Lecture 3/4 – Control Flow Hijacking

University of Illinois

ECE 422/CS 461

Control Flow Hijacking

- Altering control flow of a target program to cause it to do what attacker wants
  - Identify a **value** that will be loaded into **PC** (%eip)
  - Overwrite it (e.g., to point to shellcode)
- The simple stack buffer overflow attack from last lecture overwrote return address on stack

Goals

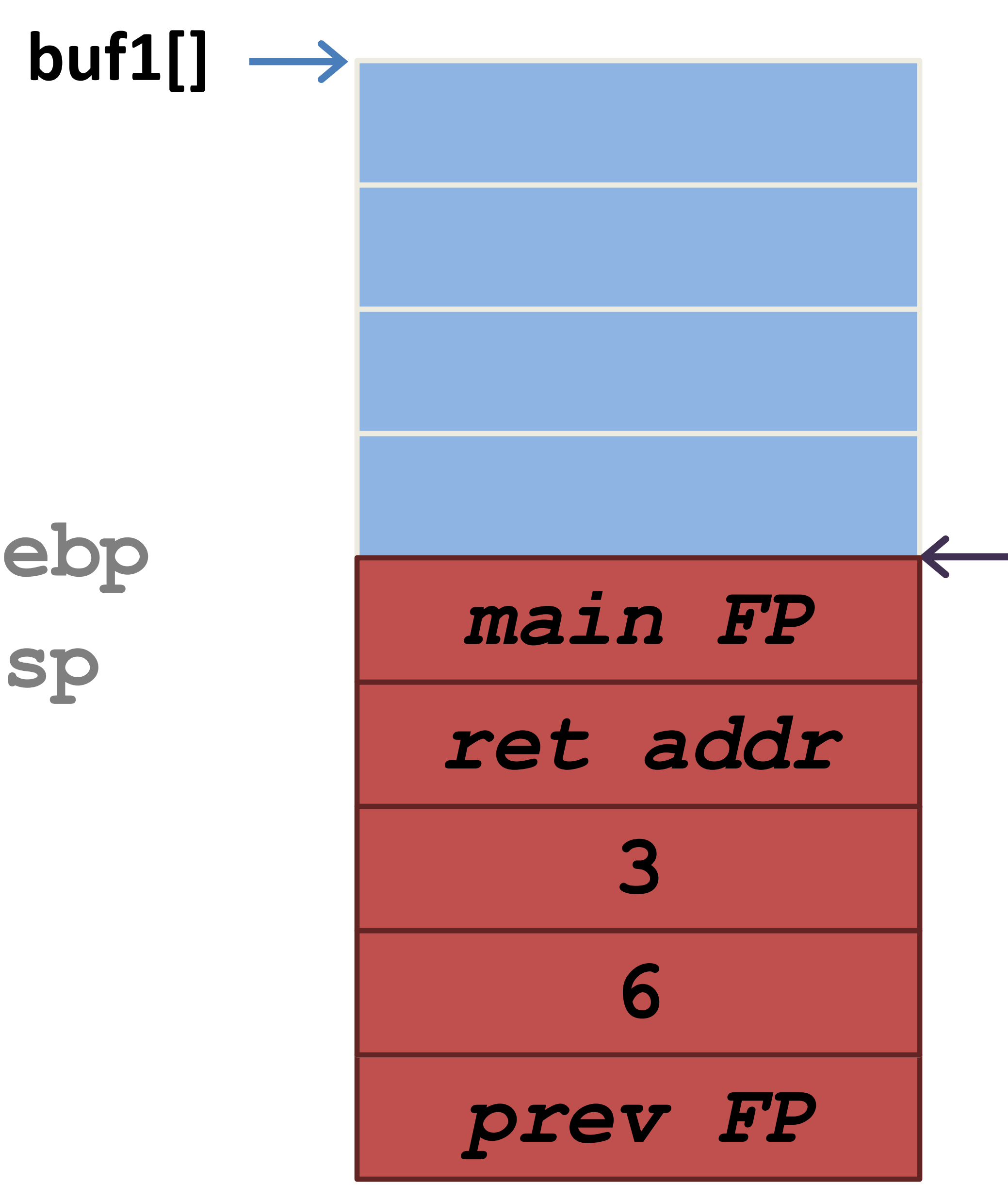
- By the end of this lecture you should:
  - Understand common vulnerabilities that lead to control flow hijacking
  - Understand common countermeasures to control flow hijacking and their limitations
  - Understand which countermeasure an (advanced) attack bypasses

Defenses and Counter-Attacks

- Stack canaries
- Other forms of control flow hijacking
- Data Execution Prevention (DEP, W^X)
- Return-to-libc and Return-Oriented Programming (ROP)
- Address Space Layout Randomization (ASLR)
- Heap Spray

```
void foo(int a, int b) {
    char buf1[16];
    gets(buf1);
}
```

```
foo:
   push    %ebp
   mov     %esp, %ebp
   sub     $16, %esp
   ……
   call    gets
   leave
   ret
```



buf1[]

main FP

ret addr

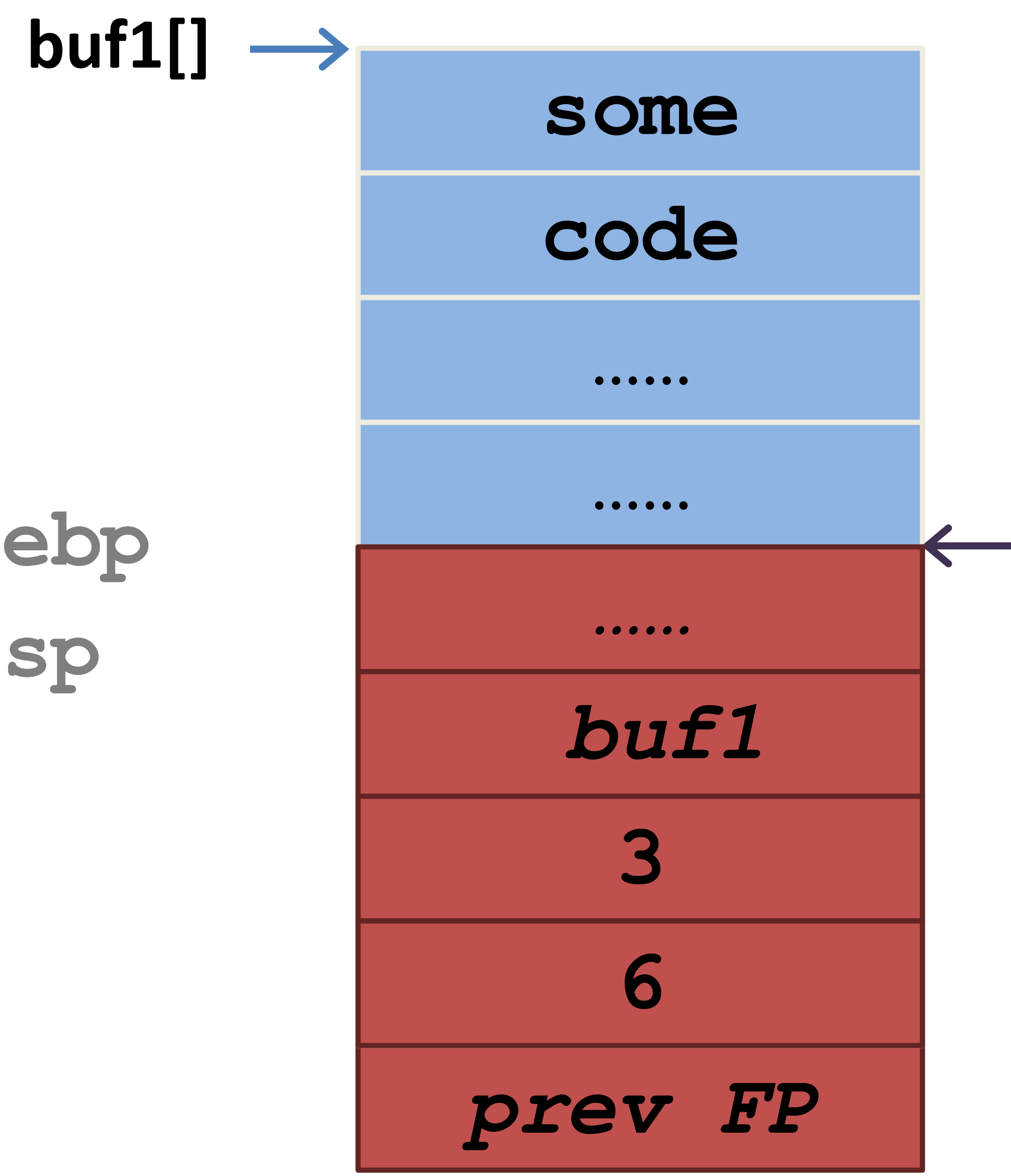
3

6

prev FP

```
void foo(int a, int b) {
    char buf1[16];
    gets(buf1);
}
```

```
foo:
  push   %ebp
  mov    %esp, %ebp
  sub    $16, %esp
  ......
  call   gets
  leave
  ret
```



buf1[]

some

code

......

......

......

buf1

3

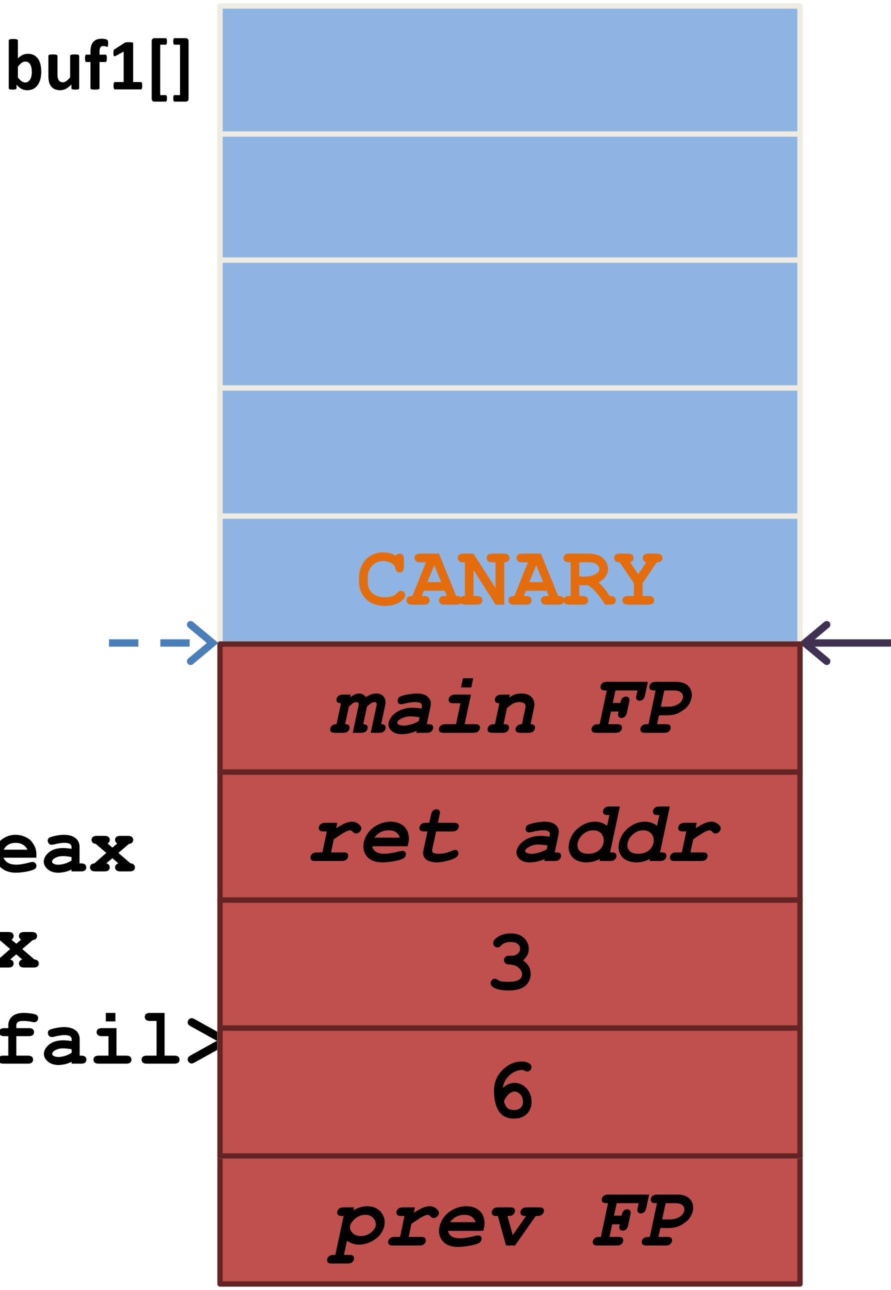
6

prev FP

Stack Canary

- **Idea:** detect return address overwrite
- Place special value (canary) before return address on the stack
- Check canary before executing `ret`
  - If return address is overwritten, so is canary

```
foo:
    push    %ebp
    mov     %esp, %ebp
    push    CANARY
    sub     $16, %esp
    ......
    call    gets
    mov     -4(%ebp), %eax
    cmp     CANARY, %eax
    jne     <stack_chk_fail>
    leave
    ret
```



buf1[]

