Lecture 06: Control-Flow Hijacking Attacks and Defenses

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Goals for Today

- Learning Objectives:
  - Consider and evaluate defenses to binary exploits
- Announcements, etc:
  - Office hours all the time! M-F 5-7pm, 4405 Siebel
  - MP1 is live!
    - Checkpoint #1: **Due TODAY Sept 9th at 6pm**
    - Checkpoint #2: **Due Sept 18 at 6pm**

Reminder: Please put away devices at the start of class

Control Flow Hijacking

- Control Flow Hijacking: Altering control flow of a target program to cause it to do what attacker wants
- Aleph One stack buffer overflow attack: overwrote function return address on stack to point to shellcode in buffer on stack
  - One of the simplest control flow hijacking attacks
  - Stack buffer overflow vulnerabilities exist today!

Defenses & Counter-Attacks

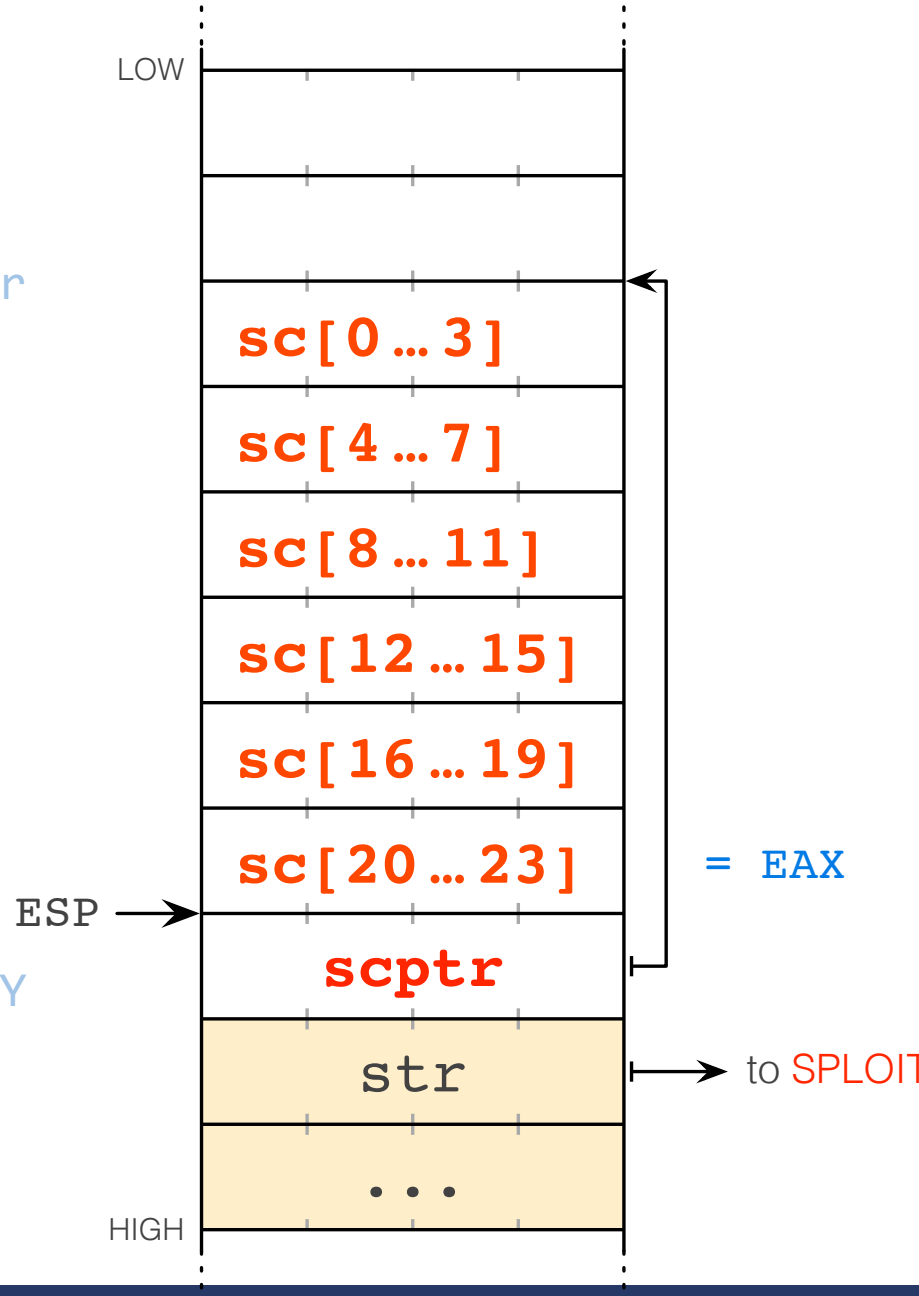
- *Last Class:* Stack canaries
  - Counter-Attack: Other forms of control flow hijacking
- Data Execution Prevention (DEP, W^X)
  - Counter-Attack: Return-to-libc
  - Counter-Attack: Return-Oriented Programming (ROP)
- Address Space Layout Randomization (ASLR)
  - Counter-Attack: Memory disclosure vulnerabilities
  - Counter-Attack: Heap Spray and JIT Spray

Stack Canary

- Idea: detect return address overwrite
  - Place special value (canary) before return address on the stack
- Check canary before executing ret
- If buffer overflows, canary is destroyed before return address
- Can be automatically inserted by compiler
  - –GCC and Clang: `-fstack-protector`

Stack Canary in Action

```
pushl   %CANARY     # not a real register
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
pushl   8(%ebp)
leal    -16(%ebp), %eax
pushl   %eax
call    mystrcpy
addl    $8, %esp
leave
popl    %eax
xorl    %eax, %CANARY
jne     _canary_fail # if %eax ≠ %CANARY
ret
```

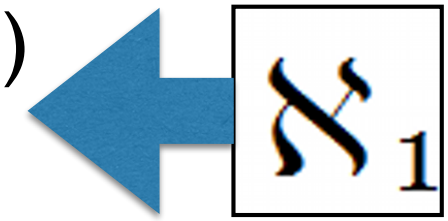


Control Flow Hijacking Attacks

- **Altering control flow** of a target program to cause it **to do what attacker wants**
- Not "injecting code to do what attacker wants"
- Alternate strategy:
  1. Identify a code pointer that is eventually loaded into PC
  2. Overwrite code pointer (memory write vulnerability)
  3. Divert PC to code that will do useful work (for attacker)
- In Aleph One attack:
  - Code pointer: return address on stack
  - Memory write vulnerability: buffer overflow
  - Divert PC to shellcode in stack buffer

Code Pointers

- Return from procedure (ret instruction)
  - Return addresses on call stack



- C-style function pointers
  - Driver function dispatch tables
  - Some libc functions
- C++ virtual method calls
  - Special case of function pointers

Class Polymorphism

- C++ classes are polymorphic: pointer to an object may point to an instance of the class or any of its derived classes (subclasses)

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

class Circle : Shape {
    float r;
    Circle(float r);
};
```

```
class Rect : Shape {
    float w, h;
    Rect(float w, h);
};
```

Class Polymorphism

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

class Circle : Shape {
    float r;
    Circle(float r);
};

Circle::Circle(float r) {
    this.r = r;
}

float Circle::area(void) {
    return PI * r * r;
}
```

```
class Rect : Shape {
    float w, h;
    Rect(float w, h);
};

Rect::Rect(float w, float h) {
    this.w = w;
    this.h = h;
}

float Rect::area(void) {
    return w * h;
}
```

Class Polymorphism

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

class Circle : Shape {
    float r;
    Circle(float r);
};

Circle::Circle(float r) {
    this.r = r;
}

float Circle::area(void) {
    return PI * r * r;
}
```

```
class Rect : Shape {
    float w, h;
    Rect(float w, h);
};

Rect::Rect(float w, float h) {
    this.w = w;
    this.h = h;
}

float Rect::area(void) {
    return w * h;
}
```

```
Shape* s1 = new Rect(3,4);
Shape* s2 = new Circle(5);
cout << s1.area(); // calls Rect::area()
cout << s2.area(); // calls Circle::area()
```

Virtual Method Dispatch

```
Shape* s1 = new Rect(3,4);
Shape* s2 = new Circle(5);
cout << s1.area(); // calls Rect::area()
cout << s2.area(); // calls Circle::area()
```

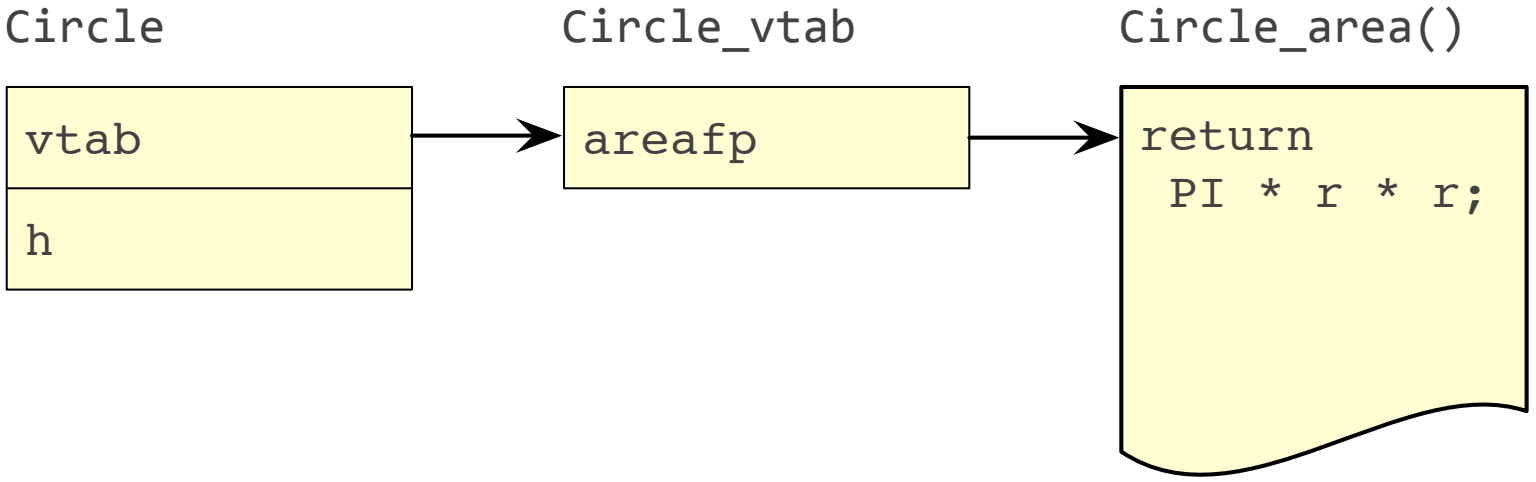
- Compiler needs to know which area() method to call in each case
- Virtual classes have additional invisible member variable: virtual function table pointer
- Compiler uses virtual function table to call the right virtual method during a virtual method call

C++

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


s1->area();
```

C equivalent

```
struct Shape {
    struct Shape_vtab * vtab;
};

struct Shape_vtab {
    float (areafp*)(Shape * this);
};

s1->vtab->areafp(s1);
```



C++

```
Circle::Circle(float r) {
    this.r = r;
}

Circle::area(void) {
    return PI * r * r;
}
```

C equivalent

```
struct Circle {
    struct Circle_vtab * vtab;
};

struct Circle_vtab {
    float (areafp*)(Shape * this);
};

float Circle_area(Circle * this) {
    return PI * this->r * this->r;
}

const struct
Shape_vtab circle_vtab = {
    &Circle_area
};

void Circle_ctor (
    Circle * this, float r) {
    this->vtab = &circle_vtab;
    this->r = r;
}
```

C++

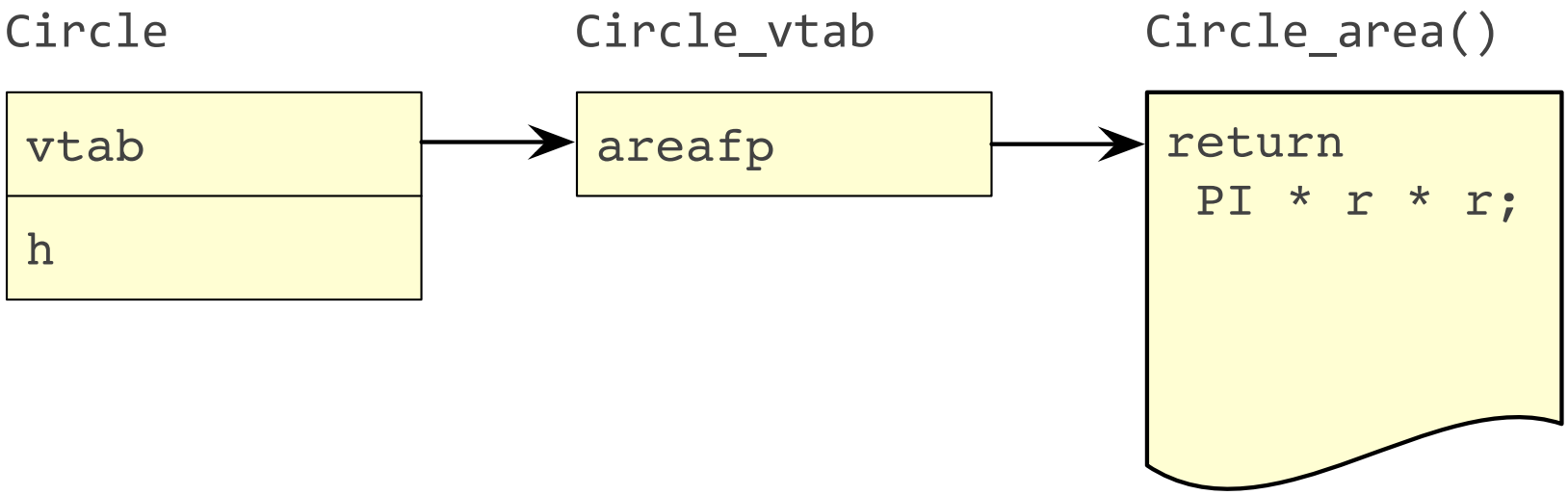
```
Circle::Circle(float r) {
    this.r = r;
}

Circle::area(void) {
    return PI * r * r;
}
```

