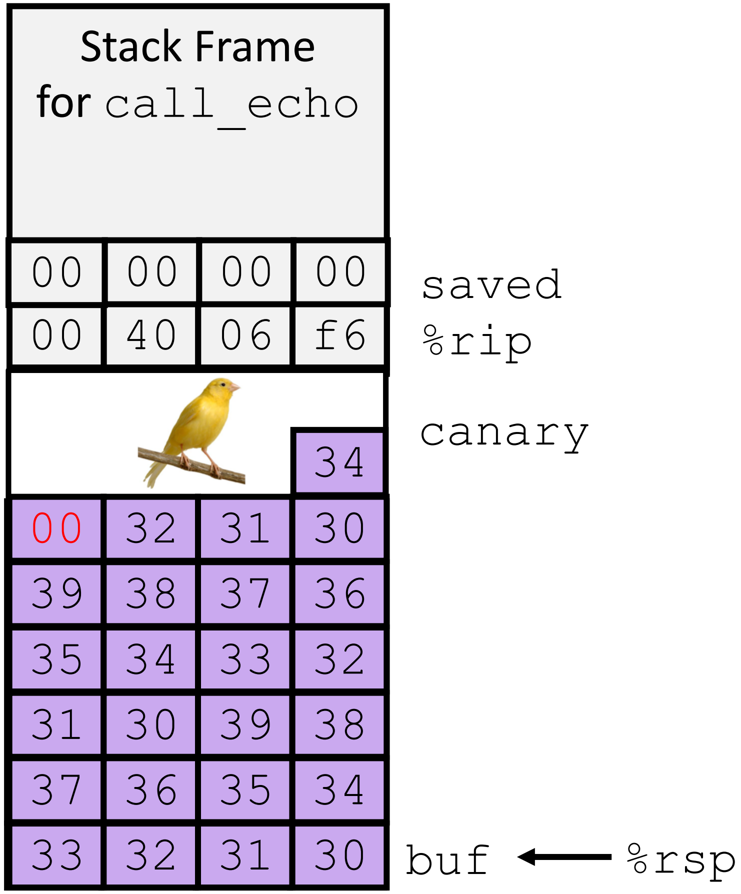
Stack Frame
for call_echo

00	00	00	00
00	40	06	34
00	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

```
/* 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
```