CANARY

main FP

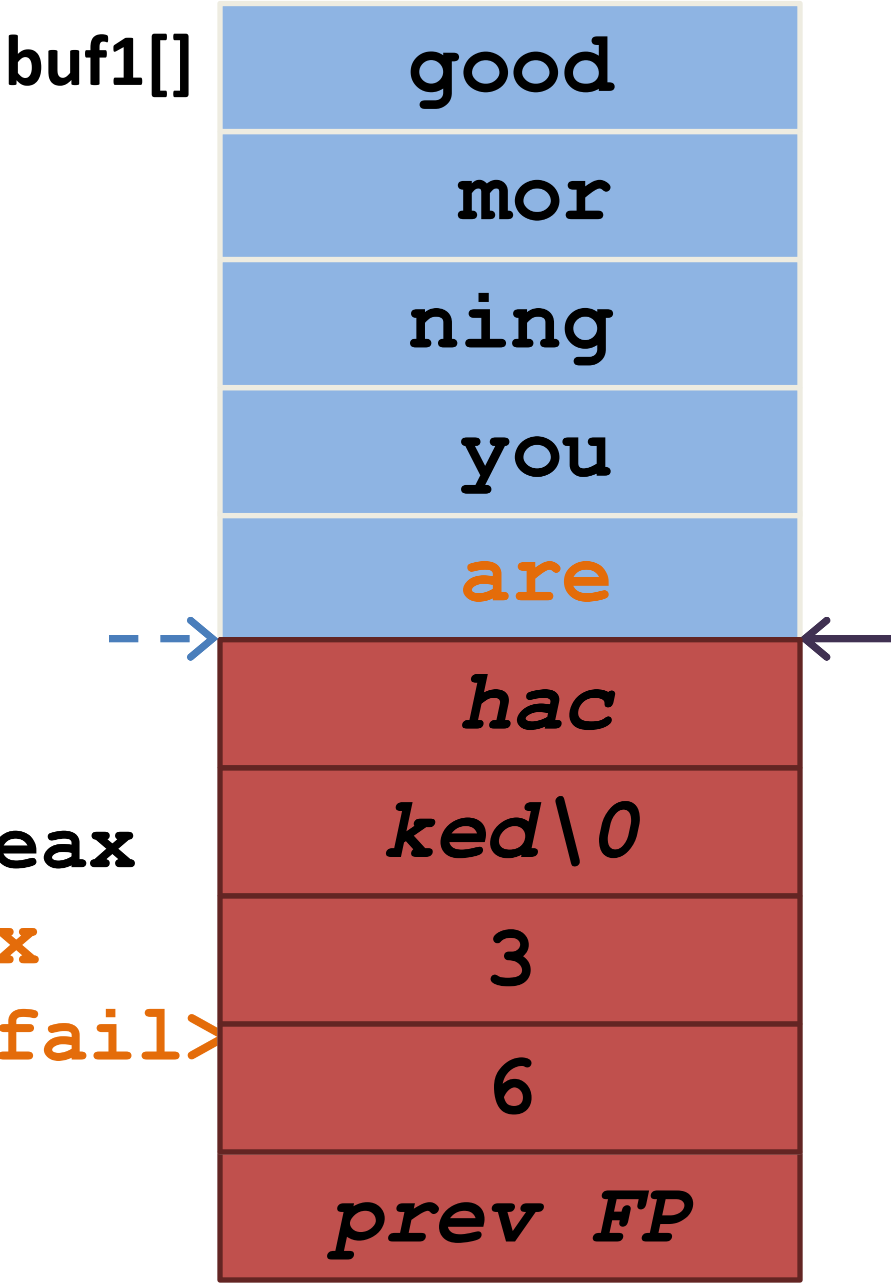
ret addr

3

6

prev FP

```
foo:
  push    %ebp
  mov     %esp, %ebp
  push    CANARY
  sub     $16, %esp
  ……
  call    gets
  mov     -4(%ebp), %eax
  cmp     CANARY, %eax
  jne     <stack_chk_fail>
  leave
  ret
```



buf1[]

good

mor

ning

you

are

hac

ked\0

3

6

prev FP

User input: good morning you are hacked

Stack Canary Value

- Exploit must contain the canary value to pass canary check

- **`CANARY = 0`:** can't `strcpy` past canary
- **`CANARY = \n`:** can't `gets` past canary
- **Random CANARY:** can't write past canary
  - Must not be discovered by attacker

Stack Canary

- Low cost and modest performance penalty
- Enabled by default in GCC and Clang
  - To disable: `-fno-stack-protector`
- Requires re-compile (need source code)

- Only protects return address against stack buffer overwrites. *Does not protect against non-stack writes!*

Control Flow Hijacking

- Altering control flow of a target program to cause it to do what attacker wants
  - Identify a **value** that will be loaded into **PC** (%eip)
  - Overwrite it (e.g., to point to shellcode)

- The simple stack buffer overflow attack from last lecture overwrote return address on stack
- Next: other control flow hijacking vulnerabilities

Function Pointers

```
char text[128];
void (*my_func)(int, int);

my_func = &foo;
*my_func(3, 6);     // equivalent to foo(3,6)
```

Q: Why does it defeat stack canary?

C++ Virtual Function

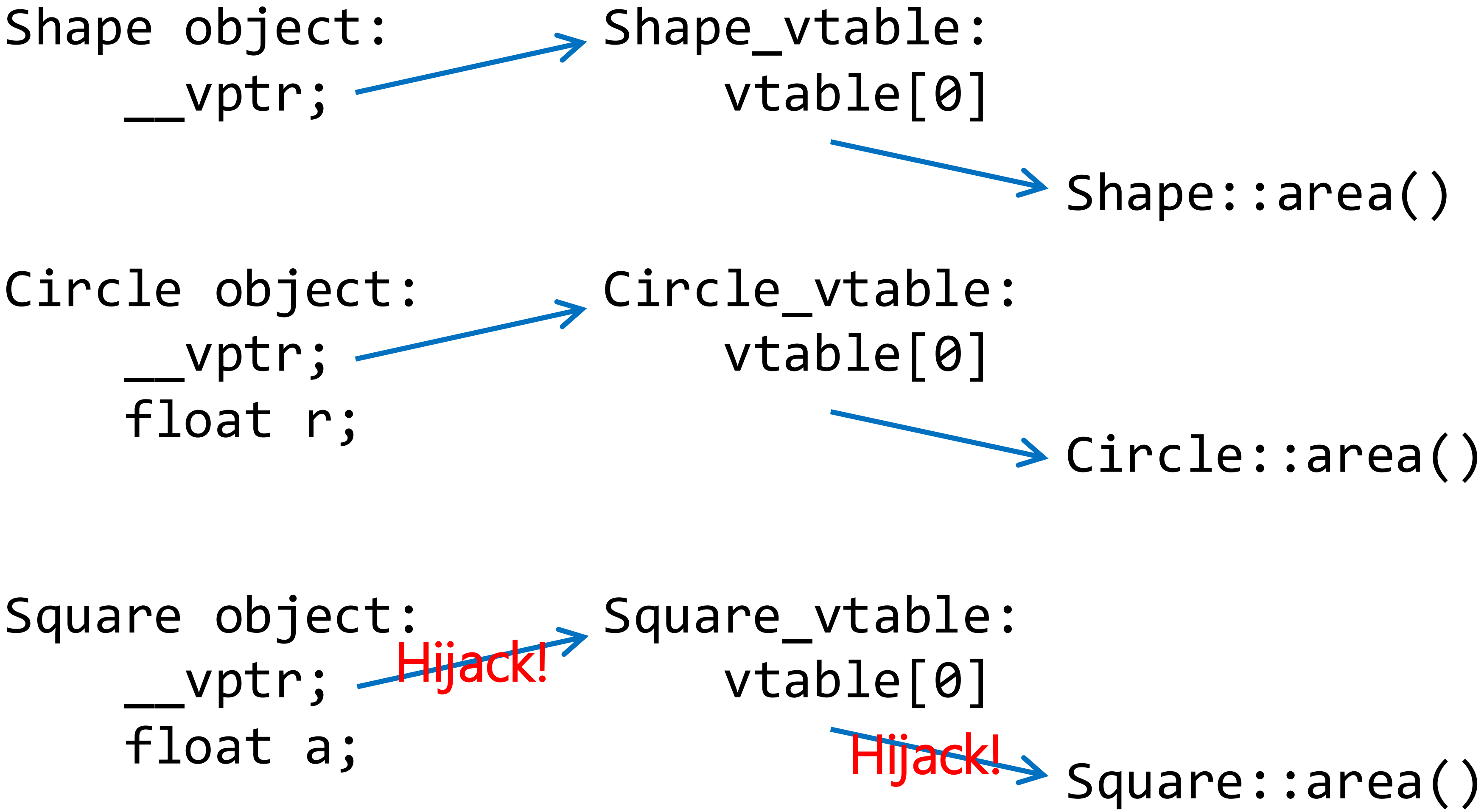
```
class Shape {
    virtual float area(void);
};

class Circle : Shape {
    float r;
    Circle(float r) {this->r = r;}
    float area() {return PI * r * r;}
};

class Square : Shape {
    float a;
    Square(float a) {this->a = a;}
    float area() {return a * a;}
};
```

C++ Class Polymorphism

```
Shape *s1, *s2;
s1 = new Square(3);
s2 = new Circle(5);
...
s1->area();           // calls Square::area()
s2->area();           // calls Circle::area()
```

- How does the program know which `area()` method to call?
- Virtual classes have an invisible member variable: virtual function table pointer

VTable: Array of Function Pointers



```
Shape object:           Shape_vtable:
    __vptr;                 vtable[0]
                                        Shape::area()

Circle object:          Circle_vtable:
    __vptr;                 vtable[0]
    float r;
                                        Circle::area()

Square object:          Square_vtable:
    __vptr;   Hijack!       vtable[0]
    float a;
                            Hijack!     Square::area()
```

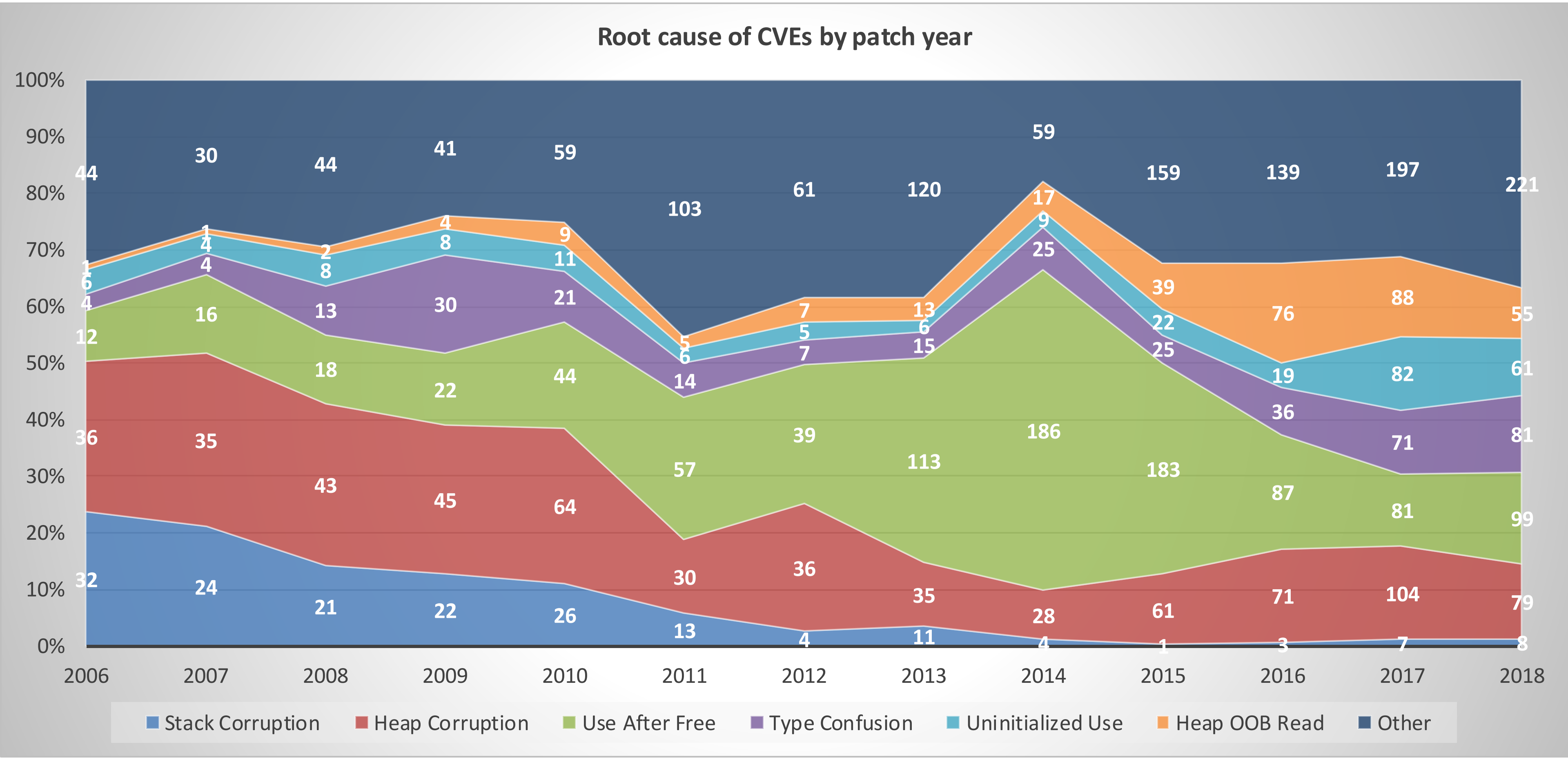
Use after Free

```
struct msg {                   struct student {
  void (*my_func)(char*);          int uid;
  char text[128];                  char name[128];
};                             };


student *s1 = malloc(sizeof(student));
free(s1);
msg *m1 = malloc(sizeof(msg));
    // may occupy the same space as s1 (freed)
s1->uid = updateUid();     // overwrite my_func
```

Control Flow Hijacking Vulnerabilities

- Stack buffer overflows now less common
  - Easy-to-find bugs getting fixed
  - Use of unsafe functions deprecated
  - Stack-specific countermeasures (canary) helped

- Other variants have grown in popularity



Top root causes since 2016:

#1: use after free

#2: heap corruption

#3: type confusion

#4: uninitialized use

Announcements and Review

- MP1 CP1 due today at 6 pm
  - No 24-hour auto extension for CP1
  - Autograder (v beta) is available on PrairieLearn
- Extra TA office hour every Wed 6-7 pm & Wed 1-4 pm when there is no discussion

- Last time:
  - Stack canary: detect overwrites to return address
  - Limitation: only protects return address (on stack).
  - Bypassed by heap buffer overflow, use after free, …

Control Flow Hijacking

- Altering control flow of a target program to cause it to do what attacker wants
- 1. Identify a **value** that will be loaded into **PC**
- 2. Overwrite it (to point to shellcode)
  - Deprecate unsafe functions and stack canaries
  - Other forms of control flow hijacking

Control Flow Hijacking

- Altering control flow of a target program to cause it to do what attacker wants
- 1. Identify a **value** that will be loaded into **PC**
- 2. Overwrite it (to point to shellcode)
  - Deprecate unsafe functions and stack canaries
  - Other forms of control flow hijacking
- 3. Implicit step: shellcode runs
  - Can we prevent execution of injected code?