C equivalent

```
struct Circle {
    struct Circle_vtab * vtab;
};

struct Circle_vtab {
    float (areafp*)(Shape * this);
};

float Circle_area(Circle * this) {
    return PI * this->r * this->r;
}

const struct
Shape_vtab circle_vtab = {
    &Circle_area
};

void Circle_ctor (
    Circle * this, float r) {
    this->vtab = &circle_vtab;
    this->r = r;
}
```

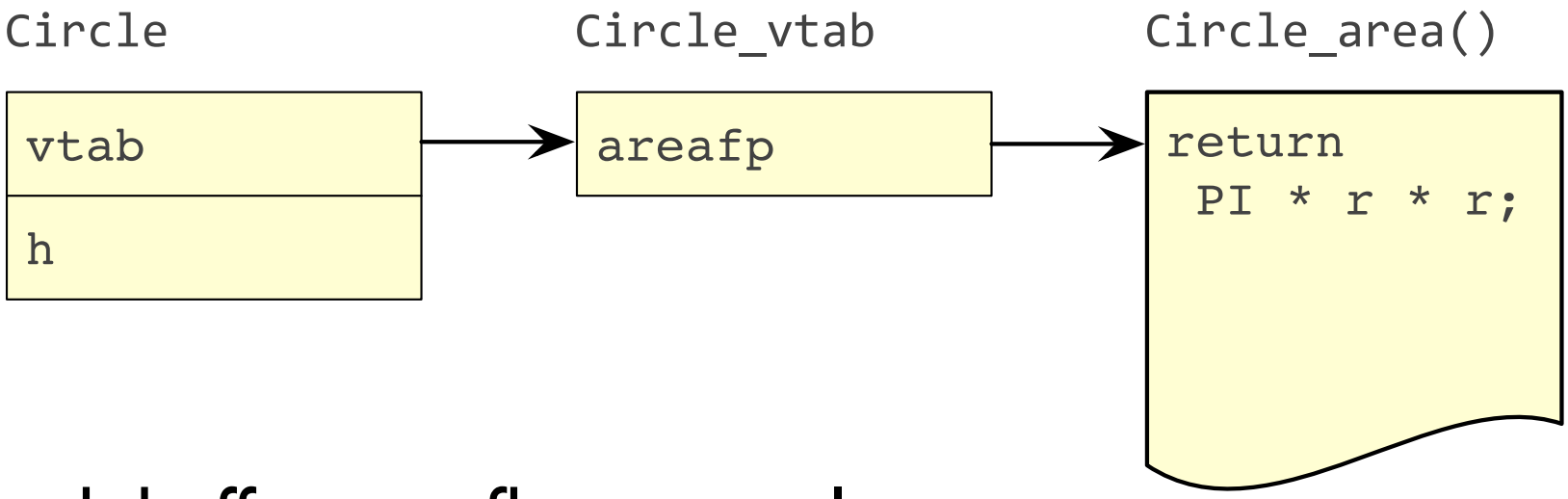
function pointer!

VTable Hijacking



- Overwrite pointers in virtual function table
- Overwrite virtual function table pointer (vtab)
  - Point to your own virtual function table

VTable Hijacking



- Stack buffer overflows now less common
  - Easy-to-find bugs getting fixed
  - Use of unsafe functions deprecated
- VTable hijacking has grown in popularity
  - Common attack vector for many modern control flow hijacking exploits

Software Vulnerability Exploitation Trends (Microsoft, 2013)

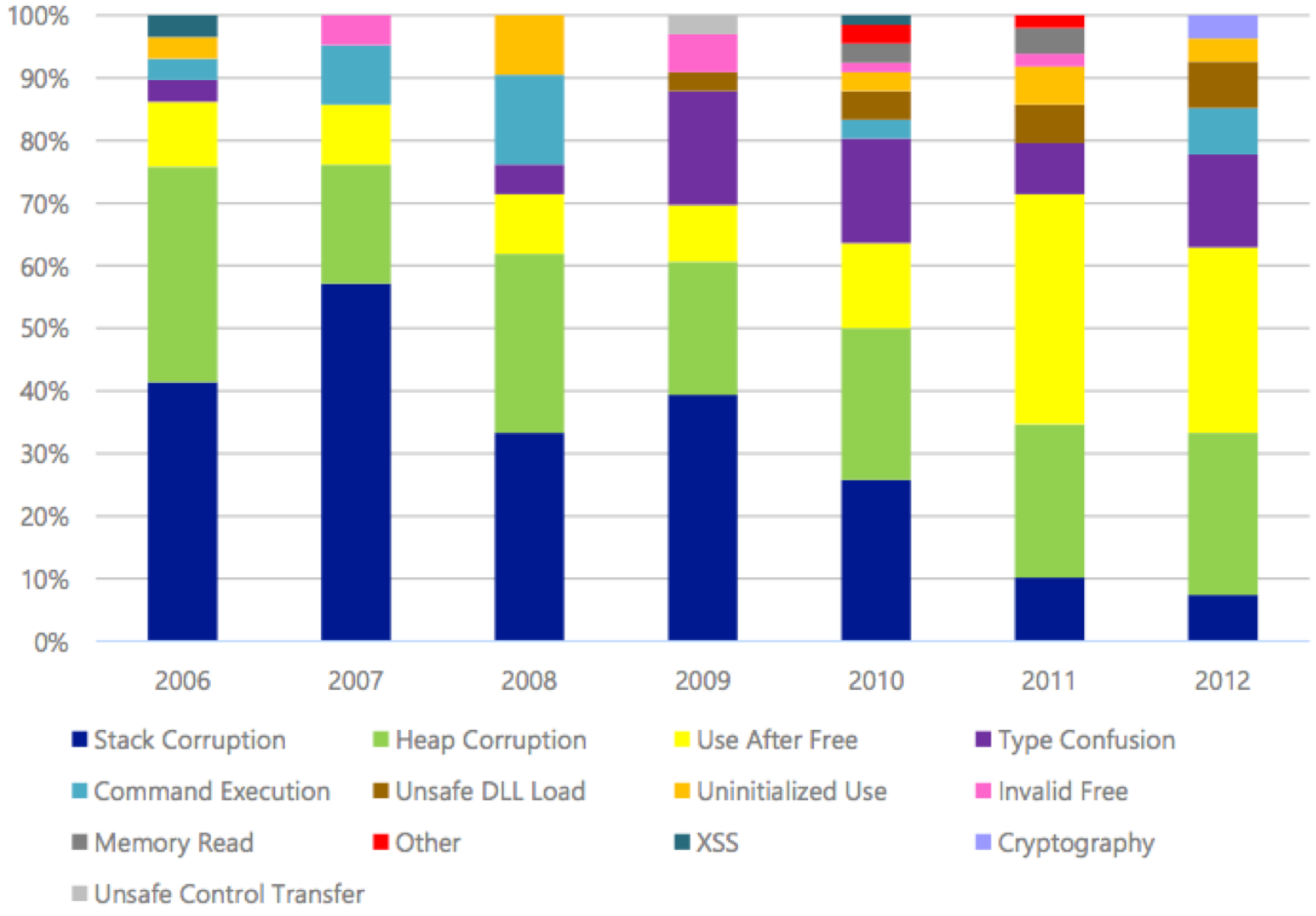


Figure 5. The distribution of CVE vulnerability classes for CVEs that are known to have been exploited

Memory Write Vulnerabilities

- Integer overflow and conversion
- Use-after-free and double-free
- Format string vulnerabilities
- Data structure consistency

Signedness Bugs

- What's wrong with this code??
- What happens when you call `copy_something(mybuf, -1)`?

```
extern void * memcpy(void * dst, const void * src, size_t n);

int copy_something(char *buf, int len) {
    char kbuf[800];

    if(len > sizeof(kbuf)) {        // always true
        return -1;
    }

    return memcpy(kbuf, buf, len);  // len cast to size_t
}                                   // which is unsigned
```

Based on "Basic Integer Overflows" by blexim, *Phrack* volume 0x0b, issue 0x3c, phile #0x0a of 0x10

Signedness Bugs

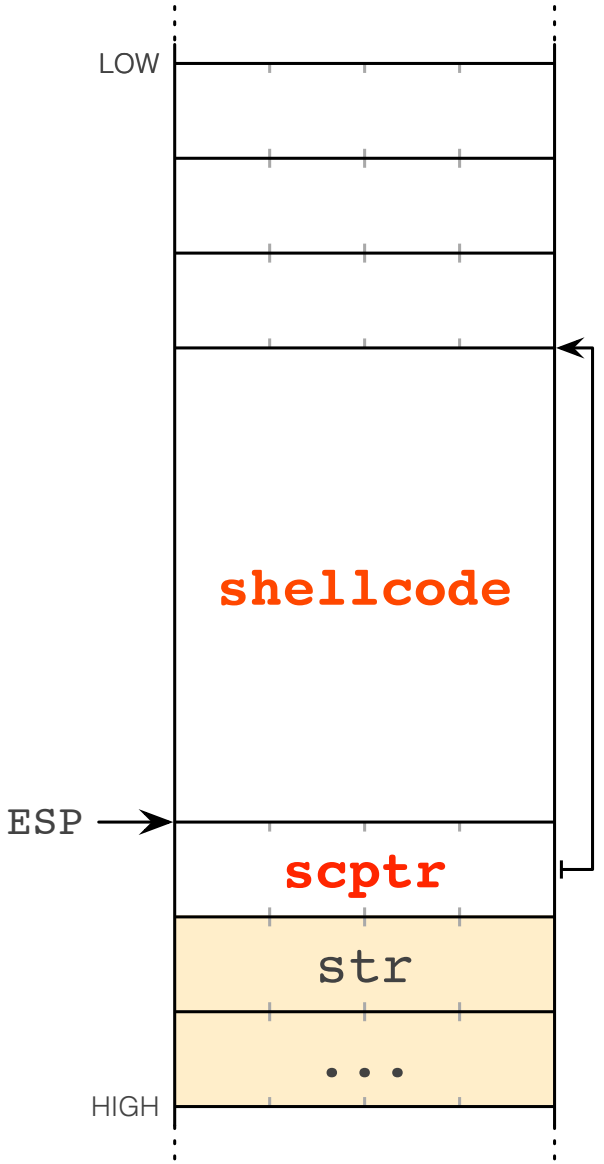
- What's wrong with this code??
- What happens when you call `copy_something(mybuf, -1)`?

```
extern void * memcpy(void * dst, const void * src, size_t n);

int copy_something(char *buf, int len) {
    char kbuf[800];

    if(len > sizeof(kbuf)) {        // always true
        return -1;
    }

    return memcpy(kbuf, buf, (size_t)(-1));
}
```

Based on "Basic Integer Overflows" by blexim, *Phrack* volume 0x0b, issue 0x3c, phile #0x0a of 0x10

Defense: Data Execution Prevention

- Observation: attacker jumped to shellcode on stack
  - Shellcode was in buffer that overflowed and overwrote RA
- But instructions normally never executed on from the stack
  - What if we prevent execution from stack?



Data Execution Prevention (DEP)

- Text (code) region is executable, not writable
- Data memory regions are non-executable
- Also called W^X (write xor execute)
  - Memory is writable or executable, never both
- NX bit (no execute) in hardware page tables
  - Intel: XD (execute disable) bit
  - AMD: Enhanced Virus Protection
  - ARM: XN (execute never) bit

Data Execution Prevention (DEP)

- Attempting to execute instructions from stack, heap, or static buffers causes exception
- No way to execute shellcode… right?

Data Execution Prevention (DEP)

- Attempting to execute instructions from stack, heap, or static buffers causes exception
- No way to execute shellcode… right?
- We could still execute code that is already there, but is that useful?
- Isn't application code "good code"?
- Can "good code" do "bad things"?