Review: Stack Canaries



Stack Frame for `call_echo`

00	00	00	00
00	40	06	f6

saved `%rip`

canary

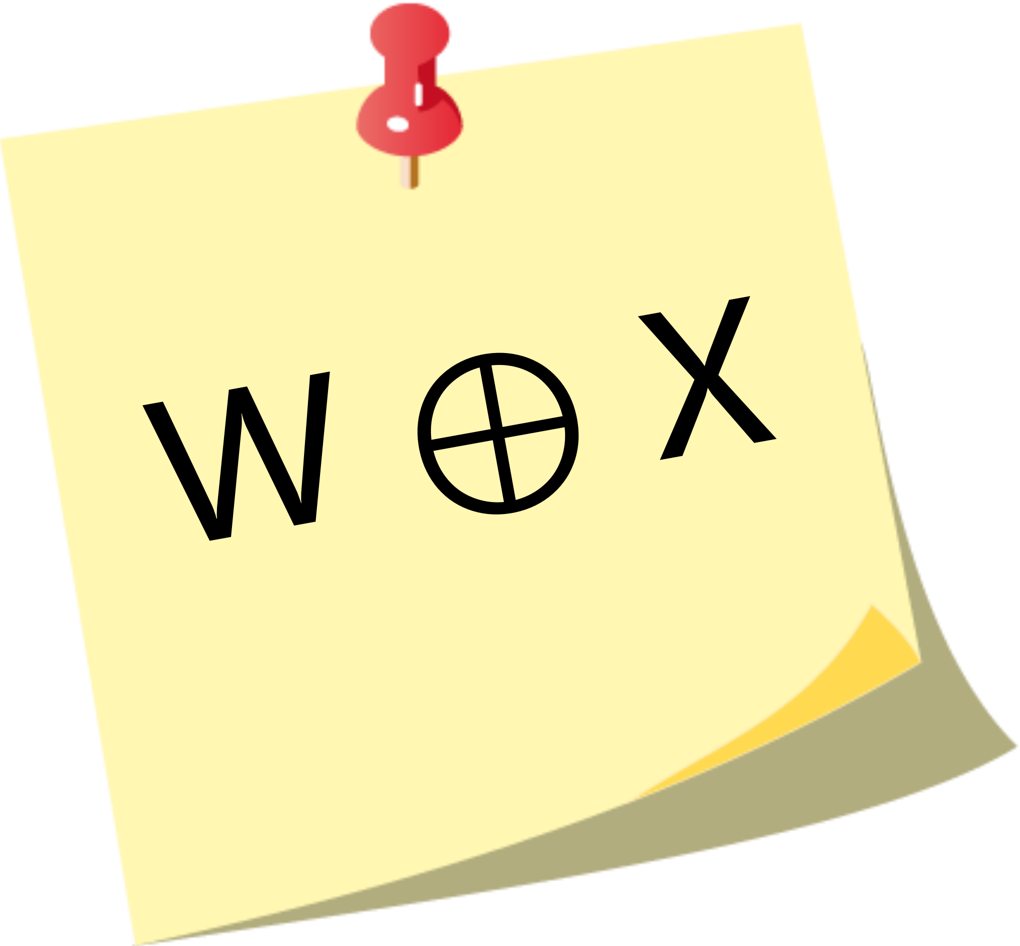
			34
00	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

`buf` ← `%rsp`

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

```
echo:
  subq  $24, %rsp
  movq  %rsp, %rdi
  call  gets
  call puts
  movq     24(%rsp), %rdx
  xorq     %fs:40, %rdx
  je       .L3
  call     __stack_chk_fail
.L3
  addq  $24, %rsp
  ret
```

Review: Memory Tagging



Code Reuse Attacks

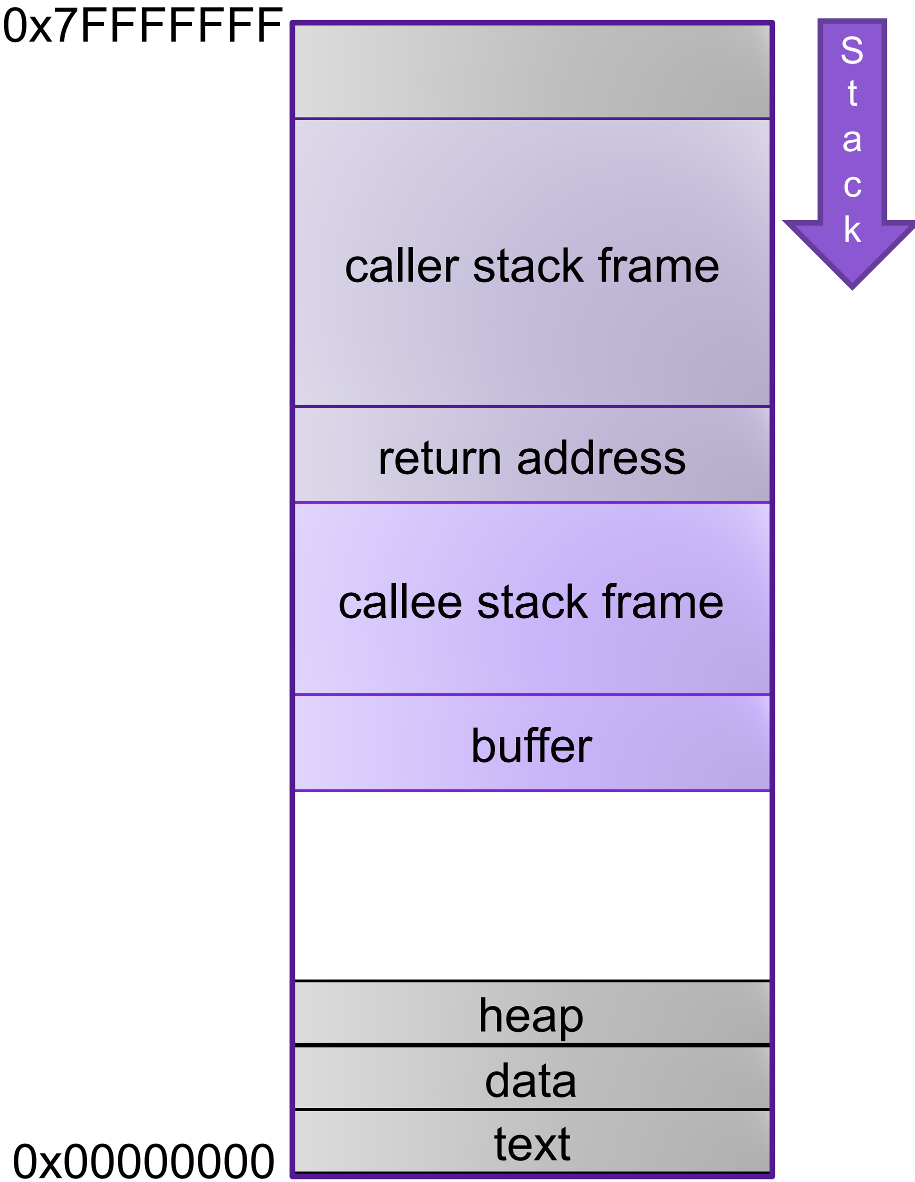
- Key idea: execute instructions that already exist