Observation on Code Injection

- Root cause: confusion between code and data
  - Will be a recurring theme in this course
- Defense: distinguish code and data
  - Data should not be executable
  - Code need not be writable
    - * Self-modifying code is a thing but uncommon

Data Execution Prevention

- Make each memory region either writable or executable, never both
  - Make use of W (writable) and NX (no execute) bits in hardware page tables
  - Also called W^X (write xor execute)
  - Supported by all major processors and OS
  - Malicious code can only be injected to writable regions. Attempting to run it will cause an error.

Data Execution Prevention

- A memory region is writable or executable, never both

- No way to run shellcode ... right?
- Can still execute the program’s code that is already there, but is that a problem?
- Isn’t the program’s code *good* code?
- Can *good code* do *bad things?*

Return-to-libc Attacks

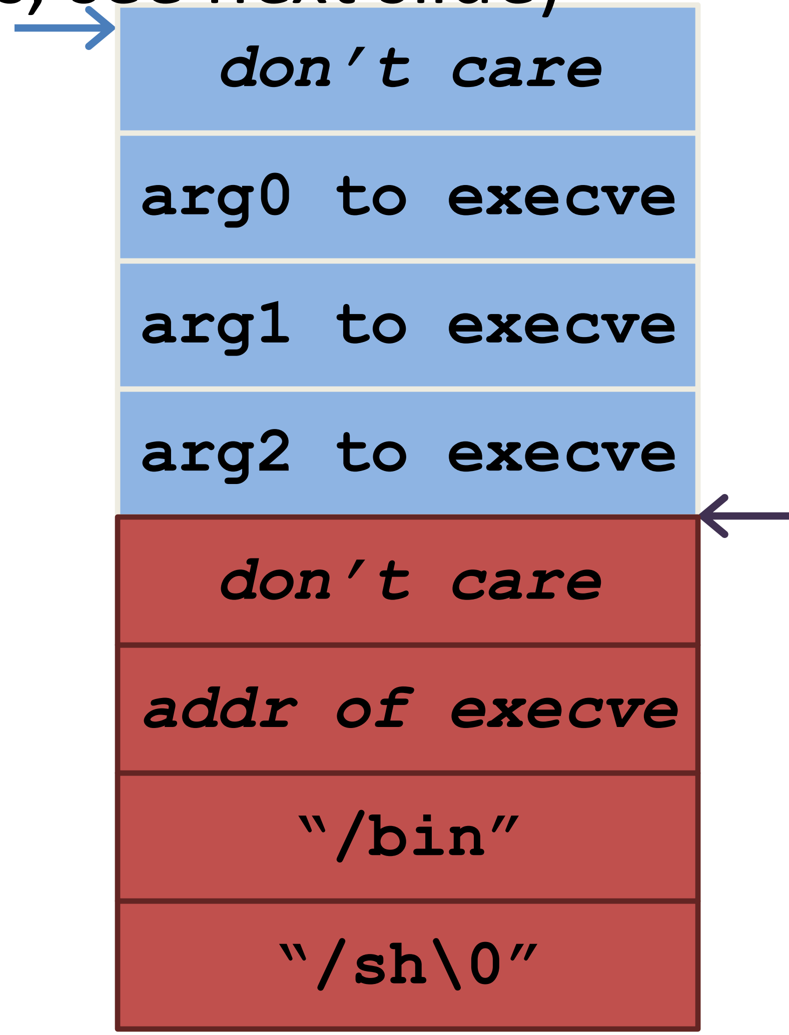
- Many programs rely on C Standard library to perform common tasks, e.g.,
  - Access files, kill processes, create users, execve, …

- The attacker needs to (and may be able to) set up arguments on stack

Return-to-libc Attacks

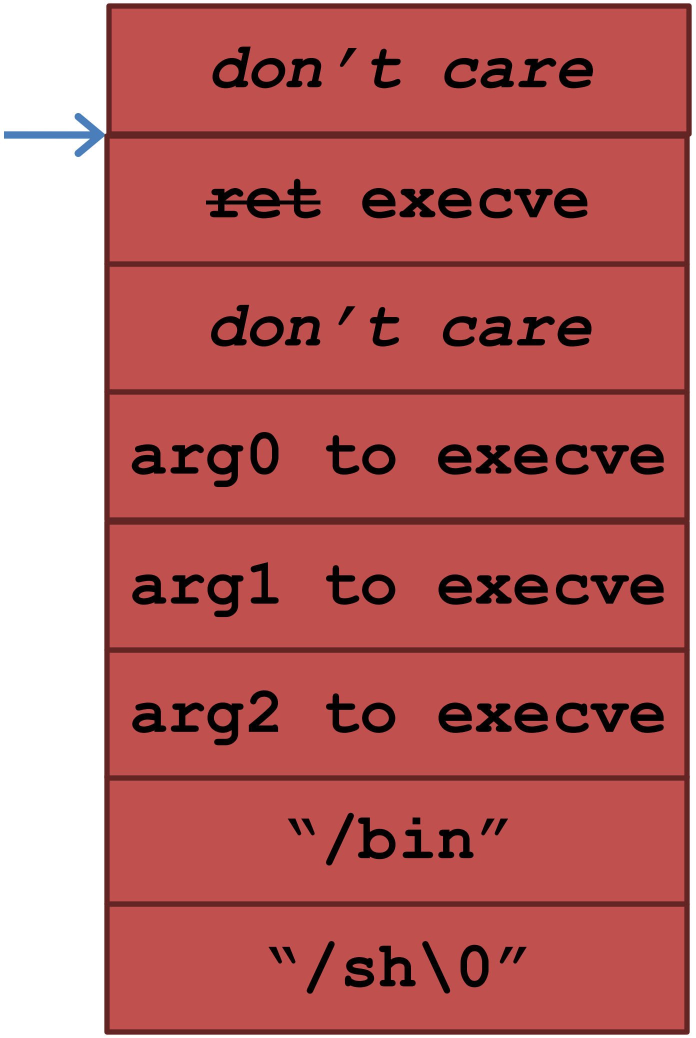
(buggy slide shown in class, see next slide)

```
execve:

push    %ebx
mov     0x10(%esp), %edx
mov     0xc(%esp), %ecx
mov     0x8(%esp), %ebx
mov     0xb, %eax
int     $0x80
```



Return-to-libc Attacks

```
execve:

push    %ebx
mov     0x10(%esp), %edx
mov     0xc(%esp), %ecx
mov     0x8(%esp), %ebx
mov     0xb, %eax
int     $0x80
```



don’t care

~~ret~~ execve

don’t care

arg0 to execve

arg1 to execve

arg2 to execve

“/bin”

“/sh\0”

Return-to-libc Attacks

- Many programs rely on C Standard library to perform common tasks, e.g.,
  - Access files, kill processes, create users, execve, …
- The attacker needs to (and may be able to) set up arguments on stack
  - Require precise knowledge about state of stack
- Does not work if libc is not loaded

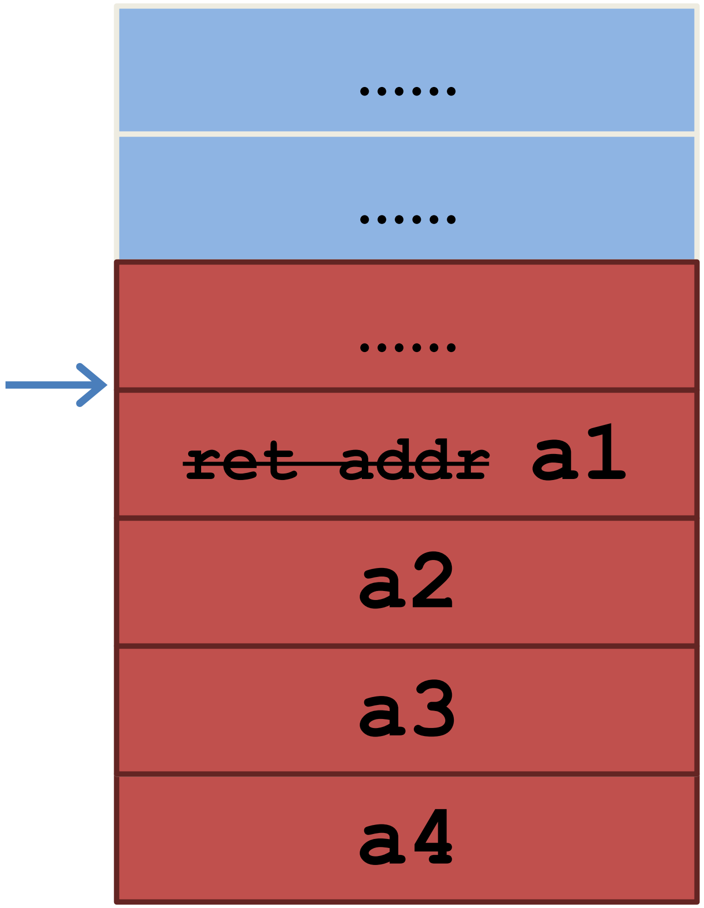
Find Shellcode in Target Program?

```
6a 0b              push    $0xb
58                 pop     %eax
31 c9              xor     %ecx, %ecx
31 d2              xor     %edx, %edx
52                 push    %edx
68 2f 2f 73 68     push    $0x68732f2f
68 2f 62 69 6e     push    $0x6e69622f
89 e3              mov     %esp, %ebx
cd 80              int     $0x80
```

Another Look at Stack Buffer Overflow

- What happens after we jump to a location via `ret`?

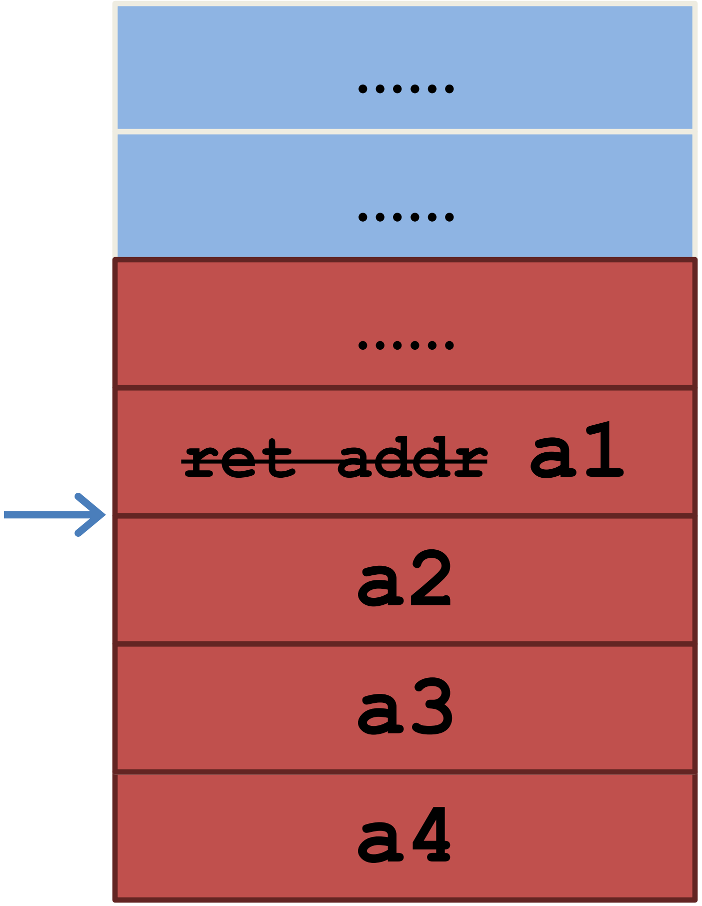
```
foo:
     ...
     ret●
```



Another Look at Stack Buffer Overflow

- What happens after we jump to a location via `ret`?
  - Jump to a1, execute there
  - %ESP → next value on stack

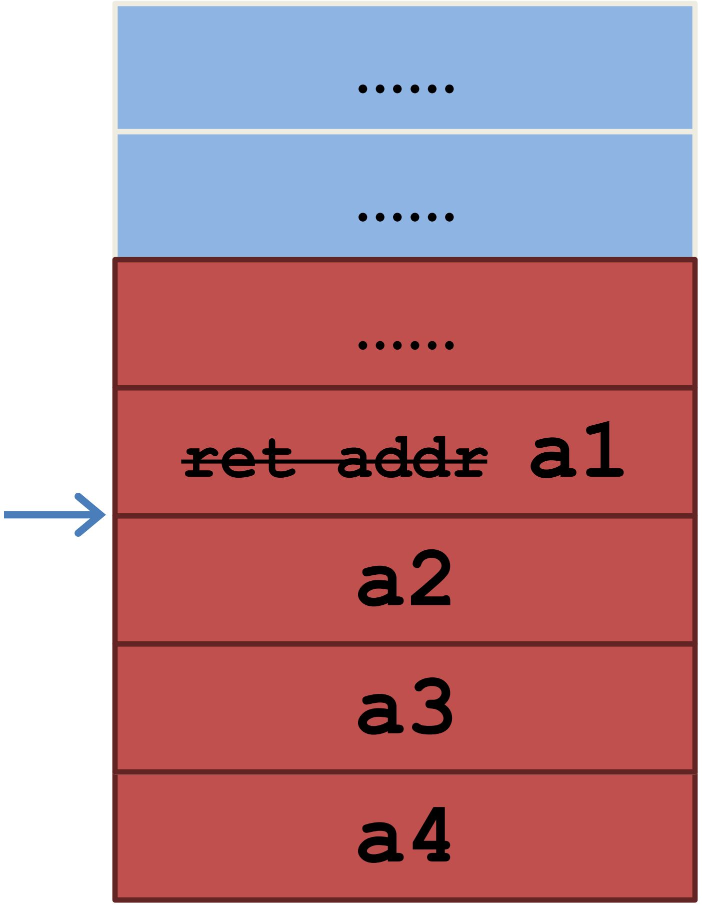
```
foo:
    ...
    ret

a1: some instr
```



Another Look at Stack Buffer Overflow

- What happens if the next instruction is also a `ret`?