Return-to-Libc

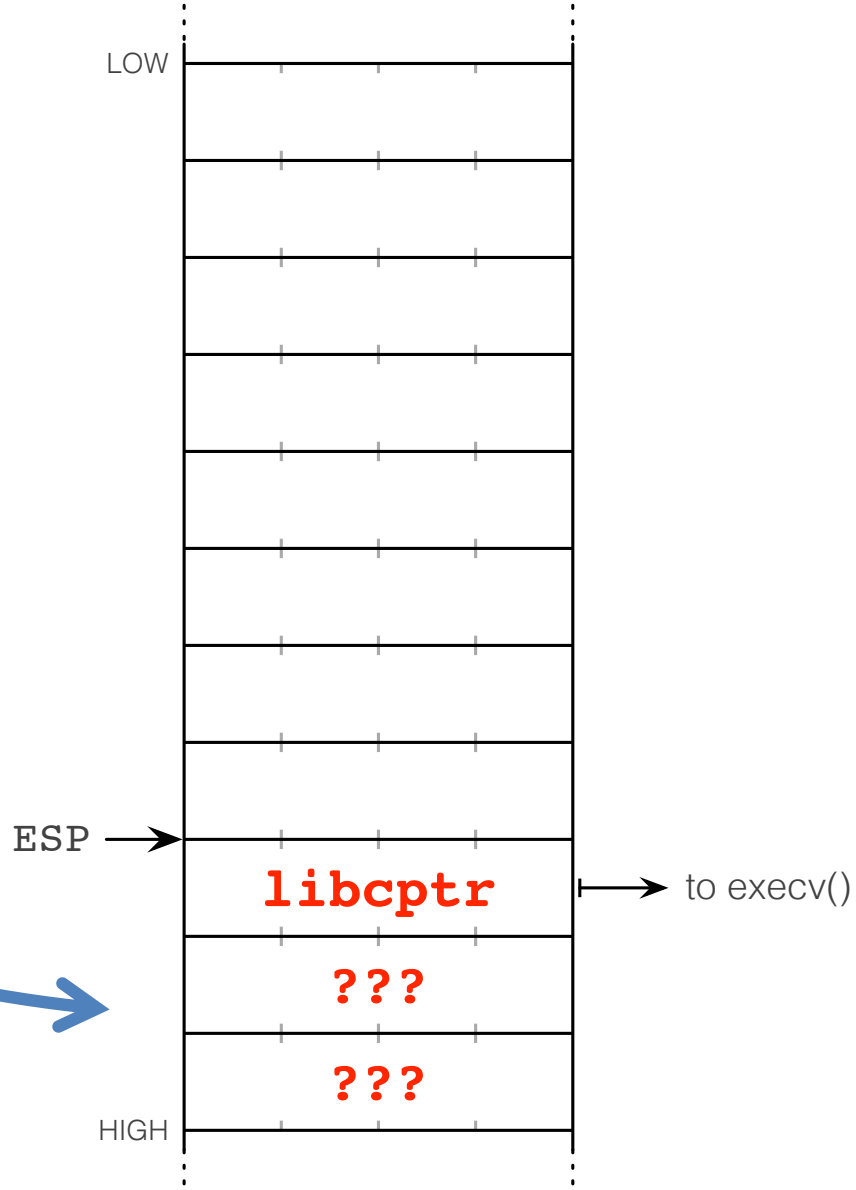
- Aleph One wanted to execute a shell
  - `execv` system call with "/bin/sh" as argument:

```
int execv ( const char *path,
            char *const argv[]);
```

- Calling execv in C actually calls a wrapper function in libc that makes the system call
- But (what if) libc is already loaded into memory!

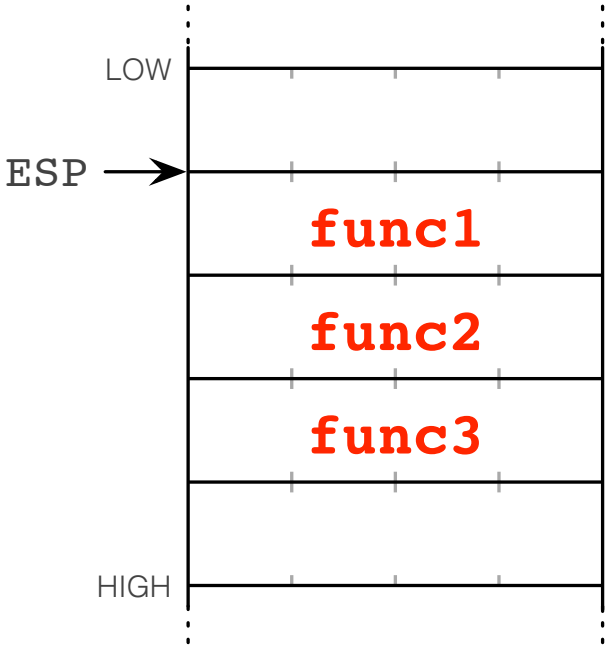
Return-to-Libc

- Overwrite return address with address of execv
- Executing ret will transfer control to execv function that will make system call
- Need to set up arguments to execv on the stack



Chaining Function Calls

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

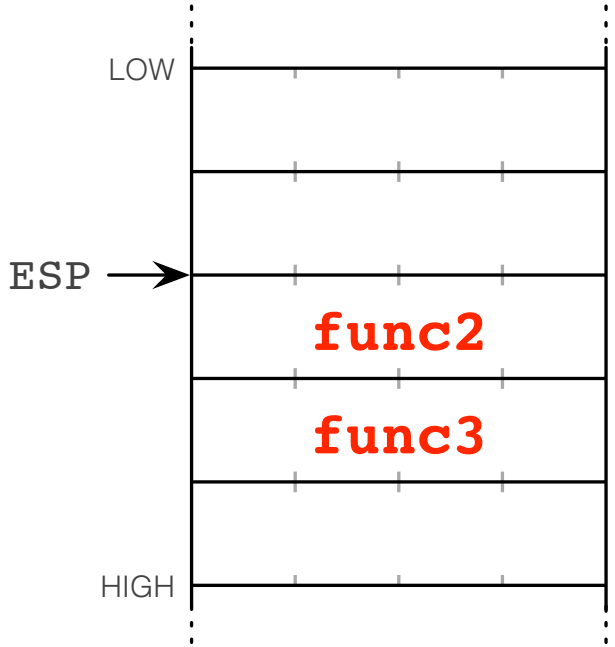



```
func0:
 ...
•ret

func1:
 ...
 ret
```

```
func2:
 ...
 ret

func3:
 ...
 ret
```

Chaining Function Calls

- What happens after we jump to a function via `ret`?
- ESP points to address on stack of next function to run

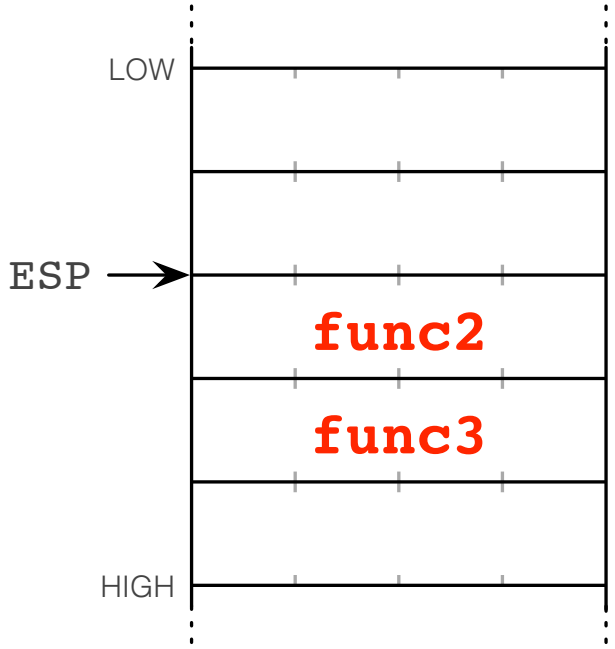



```
func0:
 ...
 ret

func1:
•...
 ret
```

```
func2:
 ...
 ret

func3:
 ...
 ret
```

Chaining Function Calls

- ESP points to address of next function to execute
- When `func1` finishes it returns to `func2`

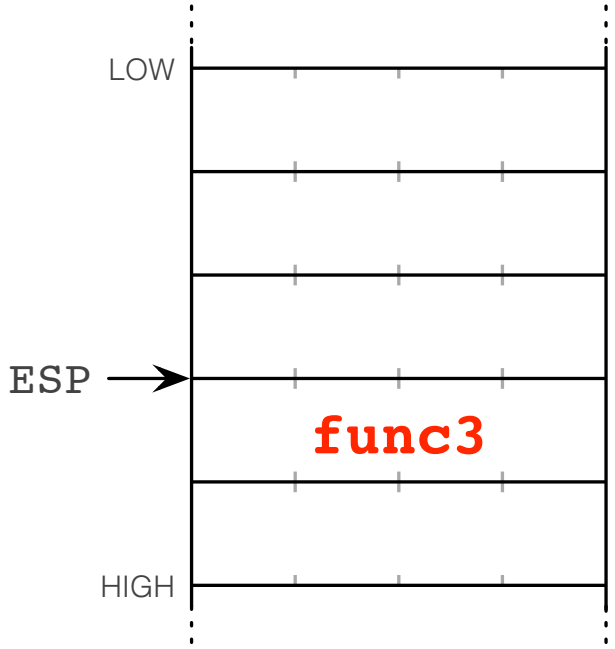



```
func0:
 ...
 ret

func1:
 ...
•ret
```

```
func2:
 ...
 ret

func3:
 ...
 ret
```

Chaining Function Calls

- ESP points to address of next function to execute
- When `func1` finishes it returns to `func2`, and so on

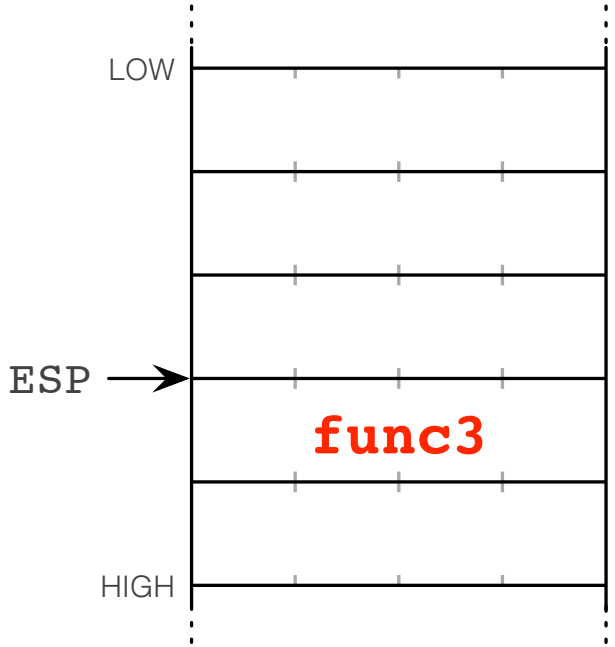


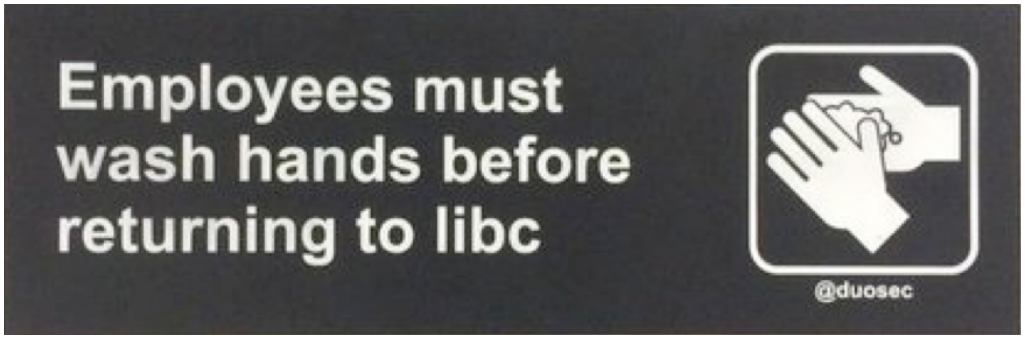

```
func0:
 ...
 ret

func1:
 ...
 ret
```

```
func2:
•...
 ret

func3:
 ...
 ret
```

Chaining Function Calls

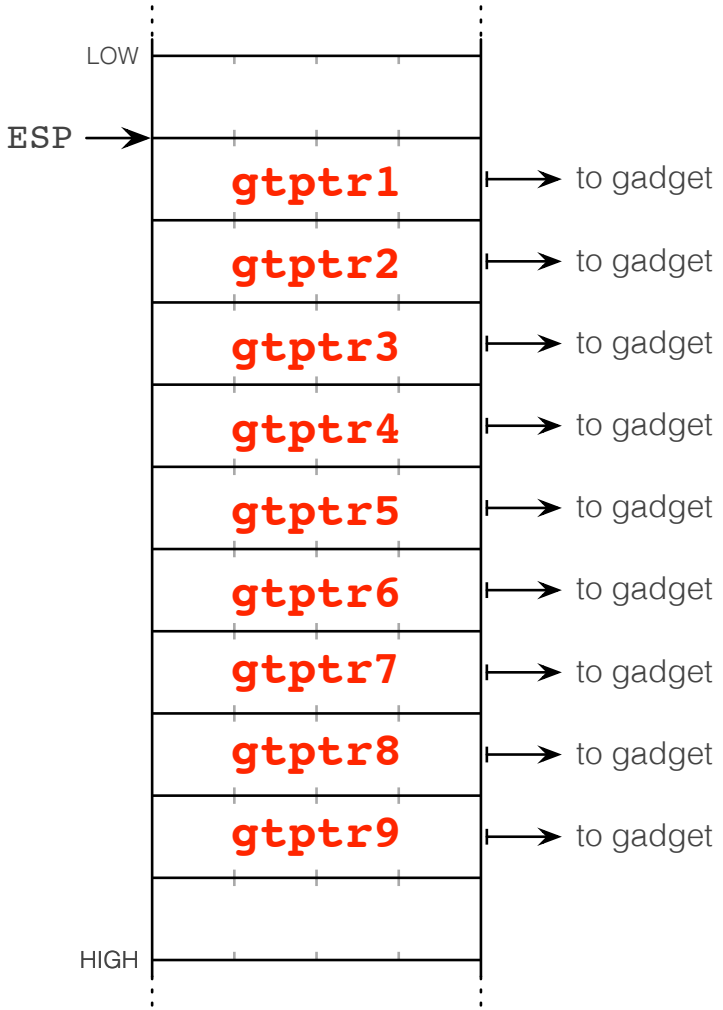
- ESP points to address of next function to execute
- When `func1` finishes it returns to `func2`, and so on
- Functions will be `ret`-called in top-to-bottom stack order
- This allows us to chain multiple functions