- Defeats memory tagging defenses

- Examples:
  1. return to a function in the current program
  2. return to a library function (e.g., return-into-`libc`)
  3. return to some other instruction (return-oriented programming)

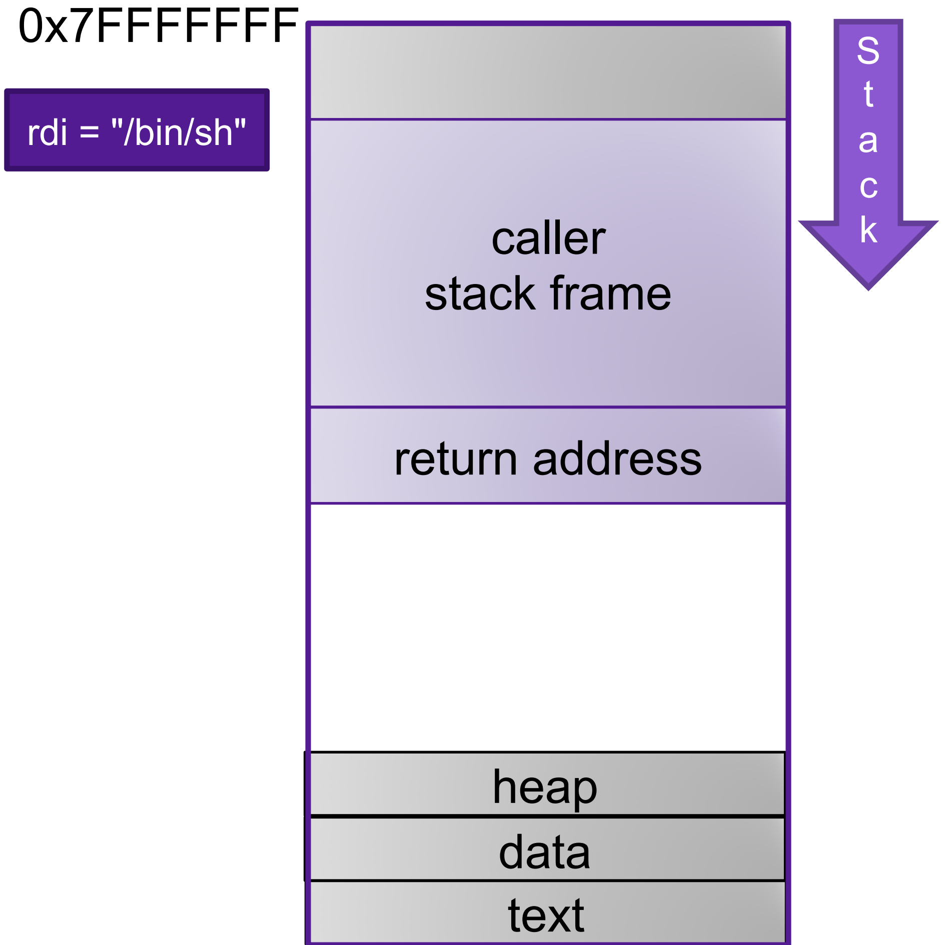
Returning to a function

- Overwrite the saved return address with the location of a function in the current program



Handling Arguments

what function expects
when it is called…



0x7FFFFFFF

rdi = "/bin/sh"