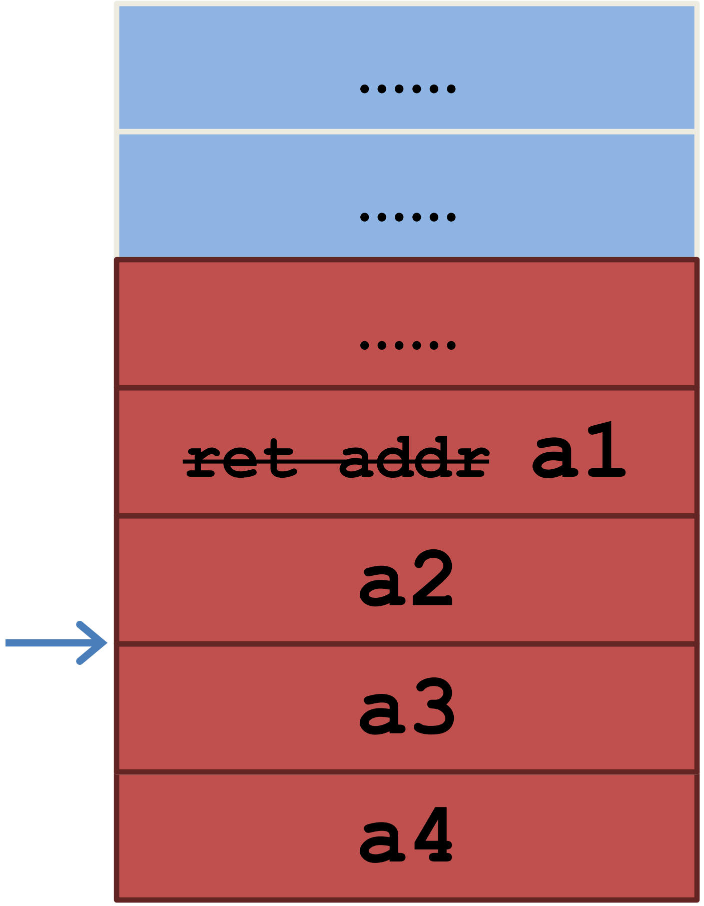
```
foo:
    ...
    ret

a1: some instr
    ret●
```



Another Look at Stack Buffer Overflow

- What happens if the next instruction is also a `ret`?
  - Jump to a2, execute there
  - %ESP → a3

```
foo:                a2: instr●
    ...
    ret

a1: some instr
    ret
```

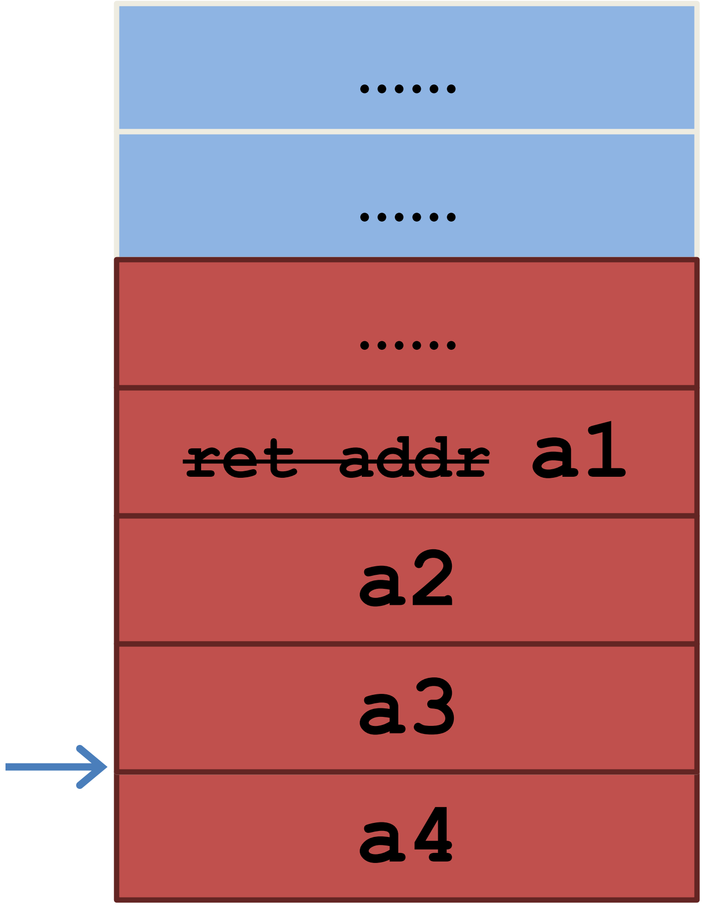


Another Look at Stack Buffer Overflow

- What happens if the next instruction is also a `ret`?
  - Jump to a3, execute there
  - %ESP → a4

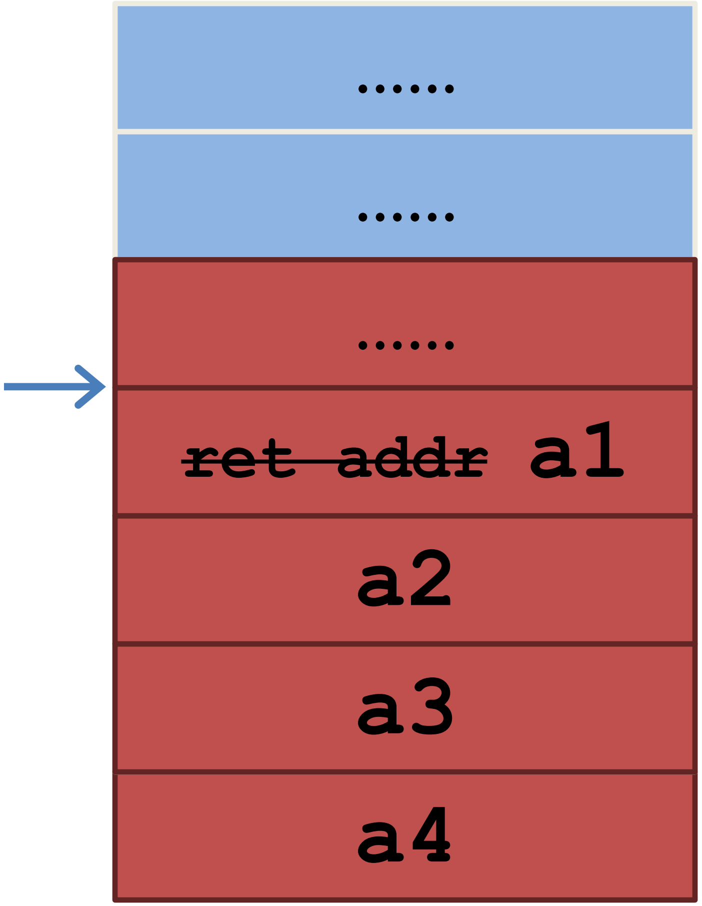
```
foo:
     ...
     ret

a1:  some instr
     ret
```

```
a2:  instr
     ret

a3:  instr●
     ret
```



Another Look at Stack Buffer Overflow

- Overwrite stack with a sequence of addresses
  - – Jump (ret) to an address
  - – Do some useful work
  - – Repeat



Return-Oriented Programming (ROP)

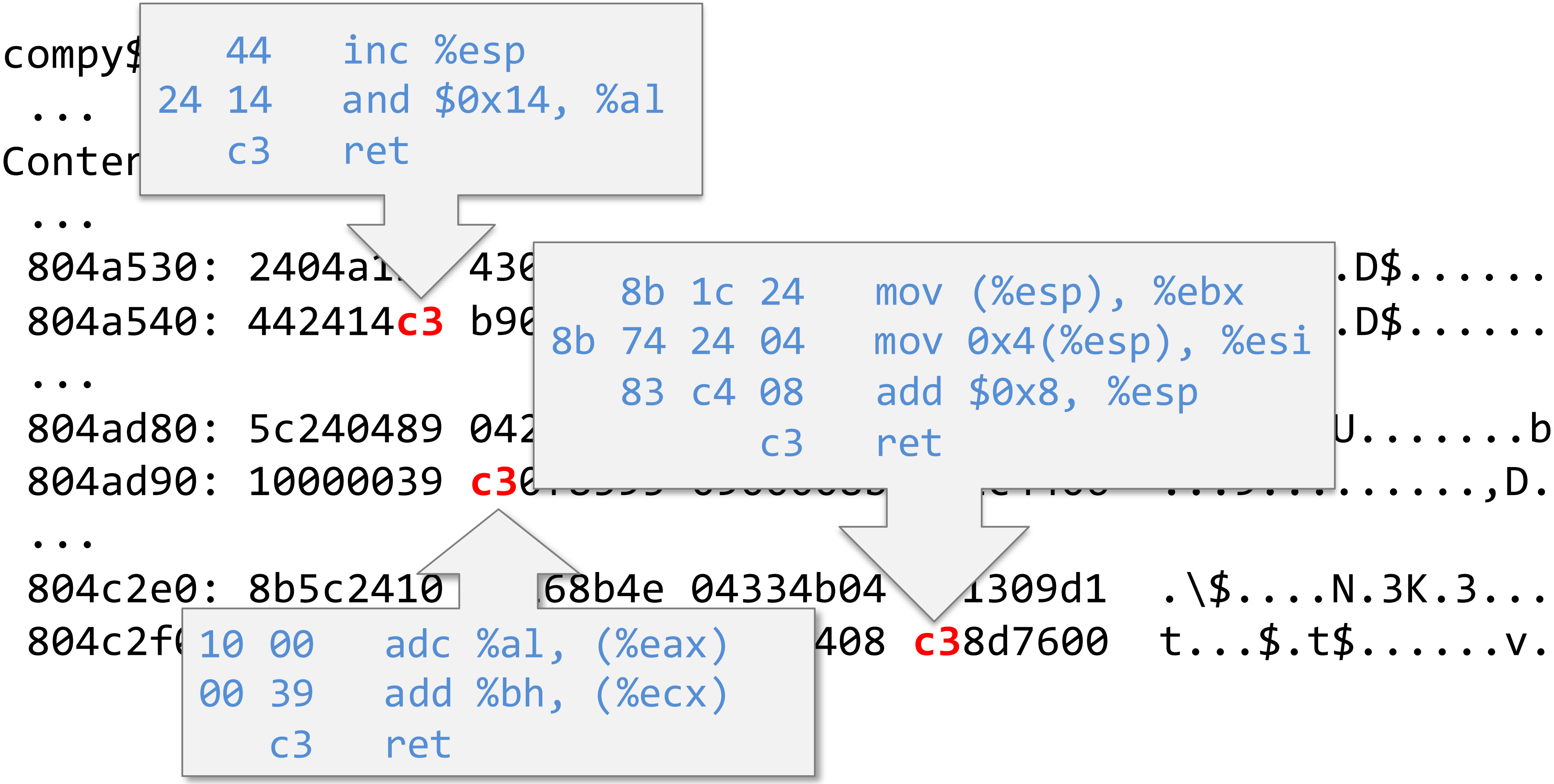
- Workflow
  - Dump executable portions of target program
  - Identify byte sequences ending in `0xC3` (`ret`)
    - Such a code fragment is called a **gadget**
  - Figure out what each gadget does — use a dissembler, e.g., https://onlinedisassembler.com/
  - Chain together useful gadgets

Havov Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86), CCS, 2007

Finding Gadgets

```
compy$ objdump -s /bin/ls
 ...
Contents of section .text:
 ...
 804a530: 2404a120 430608c7 44241800 000000c7  $.. C...D$......
 804a540: 442414c3 b90508c7 442410d3 b9050889  D$......D$......
 ...
 804ad80: 5c240489 0424e855 8b000085 c00f8462  \$...$.U.......b
 804ad90: 10000039 c30f8595 0900008b 0d2c4406  ...9.........,D.
 ...
 804c2e0: 8b5c2410 8b168b4e 04334b04 331309d1  .\$....N.3K.3...
 804c2f0: 740e8b1c 248b7424 0483c408 c38d7600  t...$.t$......v.
```

Finding Gadgets



```
compy$
 ...
Conter
 ...
 804a530: 2404a1      43                                        .D$......
 804a540: 442414c3 b9                                           .D$......
 ...
 804ad80: 5c240489 042                                          U.......b
 804ad90: 10000039 c30                                          ,D.
 ...
 804c2e0: 8b5c2410    68b4e 04334b04   1309d1  .\$....N.3K.3...
 804c2f                        408 c38d7600  t...$.t$......v.
```

```
   44    inc %esp
24 14    and $0x14, %al
   c3    ret
```

```
   8b 1c 24    mov (%esp), %ebx
8b 74 24 04    mov 0x4(%esp), %esi
   83 c4 08    add $0x8, %esp
         c3    ret
```

```
10 00    adc %al, (%eax)
00 39    add %bh, (%ecx)
   c3    ret
```

Tips on Finding Gadgets

- Suffix of a gadget is also a gadget

- Gadgets may not be “intended” code

```
89 5c 24 04     mov %ebx, 4(%esp)
89 c3           mov $eax, $ebx

04 89           add $89,  %al
c3              ret
```

Exercise with gadgets from /bin/ls

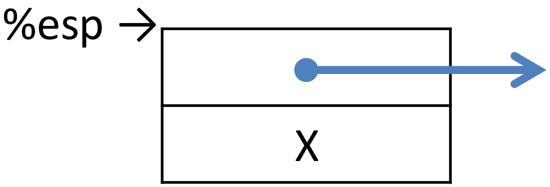
- What can I do with these?

```
8b 1c 24          mov (%esp), %ebx
8b 74 24 04       mov 0x4(%esp), %esi
83 c4 08          add $0x8, %esp
c3                ret

b8 01 00 00 00    mov $0x1, %eax
8b 34 24          mov (%esp), %esi
8b 7c 24 04       mov 0x4(%esp), %edi
83 c4 08          add $0x8, %esp
c3                ret

89 eb             mov %ebp, %ebx
5d                pop %eax
c3                ret
```

```
89 ea          mov %ebp, %edx
31 c0          xor %eax, %eax
8b 7c 24 04    mov 0x4(%esp), %edi
89 1f          mov %ebx,(%edi)
89 0a          mov %ecx,(%edx)
8b 54 24 20    mov 0x20(%esp),%edx
89 32          mov %esi,(%edx)
83 c4 0c       add $0xc,%esp
5b             pop %ebx
5e             pop %esi
5f             pop %edi
5d             pop %ebp
c3             ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X

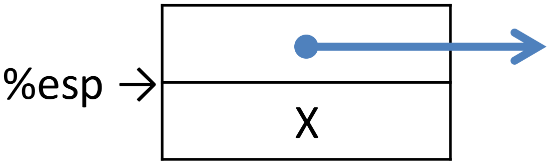