Return-Oriented Programming (ROP)

- **Observation**: We don't actually need to ret-call a real function…
- Any code that does something useful and ends in ret instruction will work
- Does not even need to be code emitted by compiler, just executable memory!
- Fragments of useful code ending in ret are called **gadgets**



ROP Workflow

- Dump executable portions of target program
- Identify byte sequences ending in 0xC3 (ret)
- Figure out what each gadget does
  - https://onlinedisassembler.com/
- Chain together useful gadgets

```
0x0808e099 : pop ebx ; pop esi ; sub eax, 0x10 ; pop edi ; ret
0x08052131 : pop ebx ; pop esi ; sub eax, edi ; pop edi ; pop ebp ; ret
0x0808da30 : pop ebx ; pop esi ; xor eax, eax ; pop edi ; ret
0x080e2ed3 : pop ebx ; push cs ; adc al, 0x41 ; ret
0x080e3779 : pop ebx ; push cs ; adc al, 0x43 ; ret
0x080481a9 : pop ebx ; ret
0x080d7ebc : pop ebx ; ret 0x6f9
0x08099947 : pop ebx ; ret 8
0x080d539d : pop ebx ; retf
0x080a7b51 : pop ebx ; sar eax, 2 ; pop esi ; pop edi ; ret
0x0806c357 : pop ebx ; sub eax, 1 ; pop esi ; pop edi ; ret
0x0808c3c0 : pop ebx ; sub eax, esi ; pop esi ; pop edi ; pop ebp ; ret
0x0808ecc6 : pop ebx ; sub eax, esi ; pop esi ; pop edi ; ret
0x080b52a8 : pop ebx ; xor eax, eax ; pop esi ; pop edi ; pop ebp ; ret
0x08068e20 : pop ebx ; xor eax, eax ; pop esi ; pop edi ; ret
0x08063340 : pop ebx ; xor eax, eax ; ret
0x0804af03 : pop ecx ; add dword ptr [eax], eax ; add byte ptr [edi], cl ; retf 0x5389
```

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
  ...
Conte
  ...
 804a530 2404a1   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.
```

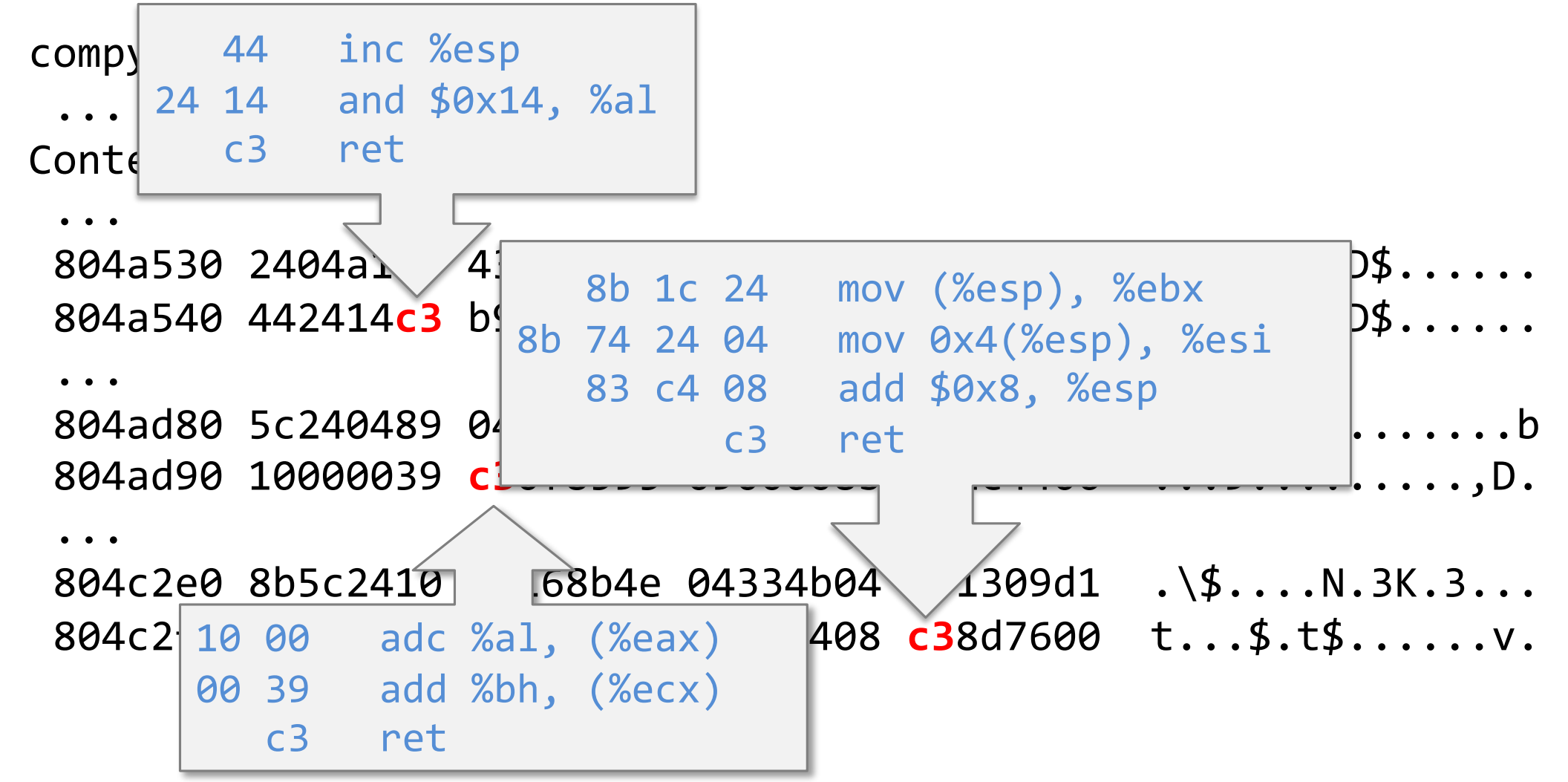
```
   44    inc %esp
24 14    and $0x14, %al
   c3    ret
```

Finding Gadgets

```
compy
  ...
Conte
  ...
 804a530 2404a1   430608c7 44241800 000000c7   $.. C...D$......
 804a540 442414c3 b90508c7 442410d3 b9050889   D$.....D$......
  ...
 804ad80 5c240489 0424e855 8b000085 c00f8462   \$...$.U.......b
 804ad90 10000039 c30f8595 0900008b 0d2c4406   ...9.........,D.
  ...
 804c2e0 8b5c2410    68b4e 04334b04 331309d1   .\$....N.3K.3...
 804c2  10 00    adc %al, (%eax)    408 c38d7600   t...$.t$......v.
        00 39    add %bh, (%ecx)
           c3    ret
```

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

Finding Gadgets



Finding Gadgets

- Gadgets may be code that is not generated by the compiler
  - This gadget is a byte sequence that was *not* intended to be code

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

- Any suffix of a gadget is also a gadget

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

Other gadgets from /bin/ls

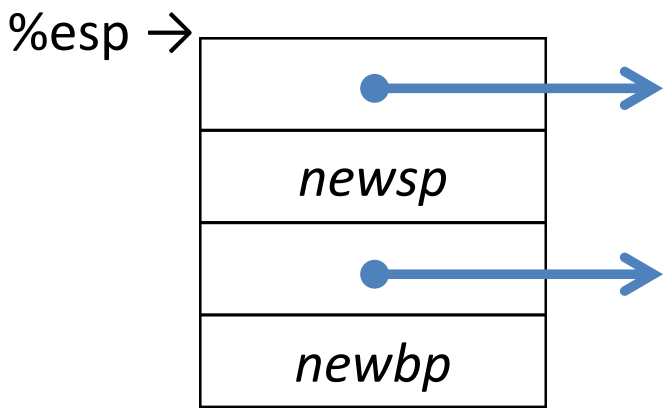
- What can I do with these?

```
 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

c9                 leave
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
```

Other gadgets from /bin/ls

- Set ESP and EBP to arbitrary value




```
•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
```

Other gadgets from /bin/ls

- Write four byte value *X* to location *Y*?

```
G000: 8b 1c 24          mov (%esp), %ebx
G003: 8b 74 24 04       mov 0x4(%esp), %esi
G007: 83 c4 08          add $0x8, %esp
G00a: c3                ret

G100: b8 01 00 00 00    mov $0x1, %eax
G105: 8b 34 24          mov (%esp), %esi
G108: 8b 7c 24 04       mov 0x4(%esp), %edi
G10c: 83 c4 08          add $0x8, %esp
G10f: c3                ret

G200: c9                leave
G201: c3                ret
```

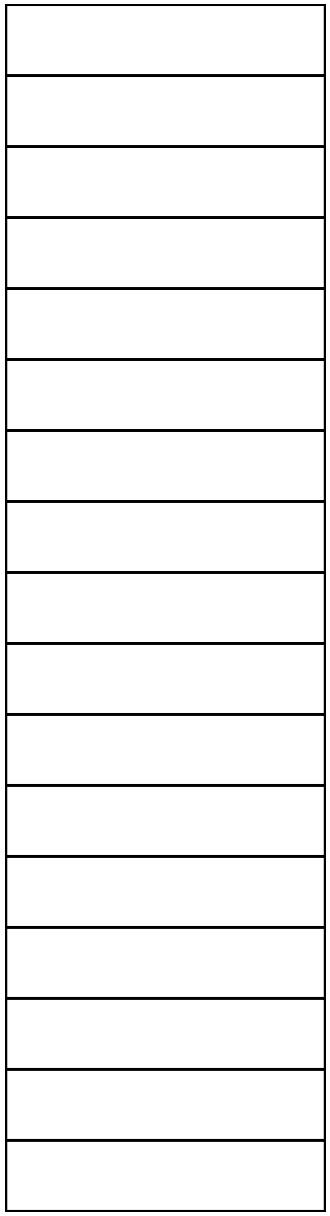
```
G100: 89 ea             mov %ebp, %edx
G102: 31 c0             xor %eax, %eax
G104: 8b 7c 24 04       mov 0x4(%esp),
G108:%edi
G10a: 89 1f             mov %ebx,(%edi)
G10c: 89 0a             mov %ecx,(%edx)
G110: 8b 54 24 20       mov
G112:0x20(%esp),%edx
G115: 89 32             mov %esi,(%edx)
G116: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

Other gadgets from /bin/ls

- Write four byte value *X* to location *Y*?

```
G000:  8b 1c 24          mov (%esp), %ebx
G003:  8b 74 24 04       mov 0x4(%esp), %esi
G007:  83 c4 08          add $0x8, %esp
G00a:  c3                ret

G100:  b8 01 00 00 00    mov $0x1, %eax
G105:  8b 34 24          mov (%esp), %esi
G108:  8b 7c 24 04       mov 0x4(%esp), %edi
G10c:  83 c4 08          add $0x8, %esp
G10f:  c3                ret

G200:  c9                leave
G201:  c3                ret
```

```
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
```



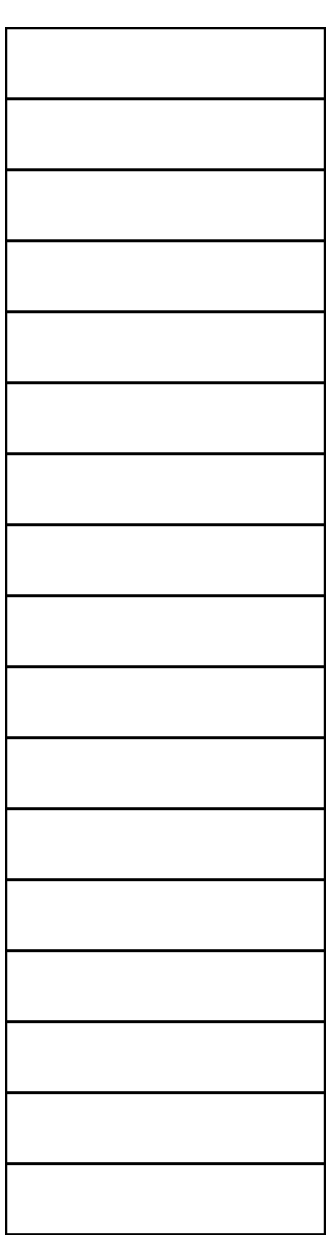
Other gadgets from /bin/ls

- Write four byte value *X* to location *Y*?

```
G000:  8b 1c 24          mov (%esp), %ebx
G003:  8b 74 24 04       mov 0x4(%esp), %esi
G007:  83 c4 08          add $0x8, %esp
G00a:  c3                ret

G100:  b8 01 00 00 00    mov $0x1, %eax
G105:  8b 34 24          mov (%esp), %esi
G108:  8b 7c 24 04       mov 0x4(%esp), %edi
G10c:  83 c4 08          add $0x8, %esp
G10f:  c3                ret

G200:  c9                leave
G201:  c3                ret
```

```
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
```

use this to write to memory



Other gadgets from /bin/ls

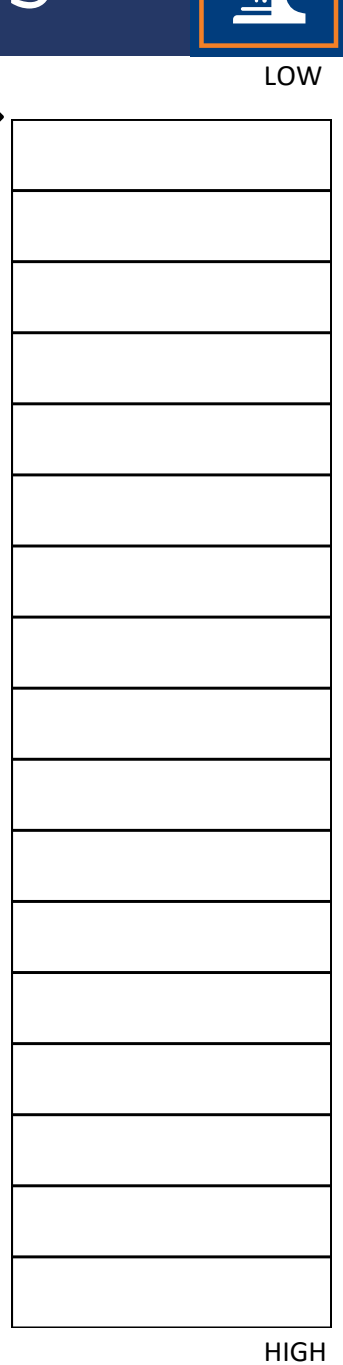
- Write four byte value *X* to location *Y*?