caller
stack frame

return address

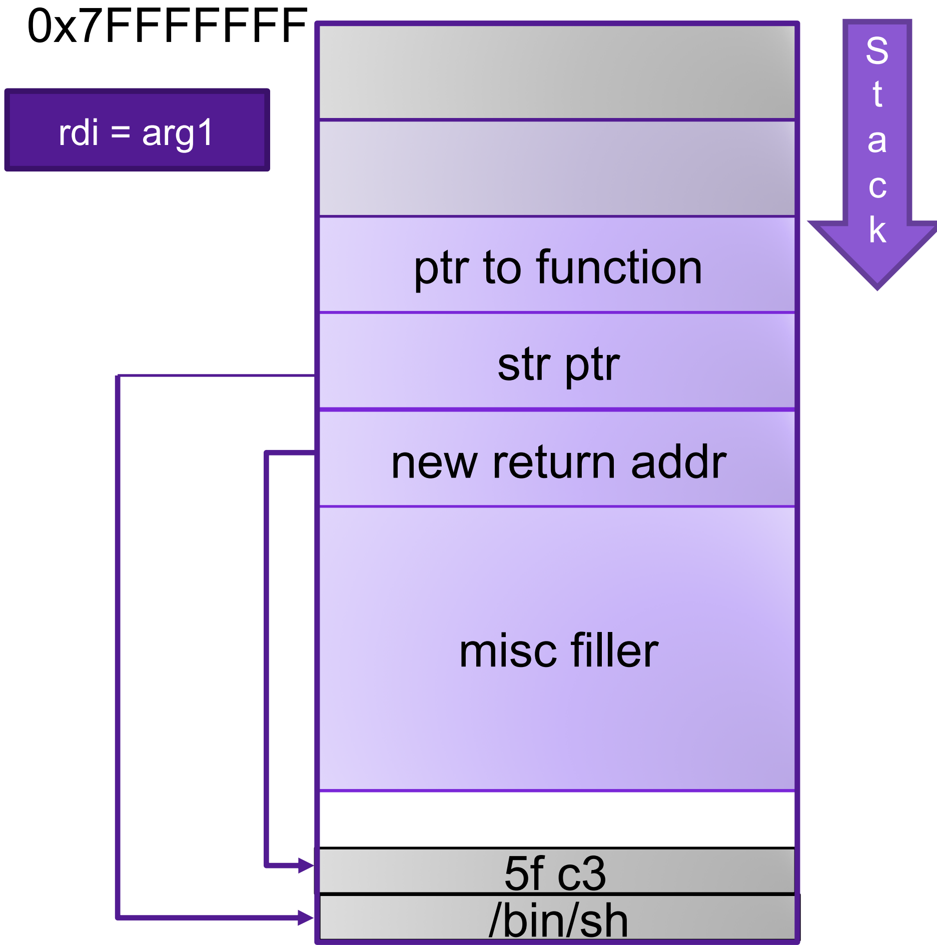
heap

data

text

Stack

overflow with argument



0x7FFFFFFF

rdi = arg1

ptr to function

str ptr

new return addr

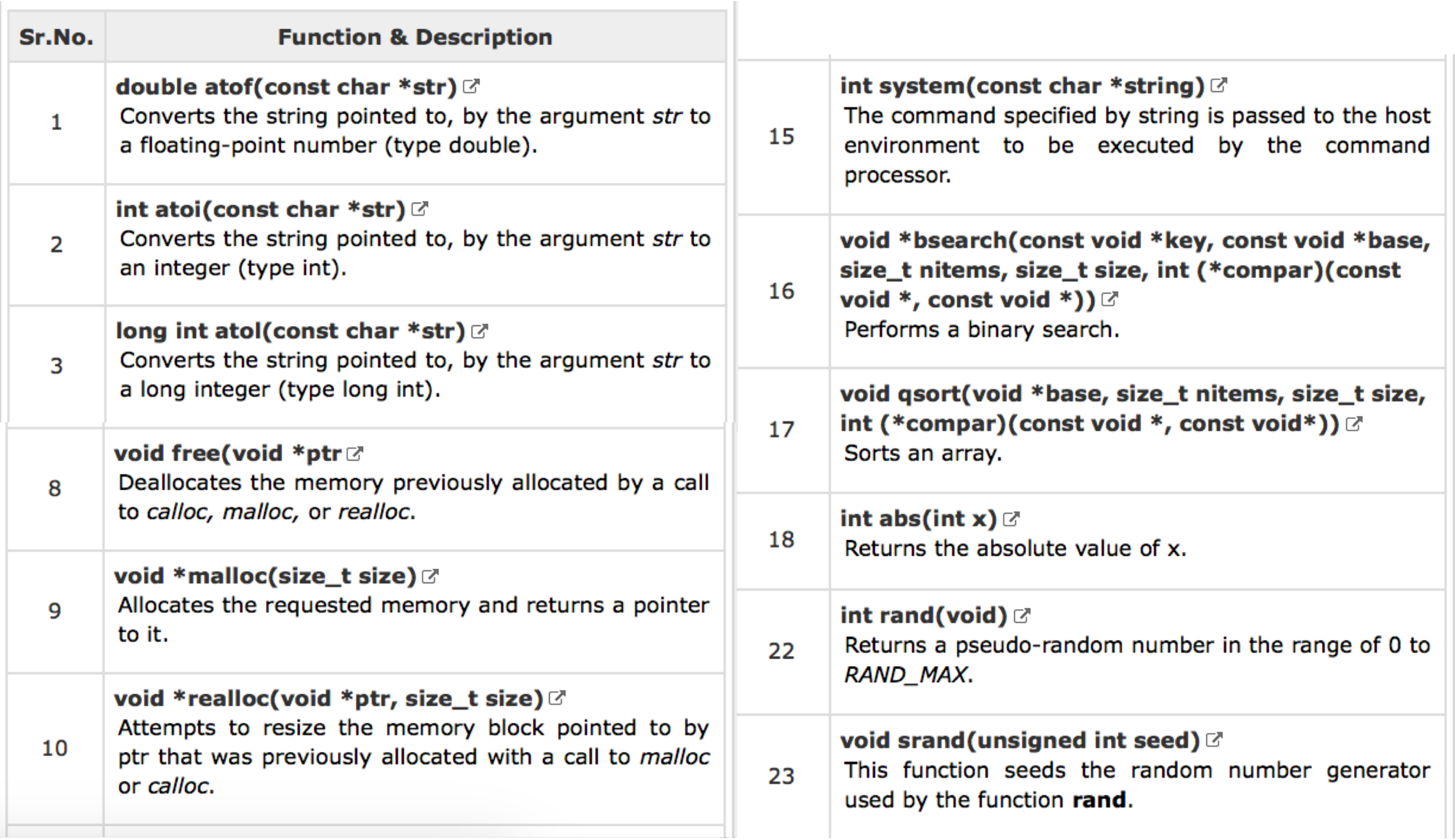
misc filler

5f c3

/bin/sh

Stack

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 x86-64

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

Return Oriented Programming

```
f7 c7 07 00 00 00        test $0x00000007, %edi
0f 95 45 c3              setnzb -61 (%ebp)
```

```
c7 07 00 00 00 0f        movl $0x0f000000, (%edi)
95                       xchg %ebp, %eax
45                       inc %ebp
c3                       ret
```