```
●c3                 ret

 89 ea              mov %ebp, %edx
 31 c0              xor %eax, %eax
 8b 7c 24 04        mov 0x4(%esp), %edi
 89 1f              mov %ebx,(%edi)
 89 0a              mov %ecx,(%edx)
 8b 54 24 20        mov 0x20(%esp),%edx
 89 32              mov %esi,(%edx)
 83 c4 0c           add $0xc,%esp
 5b                 pop %ebx
 5e                 pop %esi
 5f                 pop %edi
 5d                 pop %ebp
 c3                 ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X

```
 c3              ret

 89 ea           mov %ebp, %edx
 31 c0           xor %eax, %eax
 8b 7c 24 04     mov 0x4(%esp), %edi
 89 1f           mov %ebx,(%edi)
 89 0a           mov %ecx,(%edx)
 8b 54 24 20     mov 0x20(%esp),%edx
 89 32           mov %esi,(%edx)
 83 c4 0c        add $0xc,%esp
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
●5d              pop %ebp
 c3              ret
```

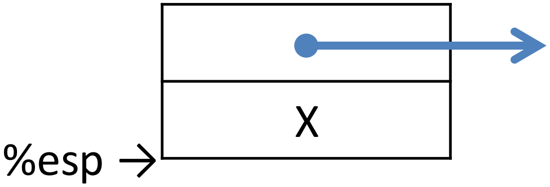


Exercise with gadgets from /bin/ls

- Set EBP to X

%ebp = X

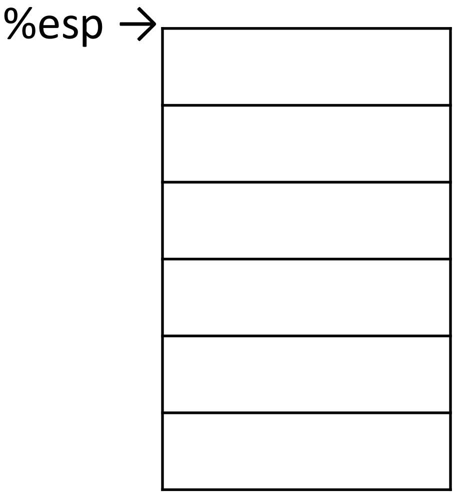
```
 c3              ret

 89 ea           mov %ebp, %edx
 31 c0           xor %eax, %eax
 8b 7c 24 04     mov 0x4(%esp), %edi
 89 1f           mov %ebx,(%edi)
 89 0a           mov %ecx,(%edx)
 8b 54 24 20     mov 0x20(%esp),%edx
 89 32           mov %esi,(%edx)
 83 c4 0c        add $0xc,%esp
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
 5d              pop %ebp
●c3              ret
```



Exercise with gadgets from /bin/ls

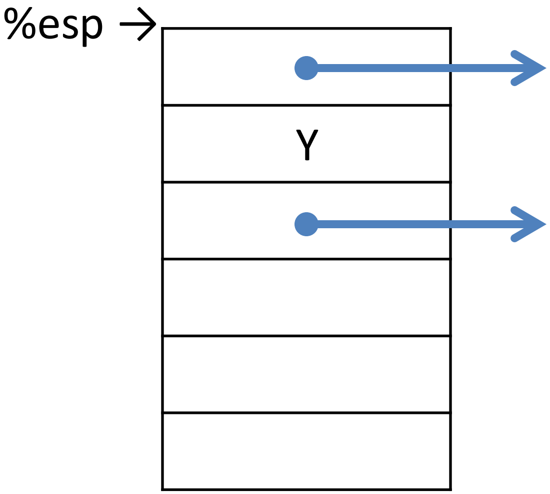
- Set EBP to X and EBX to Y

```
●c3                 ret



 ...                  ...
 5b                 pop %ebx
 5e                 pop %esi
 5f                 pop %edi
 5d                 pop %ebp
 c3                 ret

 89 eb              mov %ebp, %ebx
 5d                 pop %eax
 c3                 ret
```



Exercise with gadgets from /bin/ls

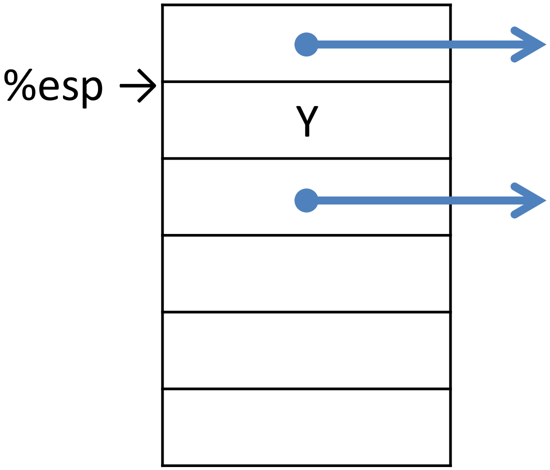
- Set EBP to X and EBX to Y



```
●c3              ret



  ...              ...
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
 5d              pop %ebp
 c3              ret

 89 eb           mov %ebp, %ebx
 5d              pop %eax
 c3              ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

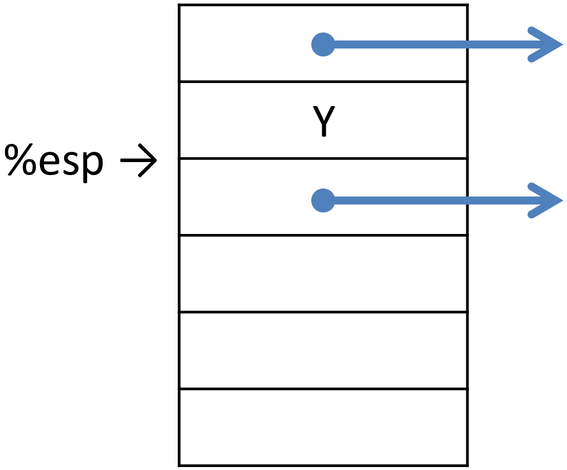


```
c3              ret



...              ...
5b              pop %ebx
5e              pop %esi
5f              pop %edi
5d              pop %ebp
c3              ret

89 eb           mov %ebp, %ebx
5d              pop %eax
c3              ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

%ebp = Y

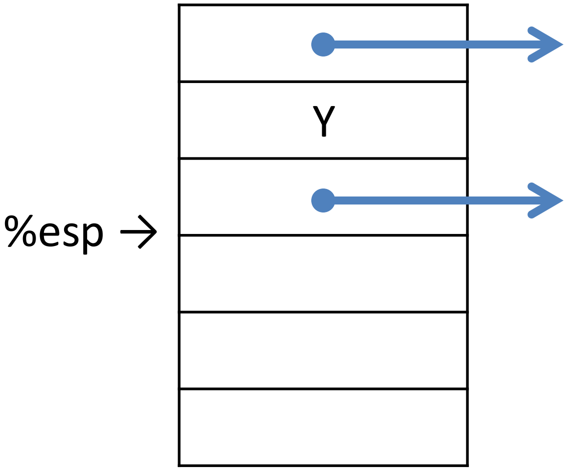


```
 c3                  ret



 ...                   ...
 5b                  pop %ebx
 5e                  pop %esi
 5f                  pop %edi
 5d                  pop %ebp
●c3                  ret

 89 eb               mov %ebp, %ebx
 5d                  pop %eax
 c3                  ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

%ebp = Y

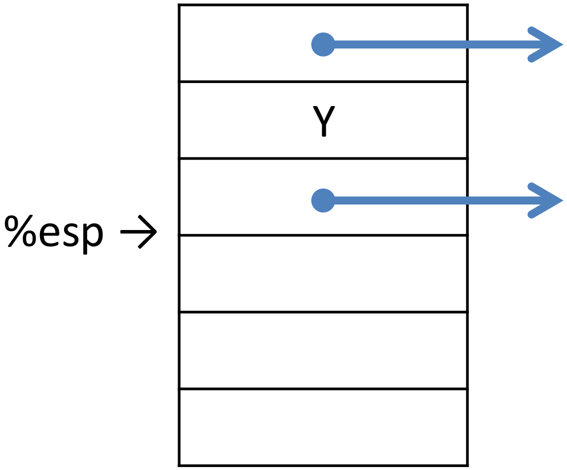


```
 c3               ret


 ...               ...
 5b               pop %ebx
 5e               pop %esi
 5f               pop %edi
 5d               pop %ebp
 c3               ret

•89 eb            mov %ebp, %ebx
 5d               pop %eax
 c3               ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

```
%ebp = Y
%ebx = Y
```

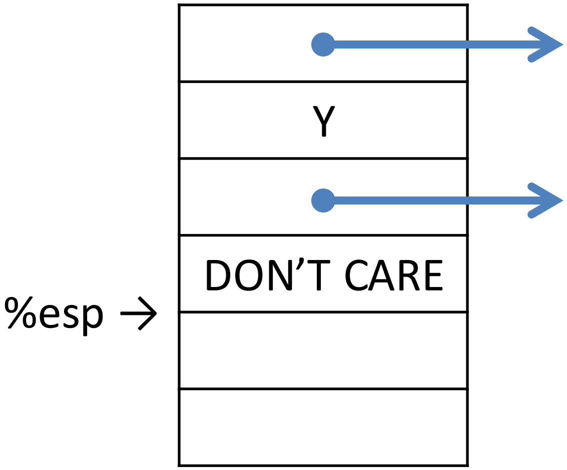


```
c3             ret


...              ...
5b             pop %ebx
5e             pop %esi
5f             pop %edi
5d             pop %ebp
c3             ret

89 eb          mov %ebp, %ebx
●5d            pop %eax
c3             ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

```
%ebp = Y
%ebx = Y
```

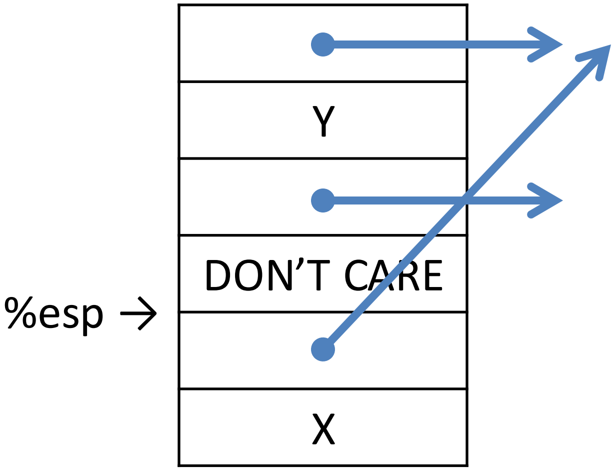


```
c3          ret



...          ...
5b          pop %ebx
5e          pop %esi
5f          pop %edi
5d          pop %ebp
c3          ret

89 eb       mov %ebp, %ebx
5d          pop %eax
●c3         ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

```
%ebp = Y
%ebx = Y
```

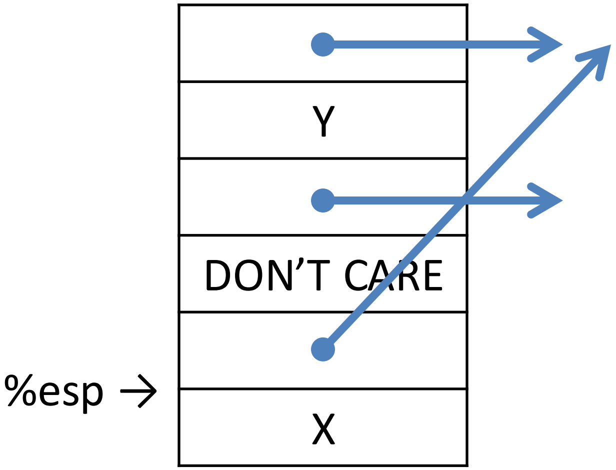


```
 c3              ret


 ...               ...
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
 5d              pop %ebp
 c3              ret

 89 eb           mov %ebp, %ebx
 5d              pop %eax
●c3              ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

```
%ebp  =  Y
%ebx  =  Y
```

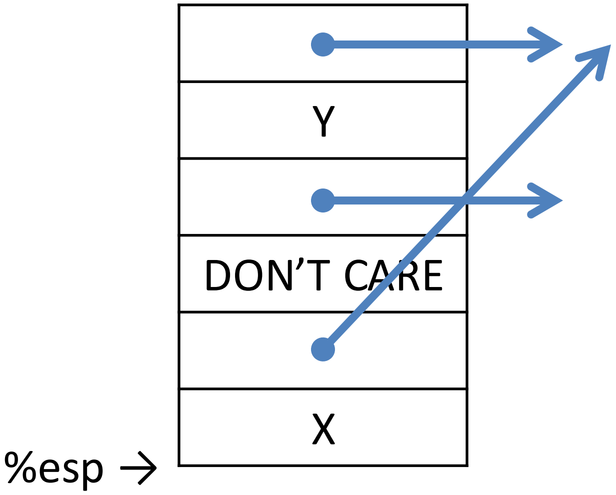


```
 c3              ret



 ...              ...
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
●5d              pop %ebp
 c3              ret

 89 eb           mov %ebp, %ebx
 5d              pop %eax
 c3              ret
```

Exercise with gadgets from /bin/ls

- Set EBP to X and EBX to Y

```
%ebp = X
%ebx = Y
```



Y

DON'T CARE

X

%esp →

```
 c3              ret



 ...              ...
 5b              pop %ebx
 5e              pop %esi
 5f              pop %edi
 5d              pop %ebp
●c3              ret

 89 eb           mov %ebp, %ebx
 5d              pop %eax
 c3              ret
```

Exercise with gadgets from /bin/ls

- Write value X to location Y?

```
8b 1c 24          mov (%esp), %ebx
8b 74 24 04       mov 0x4(%esp), %esi
83 c4 08          add $0x8, %esp
c3                ret

b8 01 00 00 00    mov $0x1, %eax
8b 34 24          mov (%esp), %esi
8b 7c 24 04       mov 0x4(%esp), %edi
83 c4 08          add $0x8, %esp
c3                ret

89 eb             mov %ebp, %ebx
5d                pop %eax
c3                ret
```

```
89 ea          mov %ebp, %edx
31 c0          xor %eax, %eax
8b 7c 24 04    mov 0x4(%esp), %edi
89 1f          mov %ebx,(%edi)
89 0a          mov %ecx,(%edx)
8b 54 24 20    mov 0x20(%esp),%edx
89 32          mov %esi,(%edx)
83 c4 0c       add $0xc,%esp
5b             pop %ebx
5e             pop %esi
5f             pop %edi
5d             pop %ebp
c3             ret
```

Exercise with gadgets from /bin/ls

- Write value X to location Y?