Want:

- *X* in %esi
- *Y* in %edx

```
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
```

use this to write to memory



Other gadgets from /bin/ls

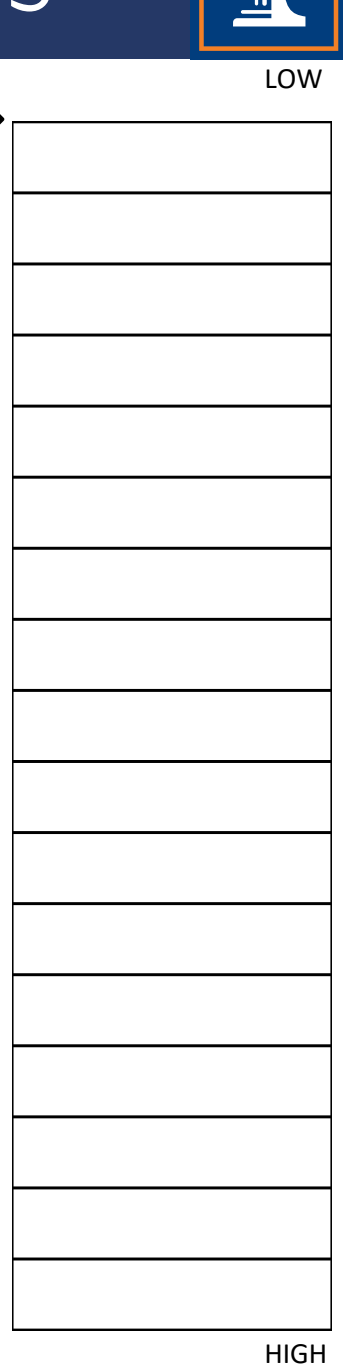
- Write four byte value *X* to location *Y*?

Want:

- *X* in %esi
- *Y* in %edx

```
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
```

use this to load %esi



Other gadgets from /bin/ls

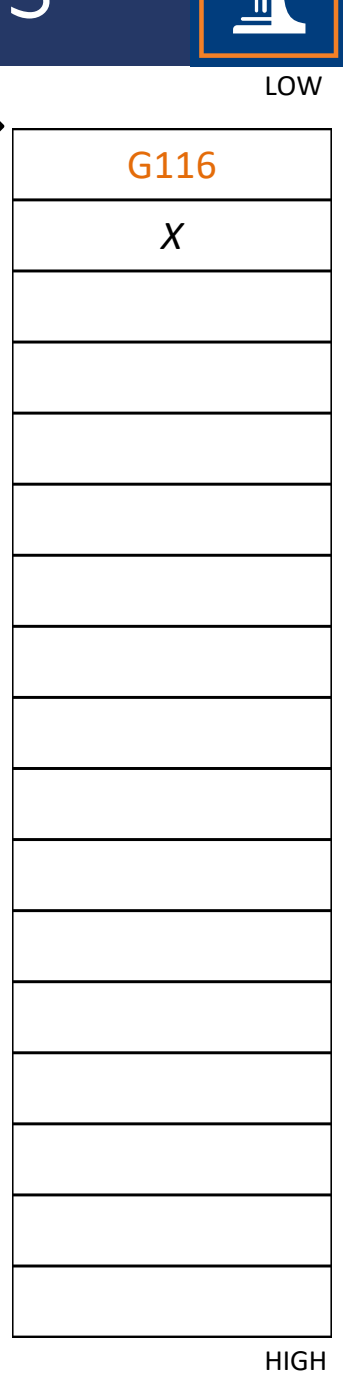
- Write four byte value *X* to location *Y*?

Want:

- *X* in %esi
- *Y* in %edx

```
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
```

use this to load %esi



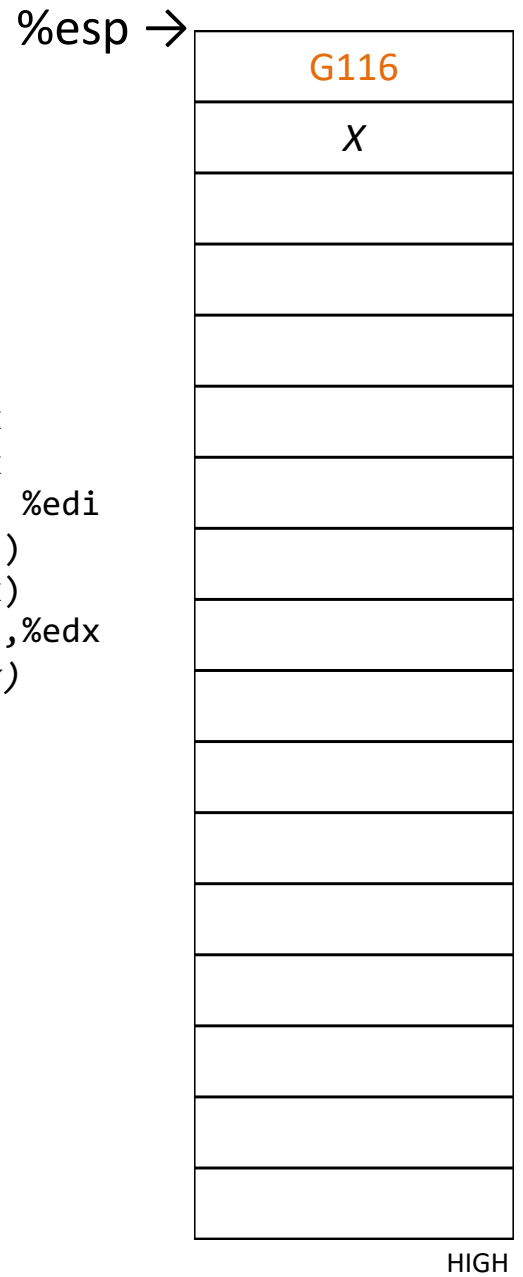
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi
- *Y* in %edx

```
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


       c3             •ret
```



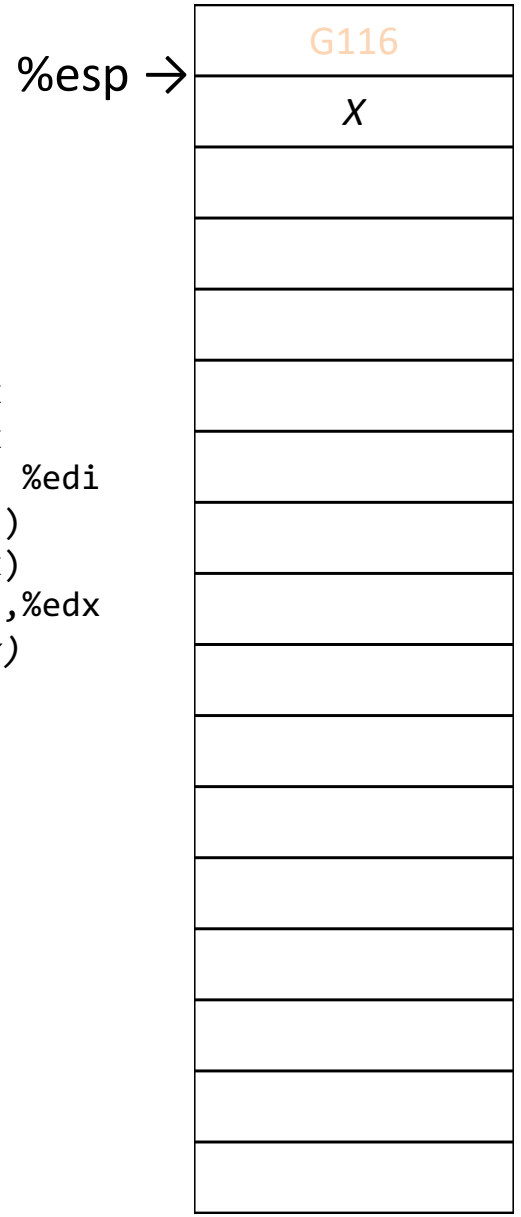
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi
- *Y* in %edx

```
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


       c3             ret
```

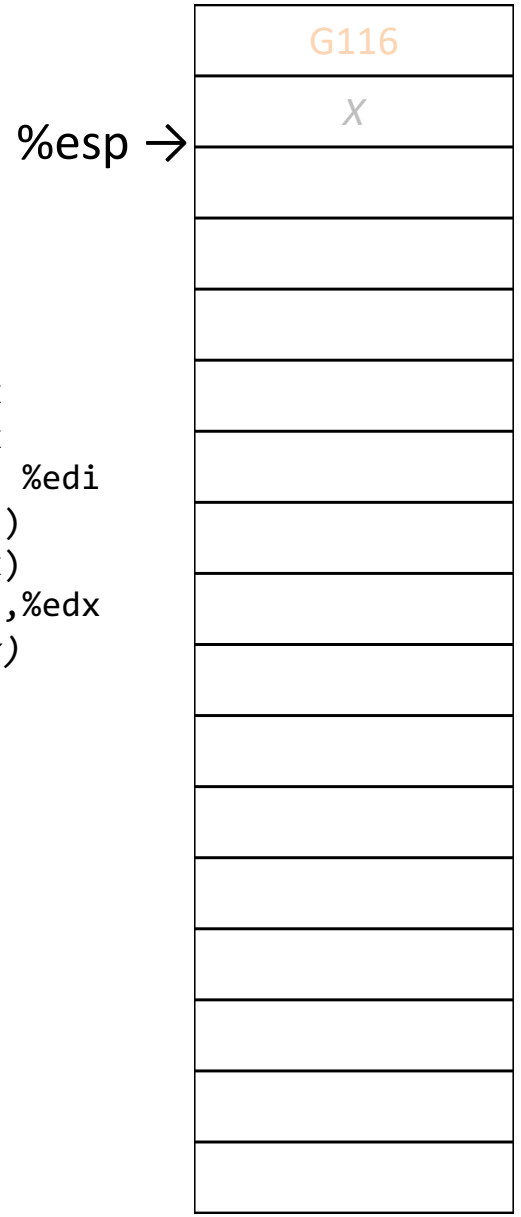


Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:
- *X* in %esi ✓
- *Y* in %edx

```
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
```