Gadgets

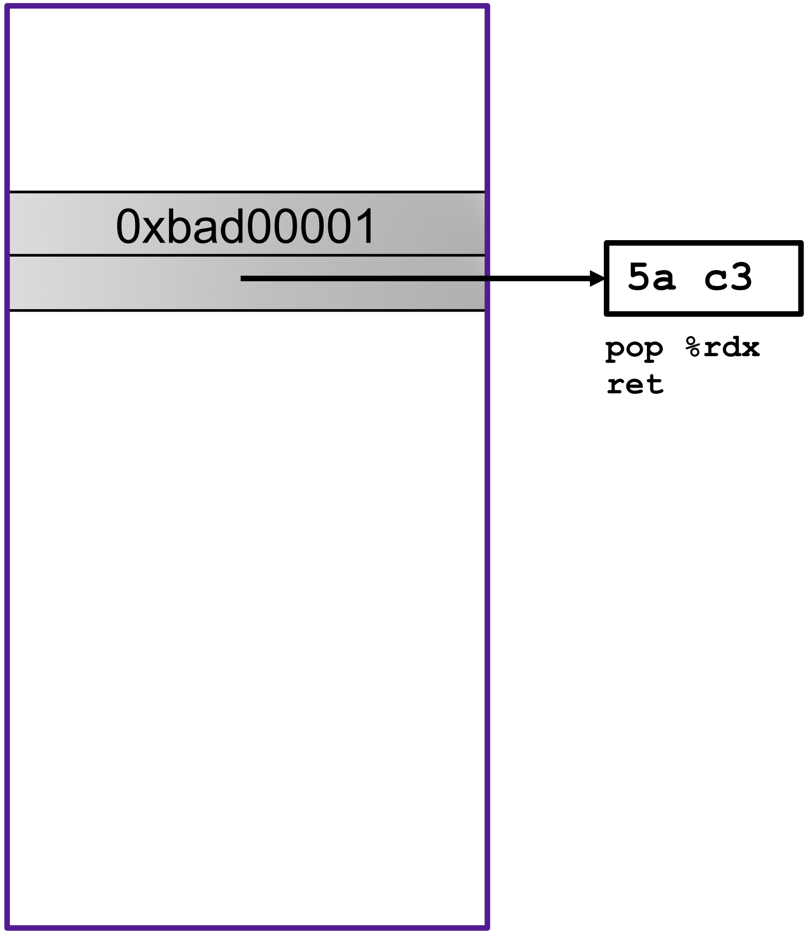
```
void setval(unsigned *p) {
      *p = 3347663060u;
}
```

```
<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

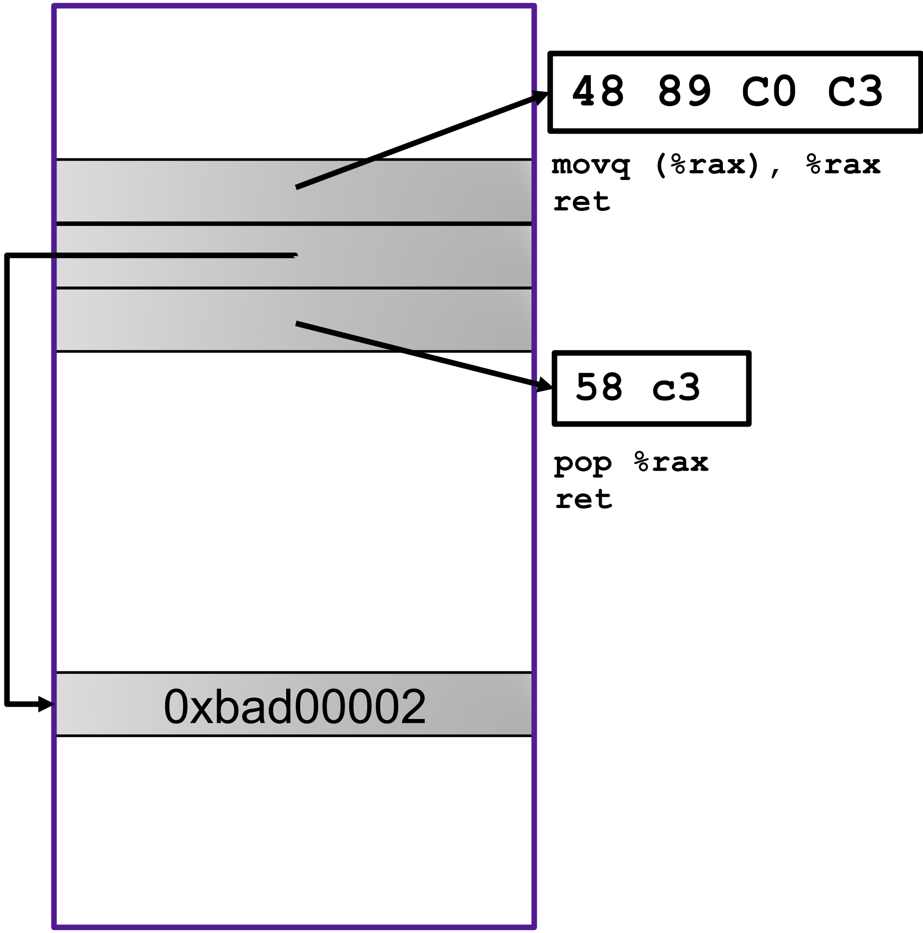


0xbad00001

5a c3