```
8b 1c 24          mov (%esp), %ebx
8b 74 24 04       mov 0x4(%esp), %esi
83 c4 08          add $0x8, %esp
c3                ret

b8 01 00 00 00    mov $0x1, %eax
8b 34 24          mov (%esp), %esi
8b 7c 24 04       mov 0x4(%esp), %edi
83 c4 08          add $0x8, %esp
c3                ret

89 eb             mov %ebp, %ebx
5d                pop %eax
c3                ret
```

```
89 ea             mov %ebp, %edx
31 c0             xor %eax, %eax
8b 7c 24 04       mov 0x4(%esp), %edi
89 1f             mov %ebx,(%edi)
89 0a             mov %ecx,(%edx)
8b 54 24 20       mov 0x20(%esp),%edx
89 32             mov %esi,(%edx)
83 c4 0c          add $0xc,%esp
5b                pop %ebx
5e                pop %esi
5f                pop %edi
5d                pop %ebp
c3                ret
```

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G116 G10c

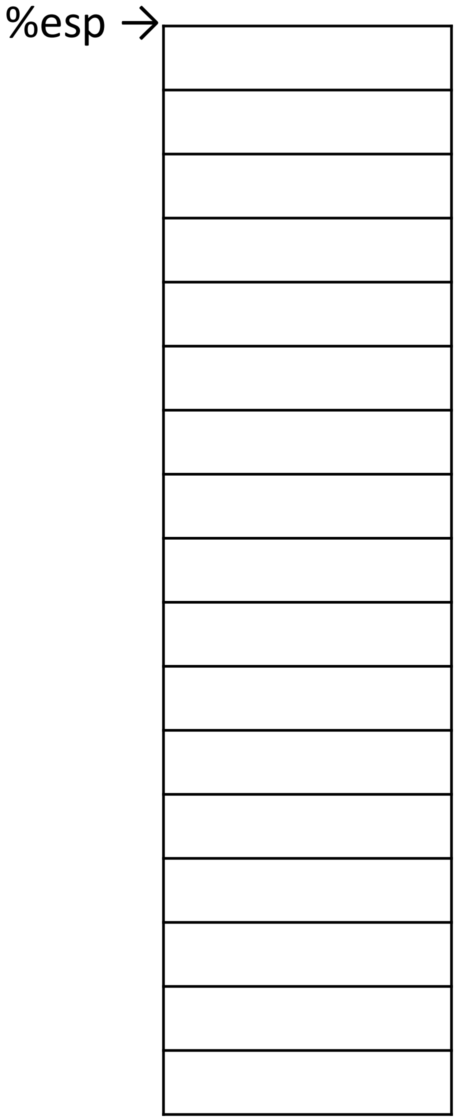
```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G116 G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```

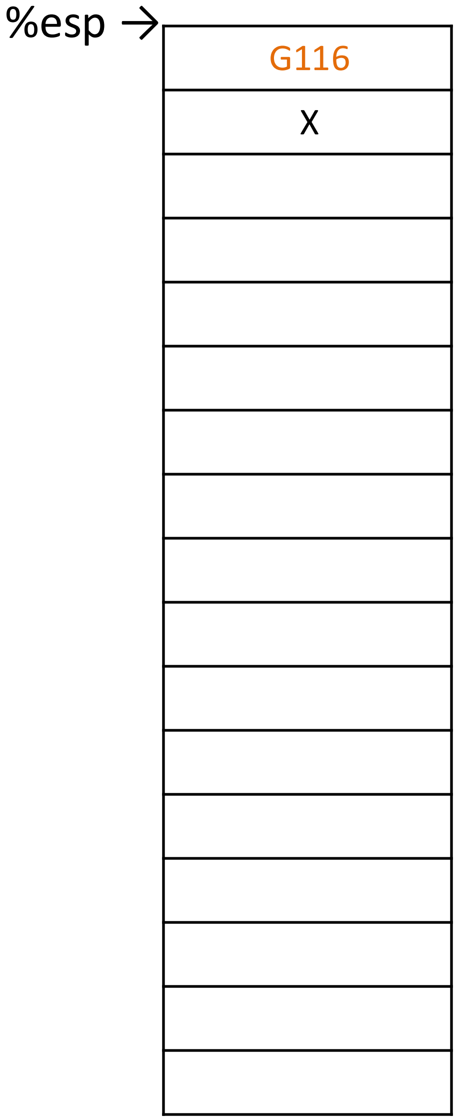


Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G116 G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```



%esp →

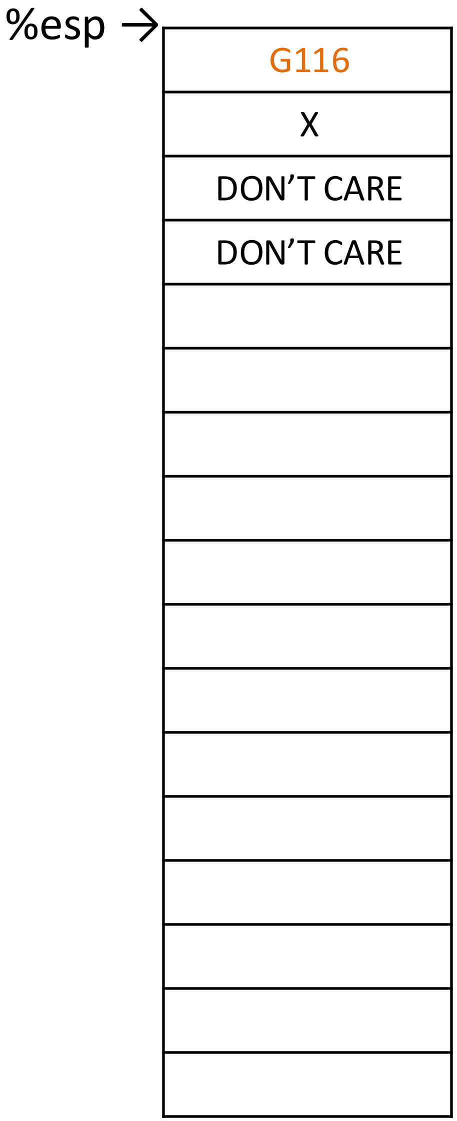
G116
X

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G116 G10c

```
G100:  89 ea         mov %ebp, %edx
G102:  31 c0         xor %eax, %eax
G104:  8b 7c 24 04   mov 0x4(%esp), %edi
G108:  89 1f         mov %ebx,(%edi)
G10a:  89 0a         mov %ecx,(%edx)
G10c:  8b 54 24 20   mov 0x20(%esp),%edx
G110:  89 32         mov %esi,(%edx)
G112:  83 c4 0c      add $0xc,%esp
G115:  5b            pop %ebx
G116:  5e            pop %esi
G117:  5f            pop %edi
G118:  5d            pop %ebp
G119:  c3            ret
```



%esp →

G116

X

DON'T CARE

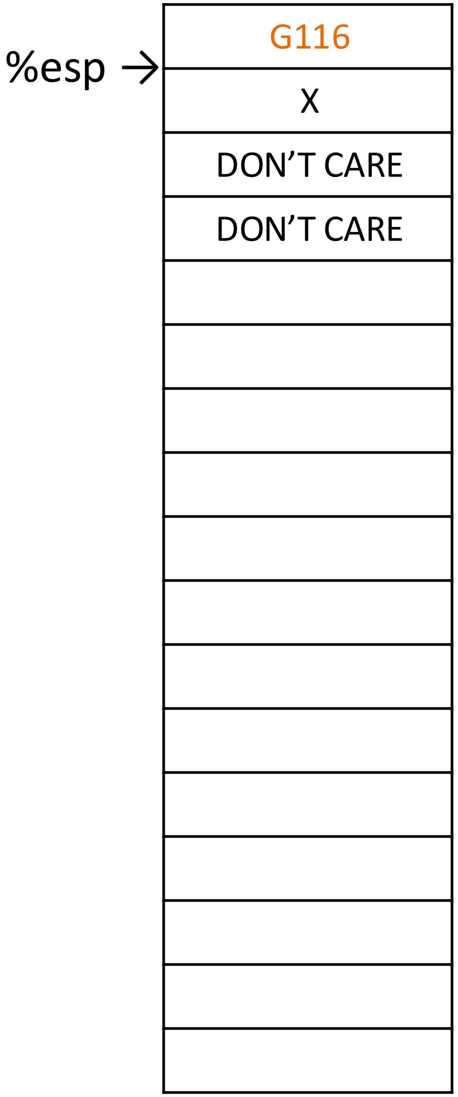
DON'T CARE

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G116 G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116: ●5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```

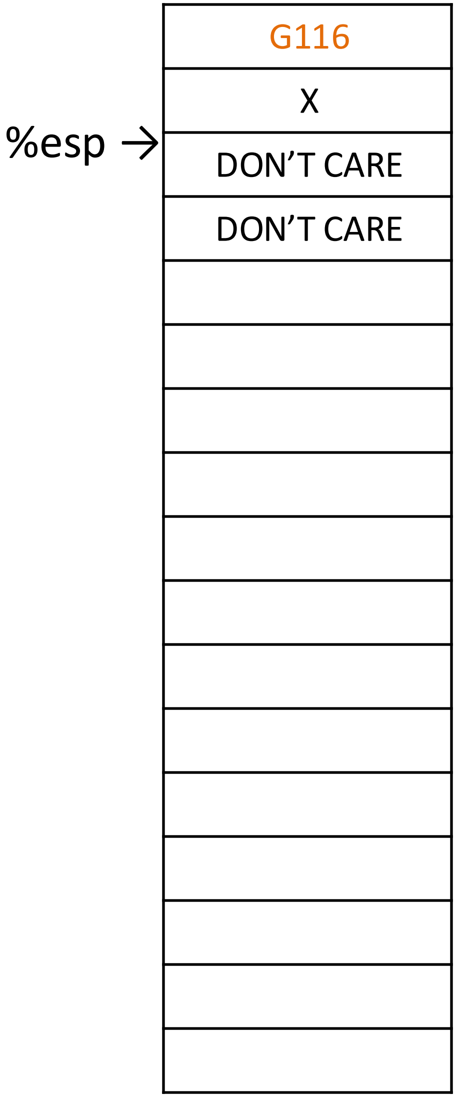


Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117: ●5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```



%esp →

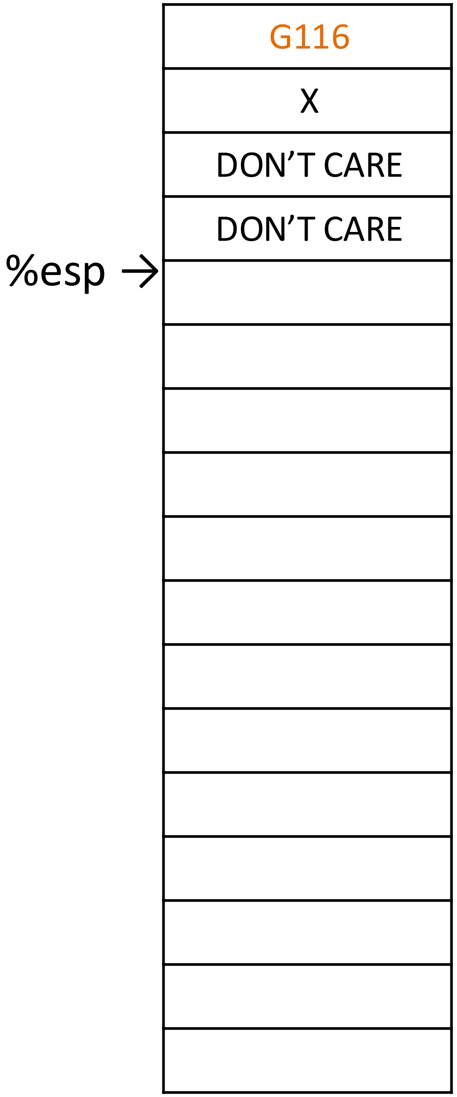
G116
X
DON'T CARE
DON'T CARE

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: ●c3             ret
```



G116

X

DON’T CARE

DON’T CARE

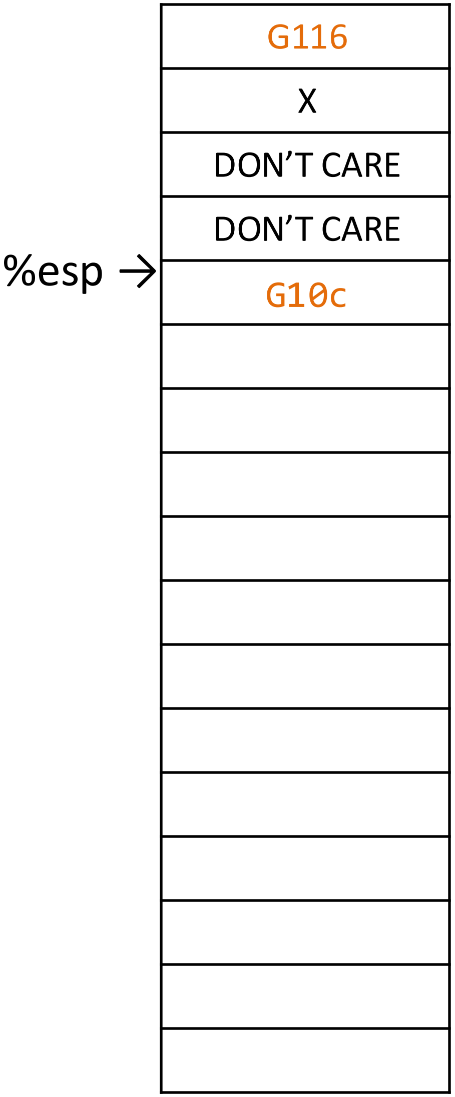
%esp →

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: ●c3             ret
```



G116

X

DON’T CARE

DON’T CARE

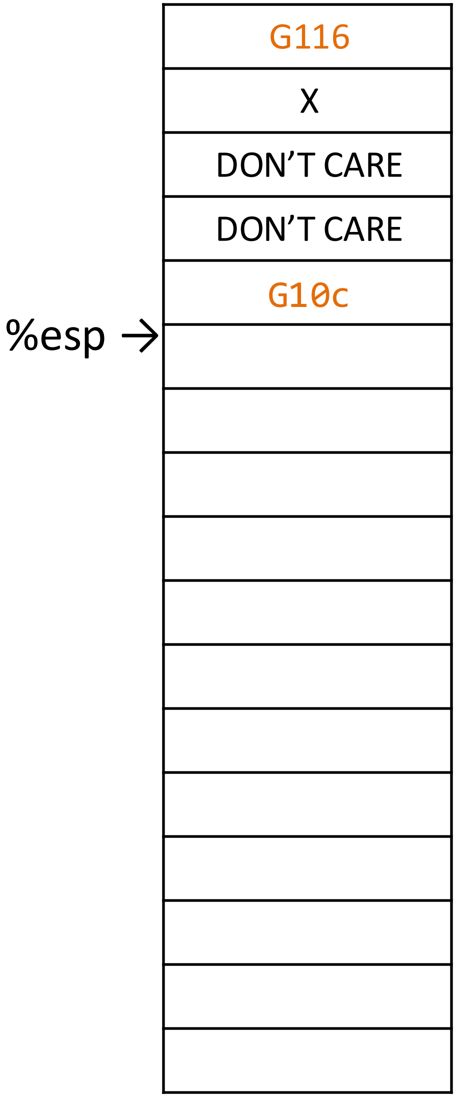
%esp → G10c

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c: ●8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```