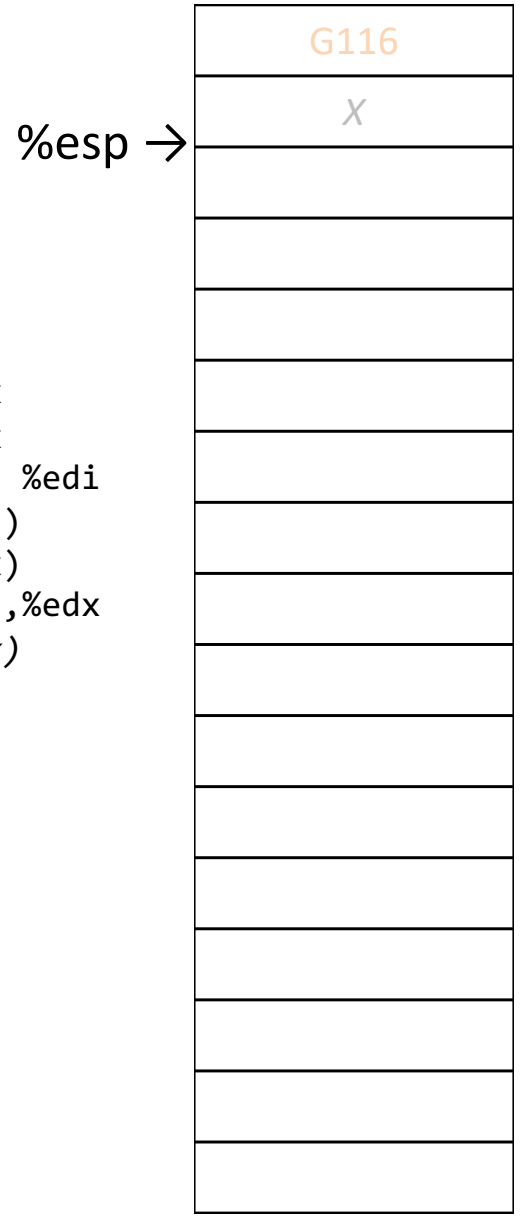
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:
- *X* in %esi ✓
- *Y* in %edx

```
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
```

will execute also



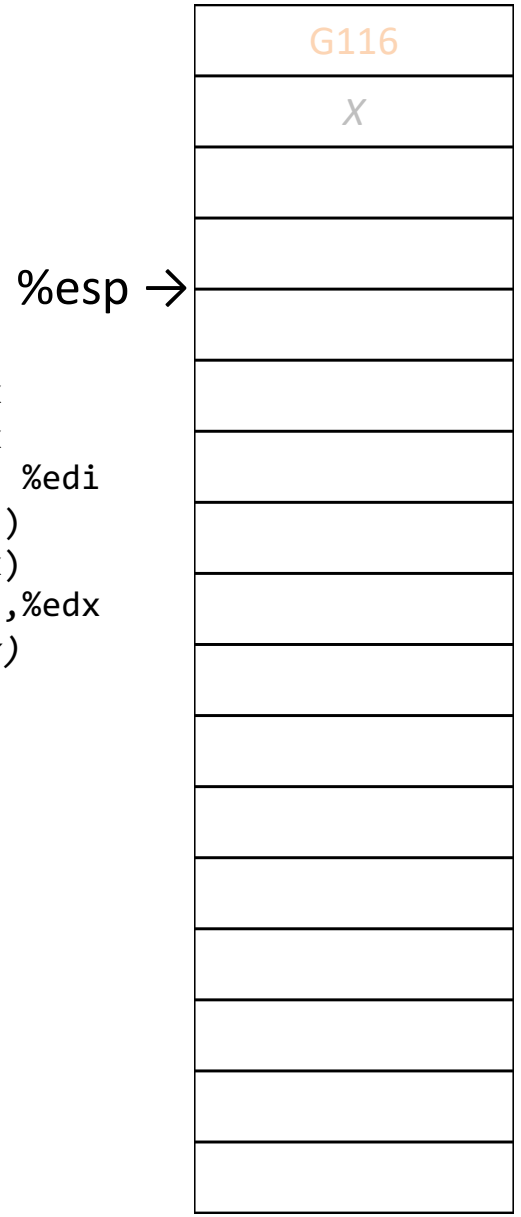
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx

```
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
```



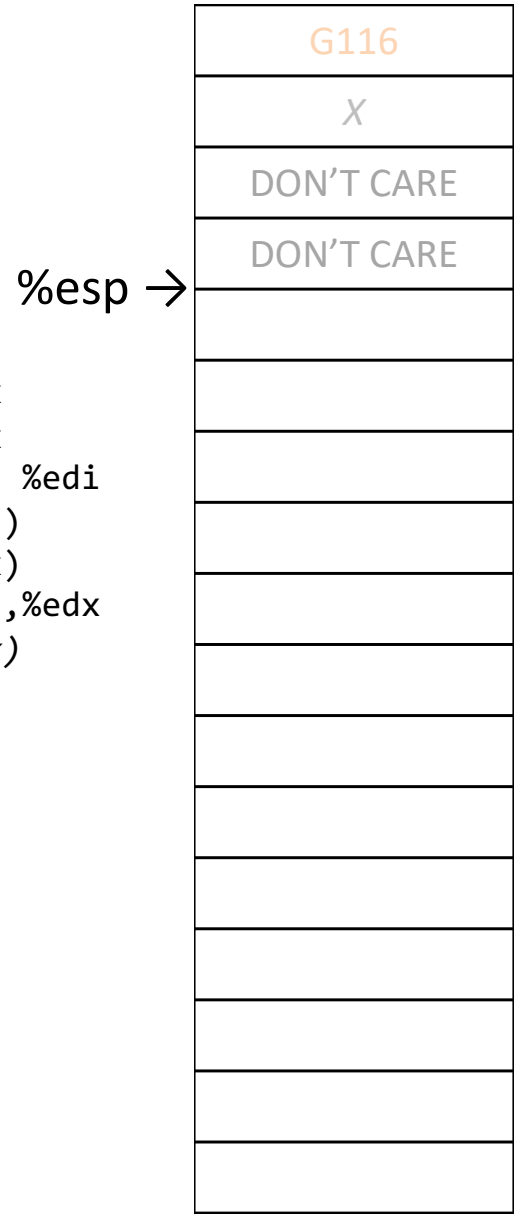
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx

```
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
```



Other gadgets from /bin/ls

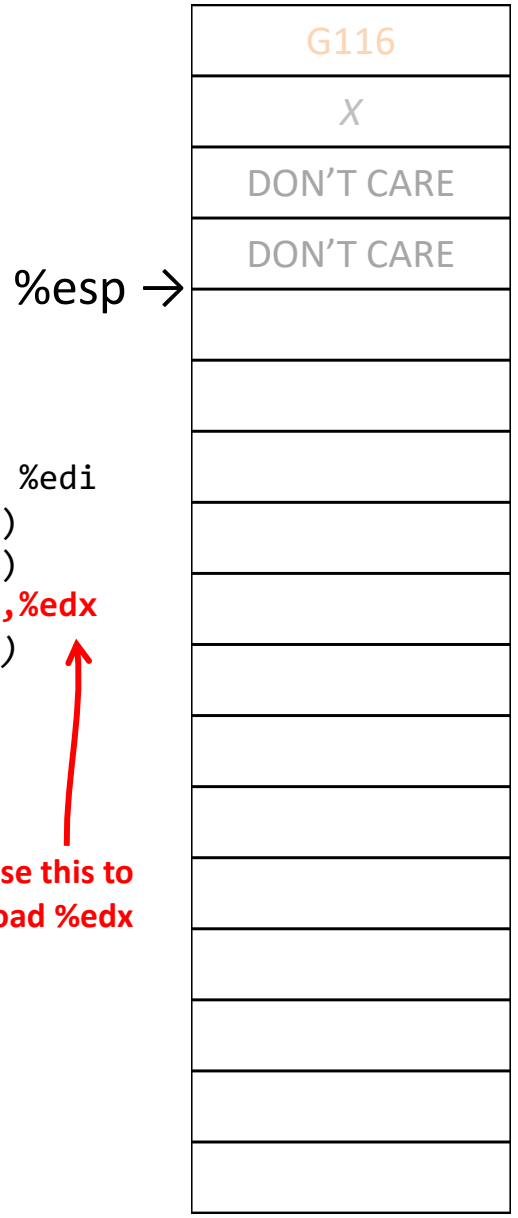
- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx

```
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
```

use this to load %edx



Other gadgets from /bin/ls

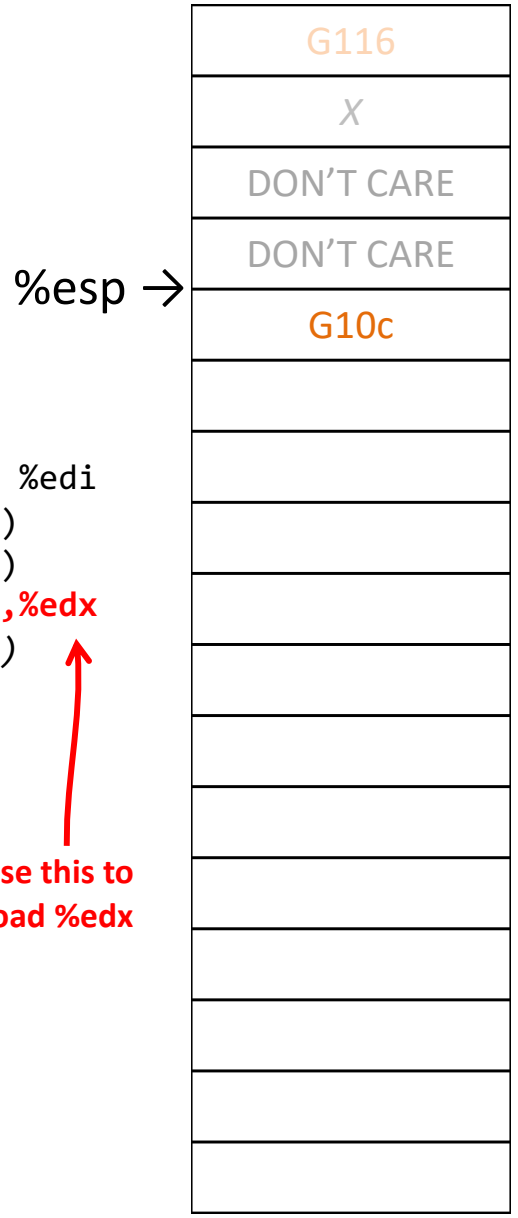
- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx

```
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
```

use this to load %edx



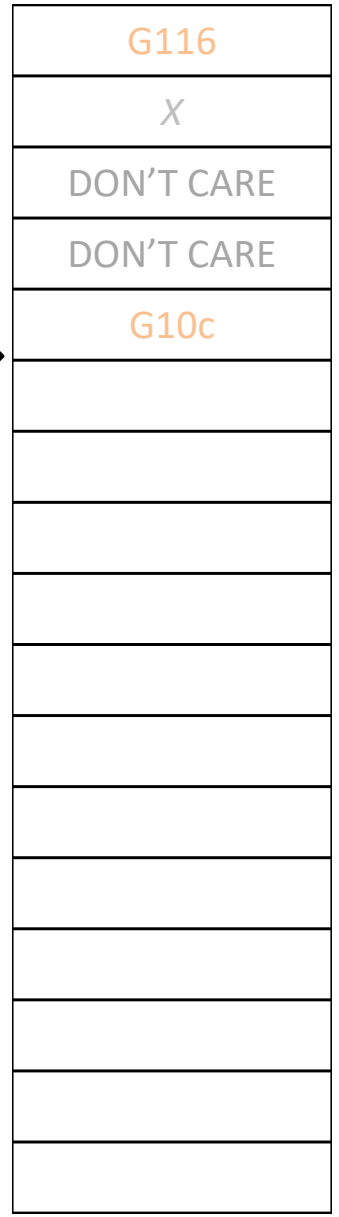
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx

```
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
```



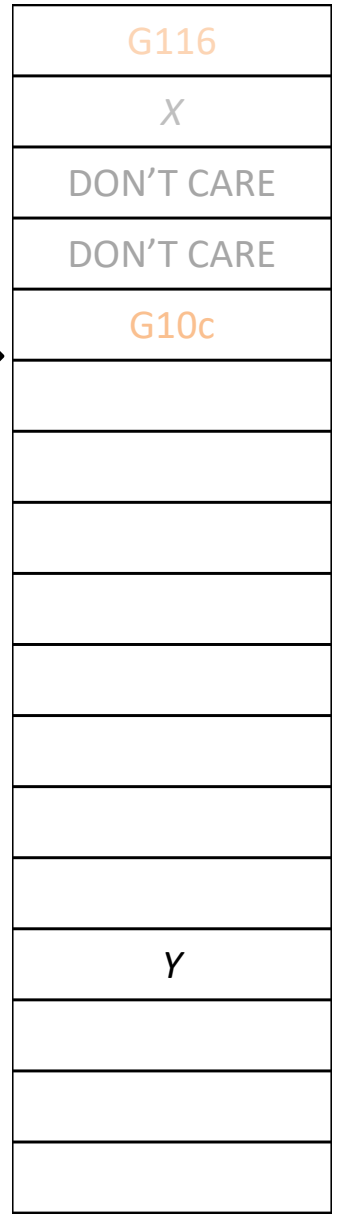
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx
  - Y in 0x20(%esp)

```
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
```



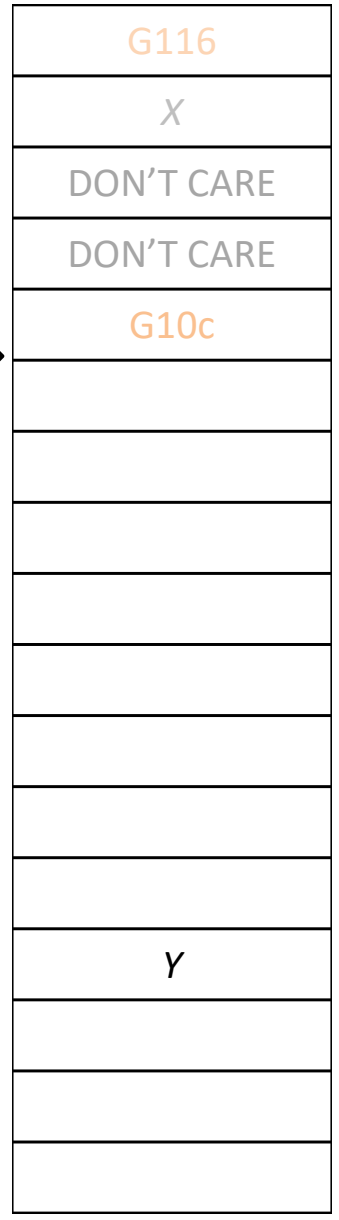
Other gadgets from /bin/ls

- Write *X* to location *Y*?