```
pop %rdx
ret
```

Load from memory



48 89 C0 C3

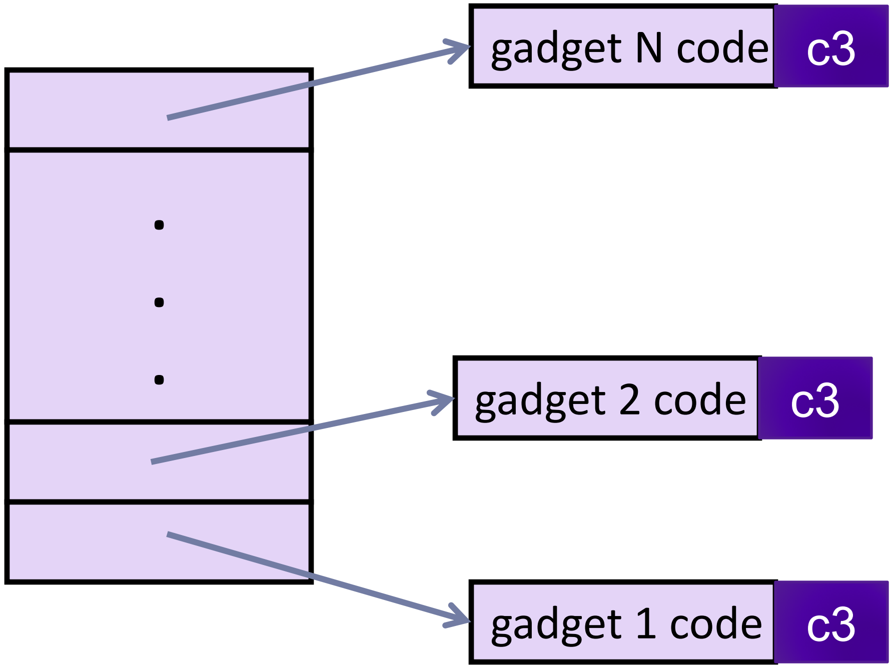
```
movq (%rax), %rax
ret
```

58 c3