G116

X

DON'T CARE

DON'T CARE

G10c

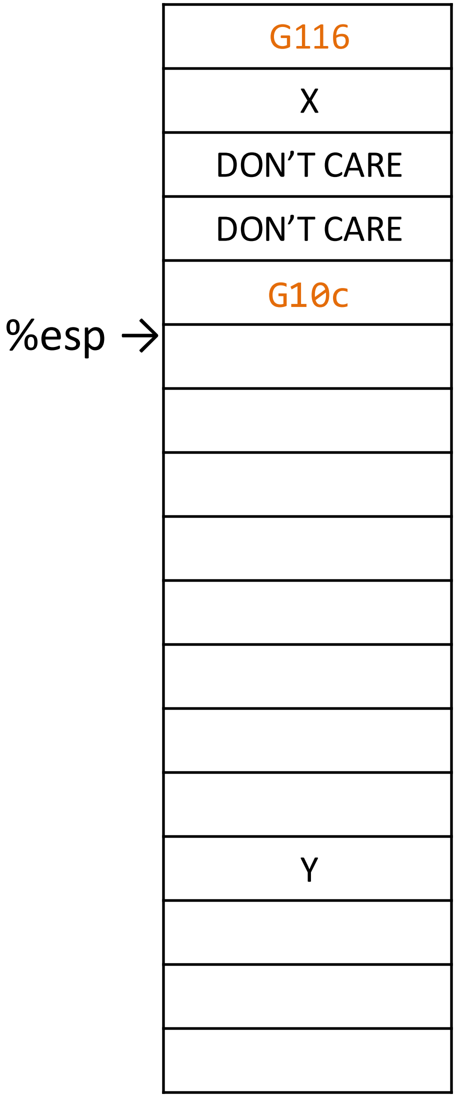
%esp →

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

G10c

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c: ●8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```

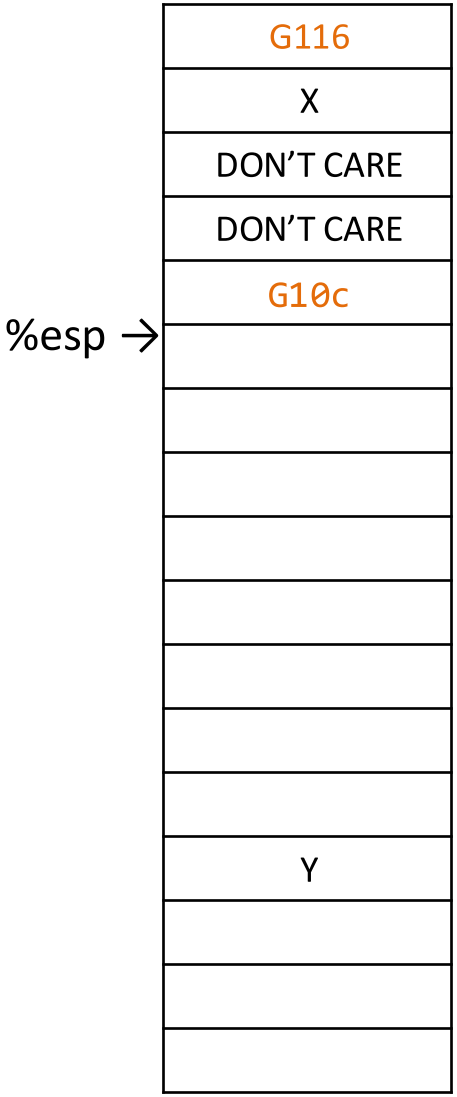


	G116
	X
	DON'T CARE
	DON'T CARE
	G10c
%esp →	
	Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

```
G100:  89 ea         mov %ebp, %edx
G102:  31 c0         xor %eax, %eax
G104:  8b 7c 24 04   mov 0x4(%esp), %edi
G108:  89 1f         mov %ebx,(%edi)
G10a:  89 0a         mov %ecx,(%edx)
G10c:  8b 54 24 20   mov 0x20(%esp),%edx
G110: ●89 32         mov %esi,(%edx)
G112:  83 c4 0c      add $0xc,%esp
G115:  5b            pop %ebx
G116:  5e            pop %esi
G117:  5f            pop %edi
G118:  5d            pop %ebp
G119:  c3            ret
```



G116

X

DON'T CARE

DON'T CARE

G10c

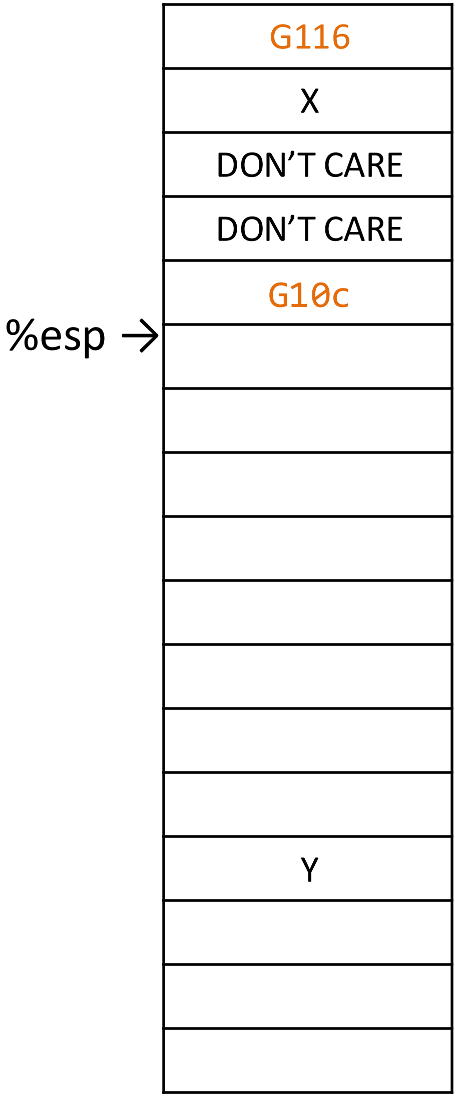
%esp →

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112: ●83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```



G116

X

DON'T CARE

DON'T CARE

G10c

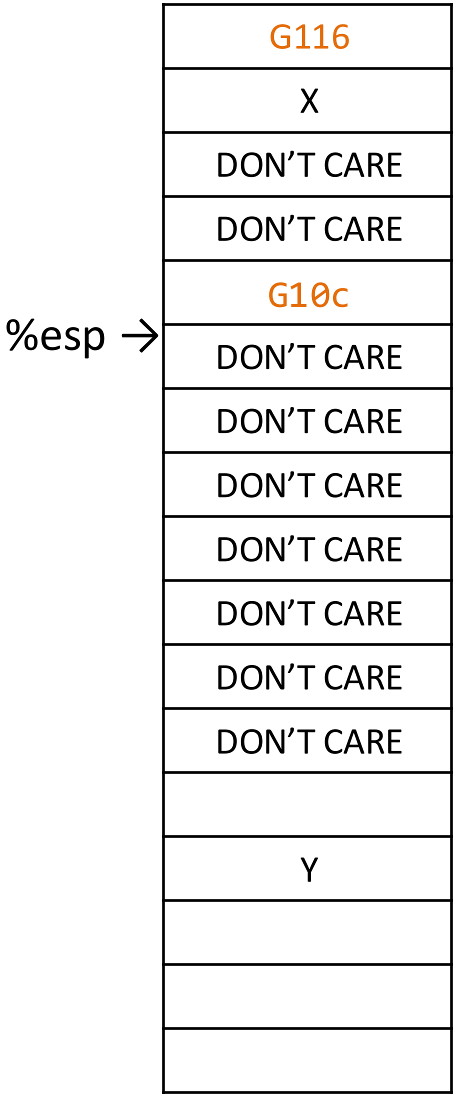
%esp →

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112: ●83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119:  c3             ret
```



G116

X

DON'T CARE

DON'T CARE

G10c

%esp →

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

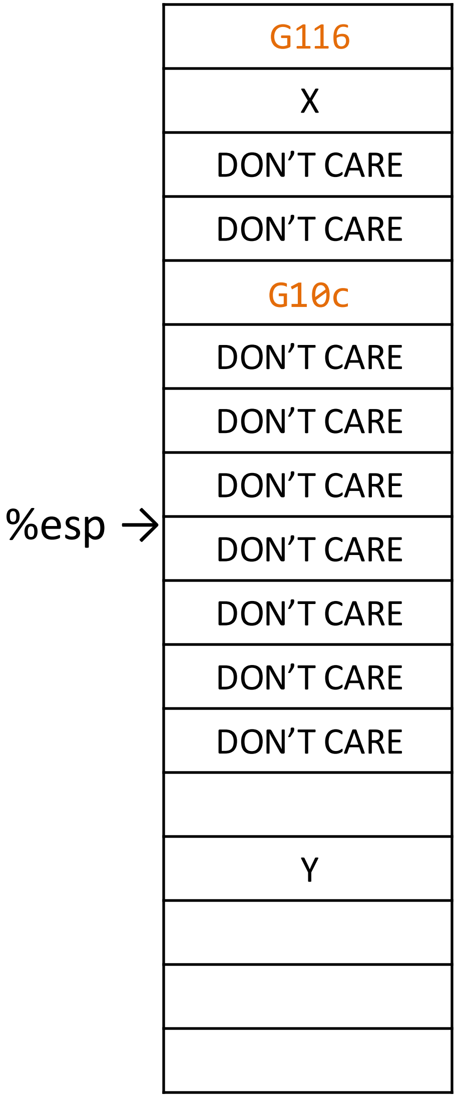
DON'T CARE

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

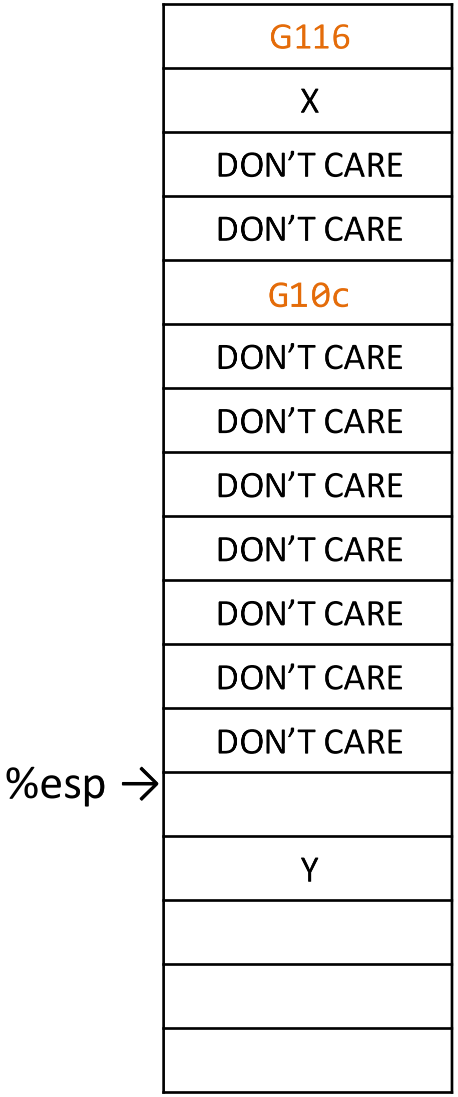
```
G100:  89 ea         mov %ebp, %edx
G102:  31 c0         xor %eax, %eax
G104:  8b 7c 24 04   mov 0x4(%esp), %edi
G108:  89 1f         mov %ebx,(%edi)
G10a:  89 0a         mov %ecx,(%edx)
G10c:  8b 54 24 20   mov 0x20(%esp),%edx
G110:  89 32         mov %esi,(%edx)
G112:  83 c4 0c      add $0xc,%esp
G115: ●5b            pop %ebx
G116:  5e            pop %esi
G117:  5f            pop %edi
G118:  5d            pop %ebp
G119:  c3            ret
```



Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: ●c3             ret
```



G116

X

DON’T CARE

DON’T CARE

G10c

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

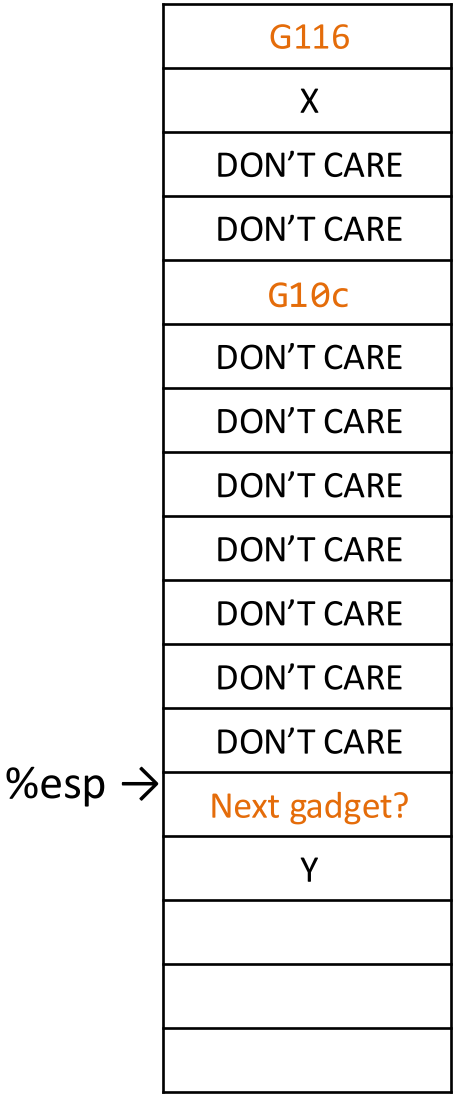
%esp →

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: •c3             ret
```



G116

X

DON'T CARE

DON'T CARE

G10c

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

DON'T CARE

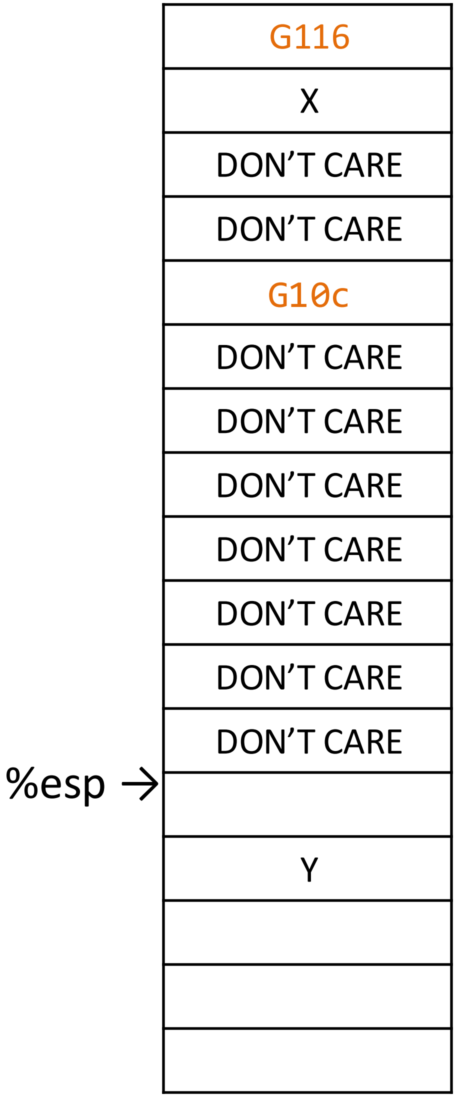
%esp → Next gadget?

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110
  - Skip Y, then next gadget

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: •c3             ret
```