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```



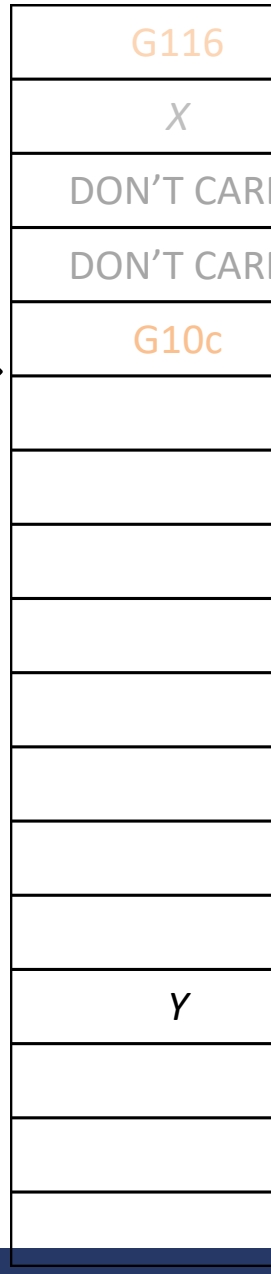
Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```



Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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 →

LOW

G116
X
DON'T CARE
DON'T CARE
G10c
DON'T CARE
DON'T CARE
DON'T CARE
Y

HIGH

Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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)   <- %esp
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
```

Other gadgets from /bin/ls

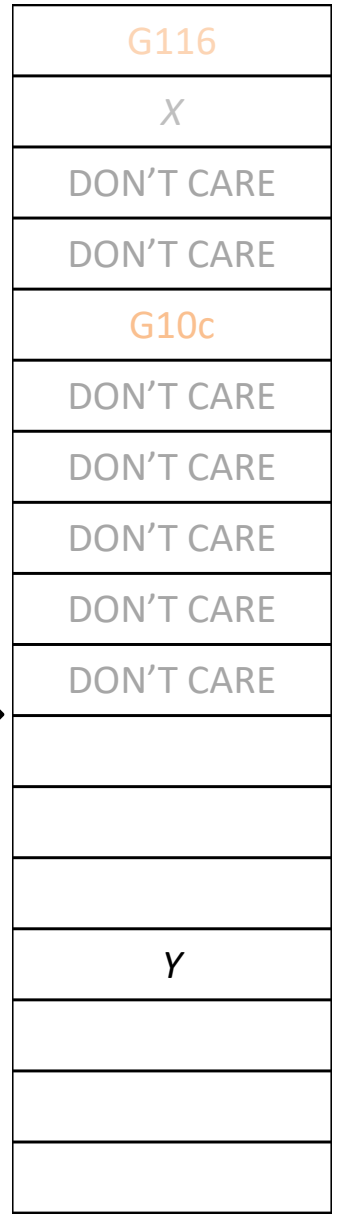
- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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 →



Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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 →

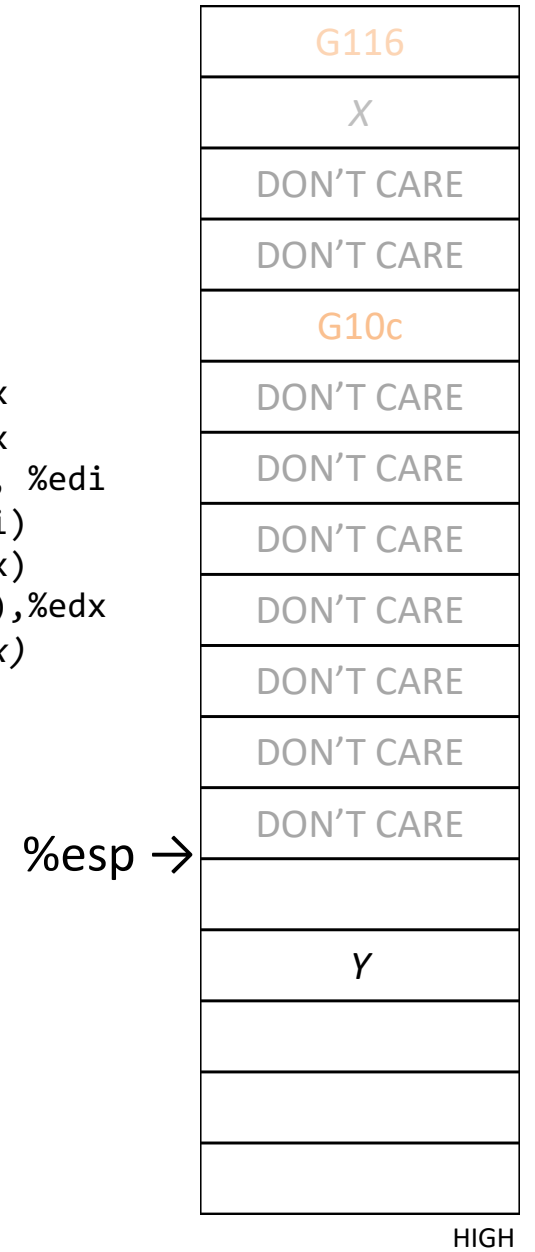
Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```



Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:
- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```

LOW

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
¿G119?
Y

%esp →

HIGH

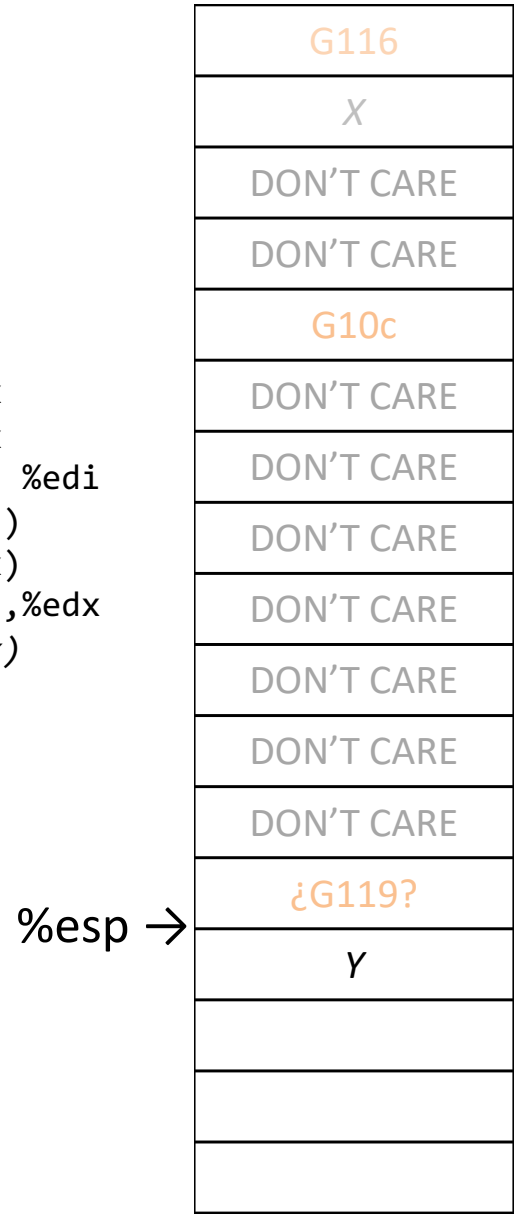
Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```



Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:
- *X* in %esi ✓
- *Y* in %edx ✓

```
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
```

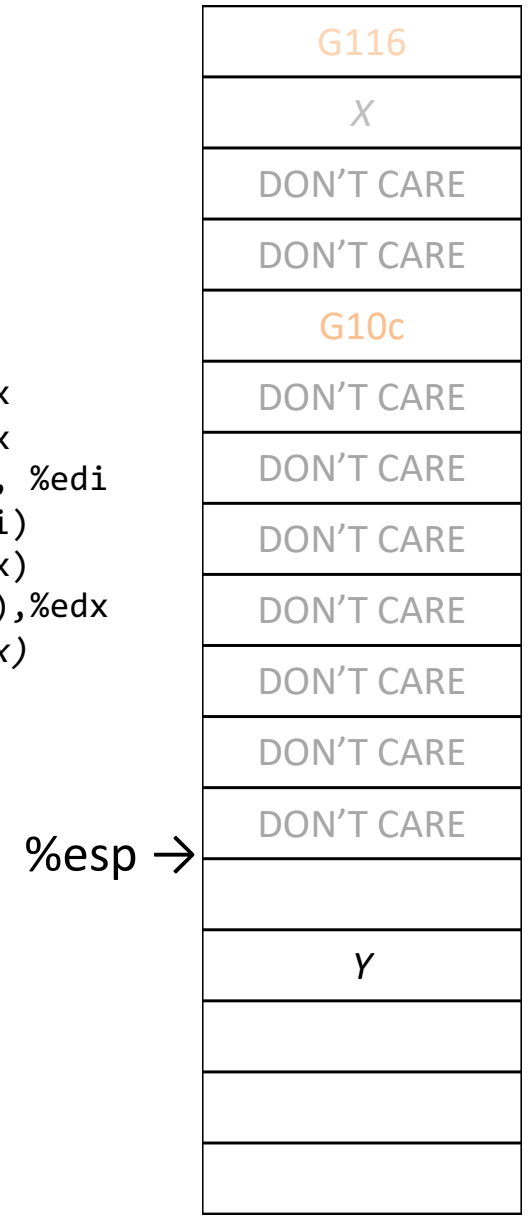
Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓
- Consume *Y*, then ret

```
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
```



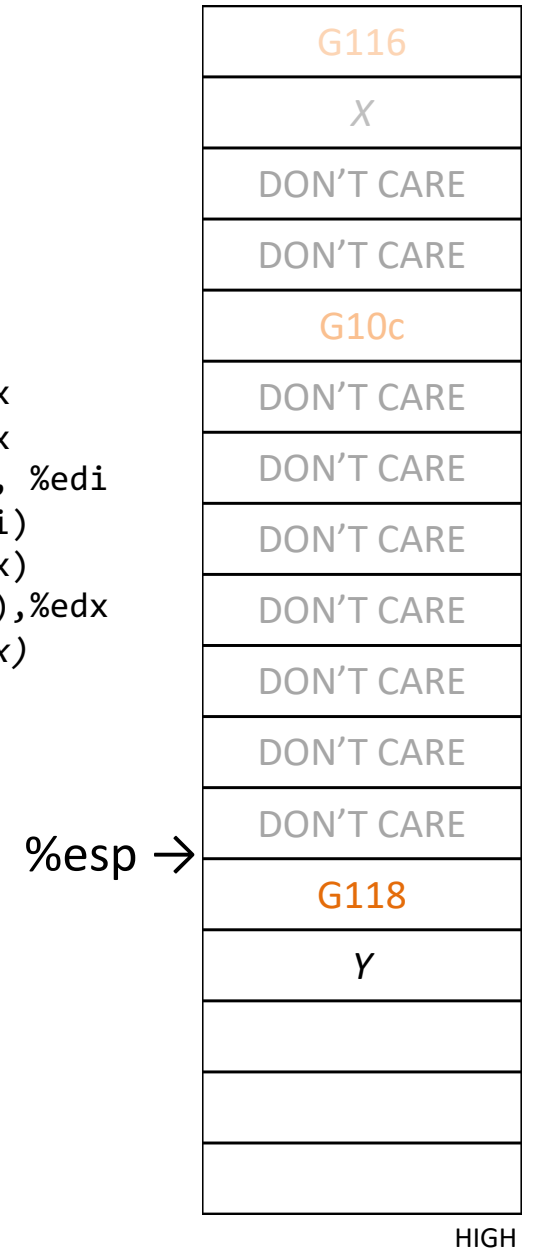
Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓
- Consume *Y*, then ret

```
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
```



Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:

- *X* in %esi ✓
- *Y* in %edx ✓
- Consume *Y*, then ret

```
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
```

LOW

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
G118
Y

%esp →

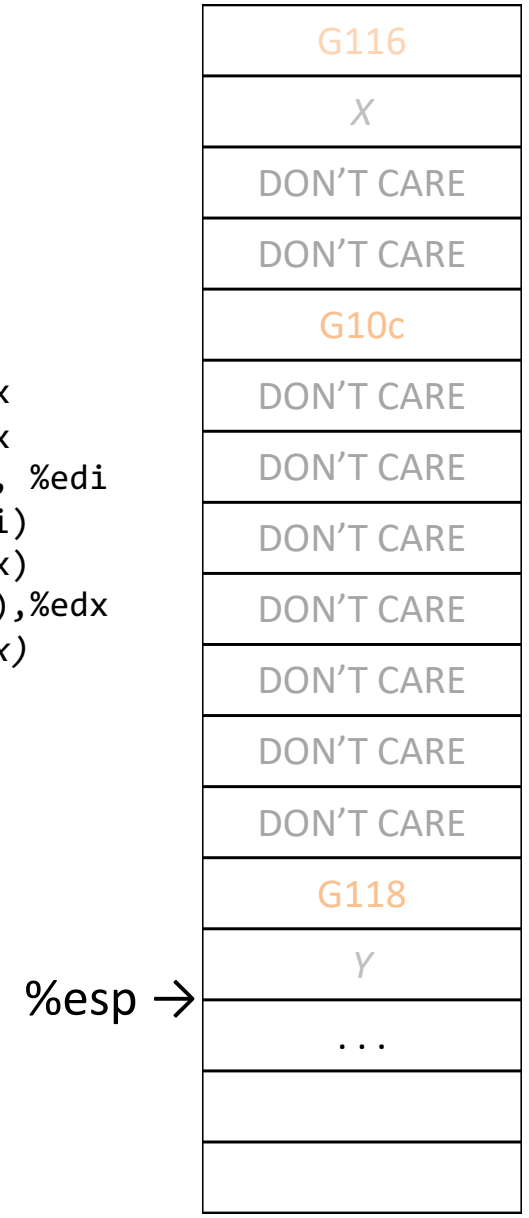
HIGH

Other gadgets from /bin/ls

- Write *X* to location *Y*? ✓

Want:
- *X* in %esi ✓
- *Y* in %edx ✓
- Consume *Y*, then ret ✓

```
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
```



Other gadgets from /bin/ls

- Write four byte value *X* to location *Y*?