```
pop %rax
ret
```

0xbad00002

Return-oriented programming attack



- Final ret in each gadget will start next one

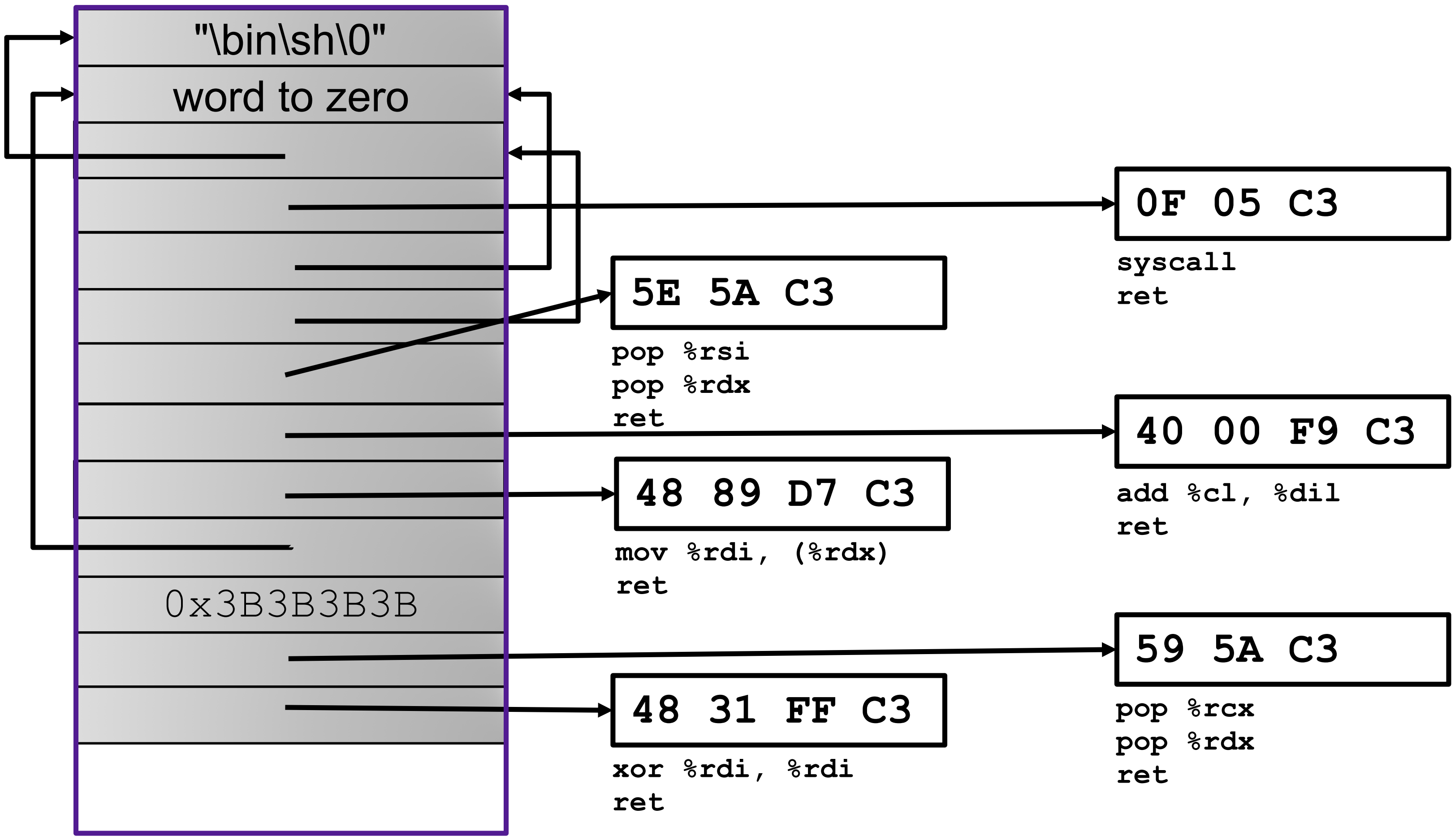
Return Oriented Programming

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

Image By: Dino Dai Zovi

Return-Oriented Shellcode



Address Space Layout Randomization

The state of the world

Defenses:

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

But all they aren't perfect!