G116

X

DON’T CARE

DON’T CARE

G10c

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

DON’T CARE

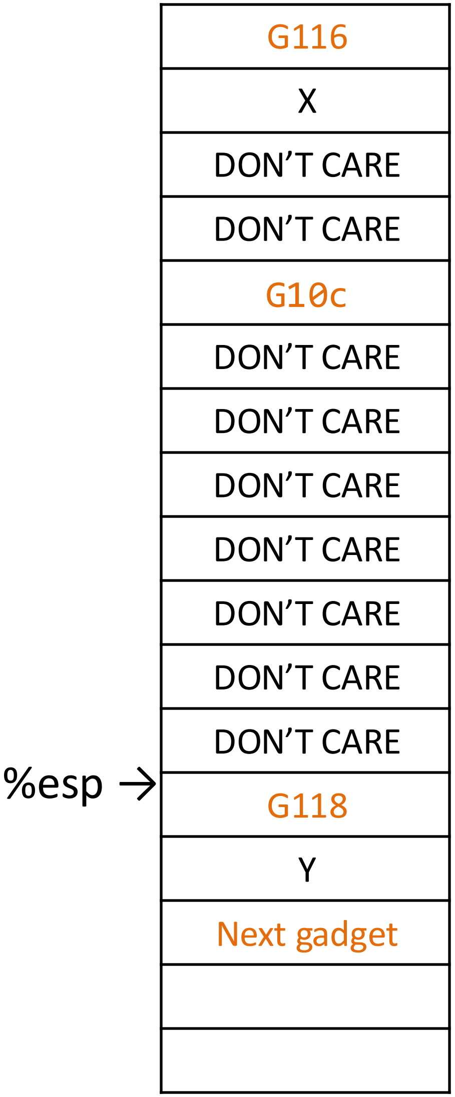
%esp →

Y

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110
  - Skip Y, then next gadget

```
G100:  89 ea         mov %ebp, %edx
G102:  31 c0         xor %eax, %eax
G104:  8b 7c 24 04   mov 0x4(%esp), %edi
G108:  89 1f         mov %ebx,(%edi)
G10a:  89 0a         mov %ecx,(%edx)
G10c:  8b 54 24 20   mov 0x20(%esp),%edx
G110:  89 32         mov %esi,(%edx)
G112:  83 c4 0c      add $0xc,%esp
G115:  5b            pop %ebx
G116:  5e            pop %esi
G117:  5f            pop %edi
G118:  5d            pop %ebp
G119: ●c3            ret
```

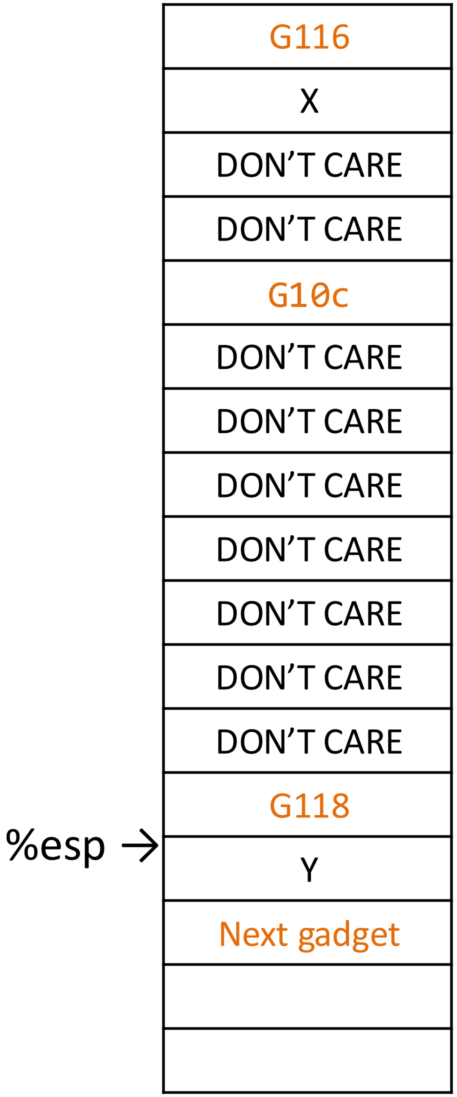


G116
X
DON'T CARE
DON'T CARE
G10c
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
%esp → G118
Y
Next gadget

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110
  - Skip Y, then next gadget

```
G100:  89 ea         mov %ebp, %edx
G102:  31 c0         xor %eax, %eax
G104:  8b 7c 24 04   mov 0x4(%esp), %edi
G108:  89 1f         mov %ebx,(%edi)
G10a:  89 0a         mov %ecx,(%edx)
G10c:  8b 54 24 20   mov 0x20(%esp),%edx
G110:  89 32         mov %esi,(%edx)
G112:  83 c4 0c      add $0xc,%esp
G115:  5b            pop %ebx
G116:  5e            pop %esi
G117:  5f            pop %edi
G118: ●5d            pop %ebp
G119:  c3            ret
```

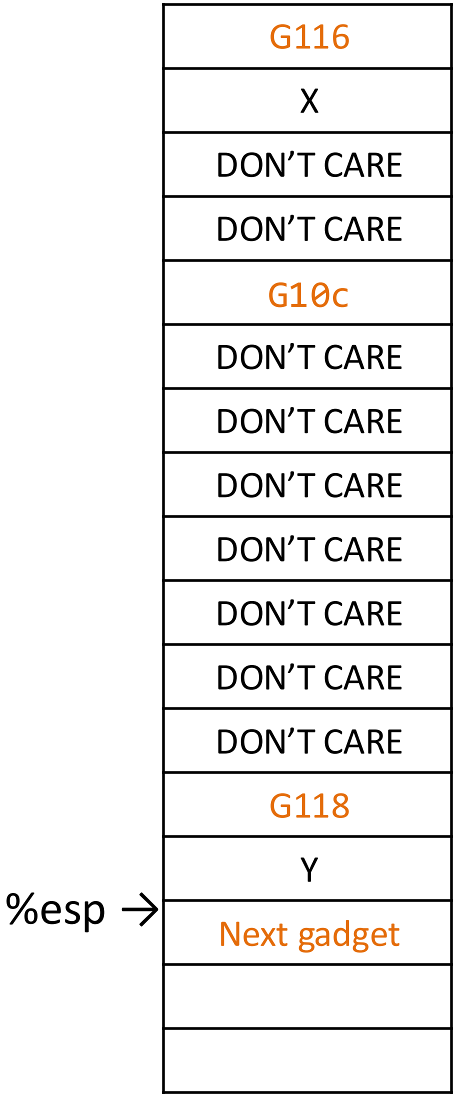


G116
X
DON'T CARE
DON'T CARE
G10c
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
G118
%esp →
Y
Next gadget

Exercise with gadgets from /bin/ls

- Write value X to location Y?
  - Plan: X in %esi, Y in %edx, then G110
  - Skip Y, then next gadget

```
G100:  89 ea          mov %ebp, %edx
G102:  31 c0          xor %eax, %eax
G104:  8b 7c 24 04    mov 0x4(%esp), %edi
G108:  89 1f          mov %ebx,(%edi)
G10a:  89 0a          mov %ecx,(%edx)
G10c:  8b 54 24 20    mov 0x20(%esp),%edx
G110:  89 32          mov %esi,(%edx)
G112:  83 c4 0c       add $0xc,%esp
G115:  5b             pop %ebx
G116:  5e             pop %esi
G117:  5f             pop %edi
G118:  5d             pop %ebp
G119: ●c3             ret
```



G116
X
DON'T CARE
DON'T CARE
G10c
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
DON'T CARE
G118
Y
%esp → Next gadget

Return-Oriented Programming (ROP)

- Gadgets serve the role of instructions
- ROP programs assembled from gadgets
- There are ROP compilers to automate this

	Normal programming	Return-oriented programming
PC	`%eip`	`%esp`
No-op	`nop`	`ret`
Jump	`jmp 4`	`pop %eax/%ebx/…`

Control Flow Hijacking

- Altering control flow of a target program to cause it to do what attacker wants
- 1. Identify a **value** that will be loaded into **PC**
- 2. Overwrite it (to point to shellcode)
  - Deprecate unsafe functions and stack canaries
  - Other forms of control flow hijacking
- 3. Shellcode runs
  - Data Execution Prevention (DEP/W^X)
  - Return-to-libc and Return-Oriented Programming

Address Space Layout Randomization

- Randomize location of stack, heap, and code
  - So that the attacker does not know where the injected shellcode is located
  - Best to randomize on every launch

- Implemented (in some form) on most OSes
  - GCC and Clang: -fPIE
  - Code must be position independent
  - Binaries must be compiled to support ASLR

32-bit PaX ASLR (x86) Base Addresses

Stack:

1	0	1	0	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	0	0	0	0

fixed

random
(24 bits)

zero

Mapped area:

0	1	0	0	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	0	0	0	0	0	0	0	0	0	0	0	0

fixed

random
(16 bits)

zero

Executable code, static variables, and heap:

0	0	0	0	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	0	0	0	0	0	0	0	0	0	0	0	0

fixed

random
(16 bits)

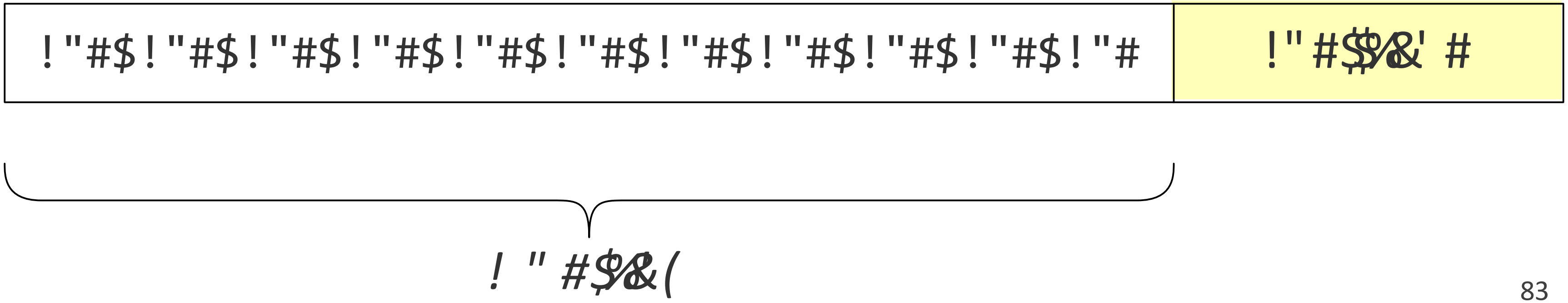
zero

Attacking ASLR

- Brute-force
  - Need to make sure an unsuccessful guess does not crash the target program
  - May be feasible on 32-bit systems: e.g. 2^{16} = 65,536 possible PaX code offsets
- Exploit other vulnerabilities to learn the random memory offset
- Heap spray

Attacking ASLR: Heap Spray

- Suppose attacker can allocate objects on the victim’s machine
- Fill the entire memory/heap with many instances of NOP sled + shellcode
- Overwrite return address / function pointer with arbitrary value → most likely land in a NOP sled



Heap Spray

- Requires attacker to be able allocate objects on the victim’s machine — how?

- Browsers are popular targets
  - Victim user visits a malicious website
  - Malicious site serves JavaScript code to browser
  - Browser is supposed to be a sandbox
  - Malicious JavaScript code performs heap spray and exploits a control flow vulnerability in browser

Summary: Countermeasures

- Altering control flow of a target program to cause it to do what attacker wants
- 1. Identify a **value** that will be loaded into **PC**
- 2. Overwrite it (to point to shellcode)
  - Deprecate unsafe functions and stack canaries
  - Other forms of control flow hijacking
- 3. Shellcode runs
  - Data Execution Prevention (DEP/W^X)
  - Return-to-libc and Return-Oriented Programming
  - Address Space Layout Randomization (ASLR)
  - Heap spray

Summary: Countermeasures

- Combination of defenses more effective
  - DEP, to some extent, forces attacker to use ROP
  - ASLR randomizes where ROP gadgets are located
  - ROP happens on stack; stack canaries prevent some forms of stack overwrites

- To Learn More
  - Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. *SoK: Eternal War in Memory*. 2013.

References/Acknowledgements

- hhttps://users.ece.cmu.edu/~dbrumley/courses/18487-f14/www/powerpoint/
- Return-to-libc demo http://www.securitytube.net/video/258
- http://seclists.org/bugtraq/1997/Aug/63
- https://hovav.net/ucsd/dist/geometry.pdf
- https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf
- http://security.stackexchange.com/questions/20497/stack-overflows-defeating-canaries-aslr-dep-nx
- http://phrack.org/issues/58/4.html