```
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
```

Return Oriented Programming

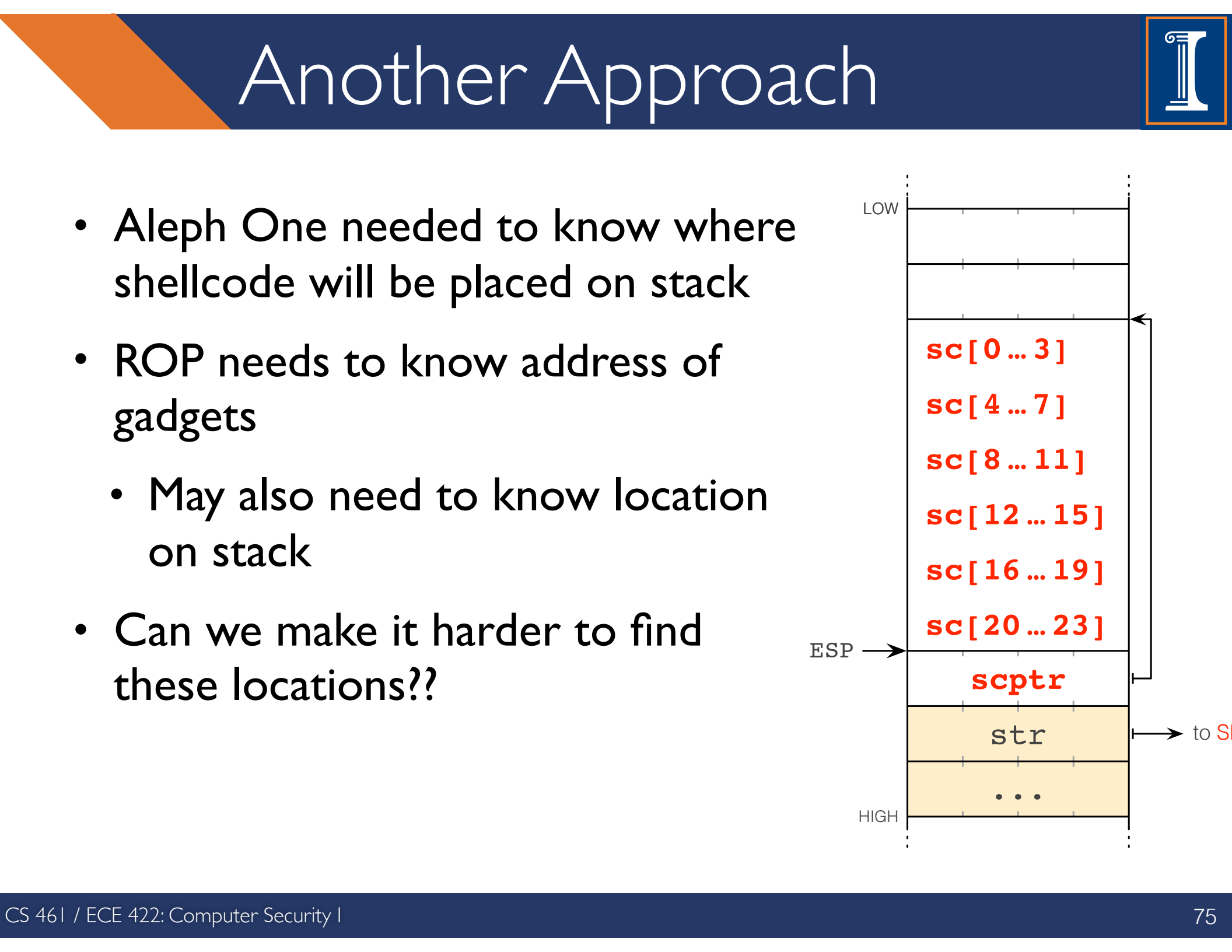
- Gadgets serve the role of instructions
- ROP programs assembled from gadgets
- There are ROP compilers to automate this

	Normal programming	Return-oriented programming
Instruction pointer	`%eip`	`%esp`
No-op	`nop`	`ret`
Jump	`jmp` *`addr`*	`movl` *`addr`*`, %esp`
Variable storage	Registers and memory	Memory

Return Oriented Programming

- Effectively defeated DEP as an effective defense against control flow hijacking
- Also used for attacking Harvard architecture systems (separate program and data memory)
- Vibrant area of research
  - –Persistent ROP, ROP without returns, etc.

Another Approach



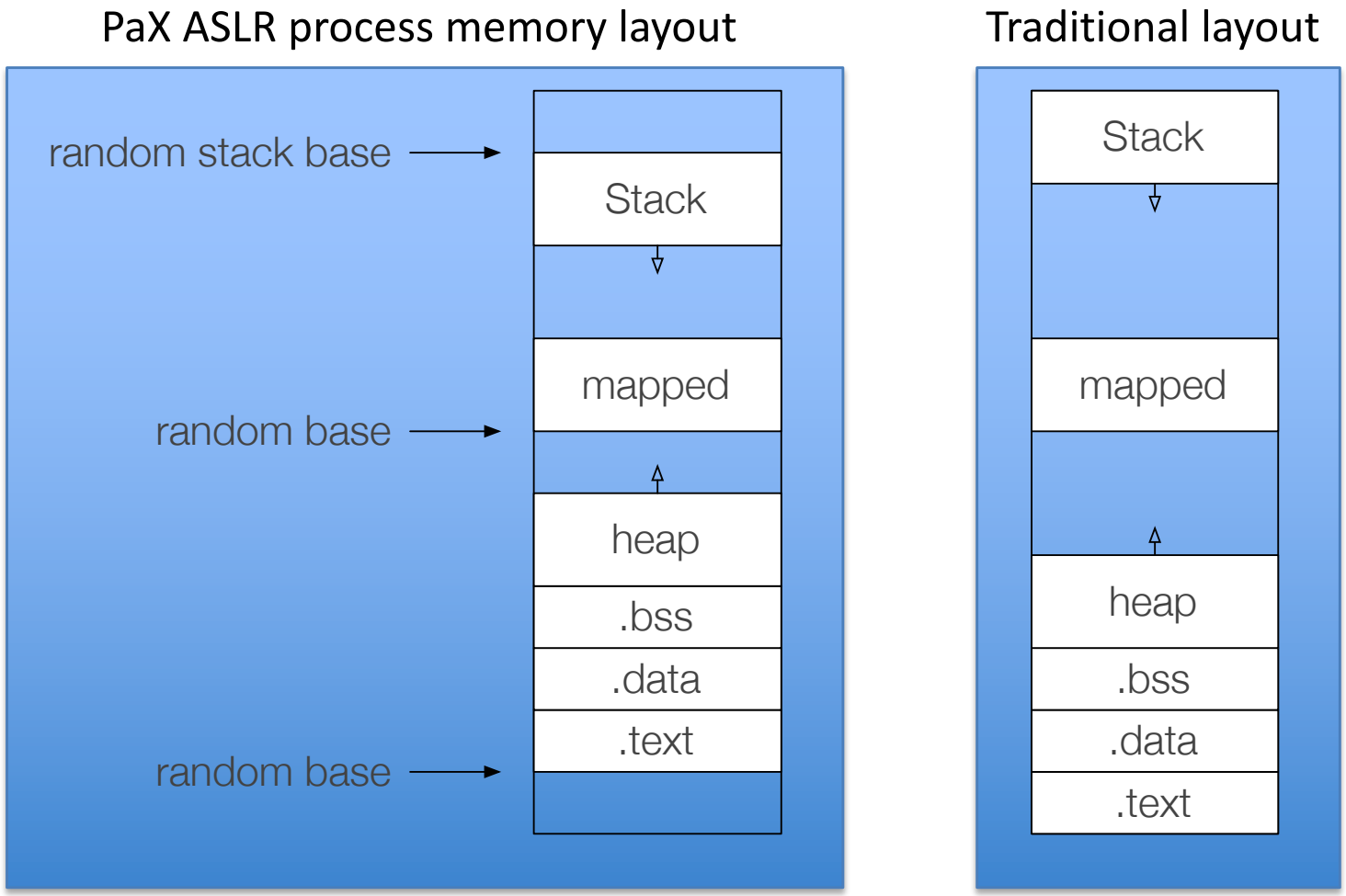
- Aleph One needed to know where shellcode will be placed on stack
- ROP needs to know address of gadgets
  - May also need to know location on stack
- Can we make it harder to find these locations??

Address Space Layout Randomization (ASLR)

- Randomize location of stack, heap, and code
- Locations must be not be known to attacker
  - Randomize on every launch (best)
  - Randomize at compile time
- Implemented (in some form) on most Oses
- Binaries must be compiled to support ASLR
  - Code must be position independent
  - GCC and Clang: -fPIE

32-bit PAX ASLR

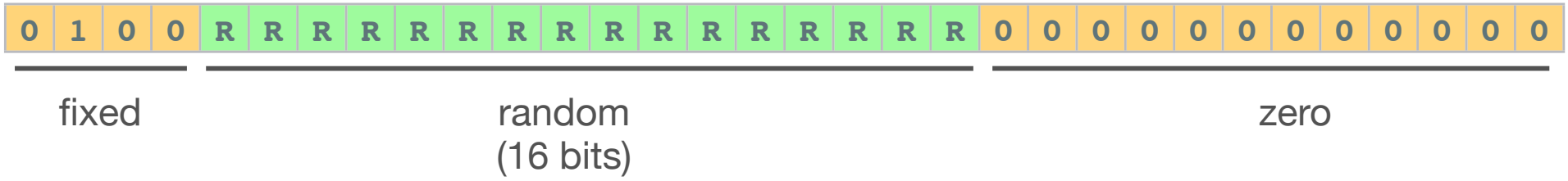
- Implementation of ASLR for Linux userspace



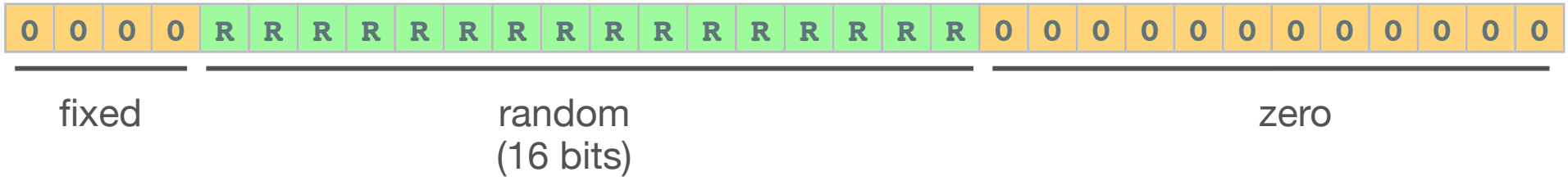
32-bit PAX ASLR (x86) Base Addresses

Stack:

Mapped area:



Executable code, static variables, and heap:

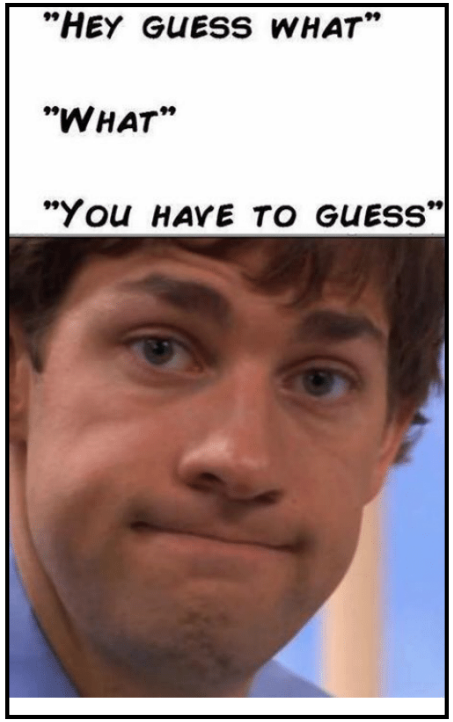


Control Flow Hijacking Attacks

- **Altering control flow** of a target program to cause it **to do what attacker wants**
- Strategy:
  1. **Identify a code pointer that is eventually loaded into PC**
  2. **Overwrite code pointer (memory write vulnerability)**
  3. **Divert PC to code that will do useful work (for attacker)**
- ASLR blocks step 3 because attacker does not know where the shellcode (or ROP gadgets) are located
- ALSR may also block step 2 if memory write vulnerability requires knowing a specific destination address to write to

Attacking ASLR

- Guess!
  - May be feasible on 32-bit systems: e.g. 216 = 65,536 possible PaX code offsets



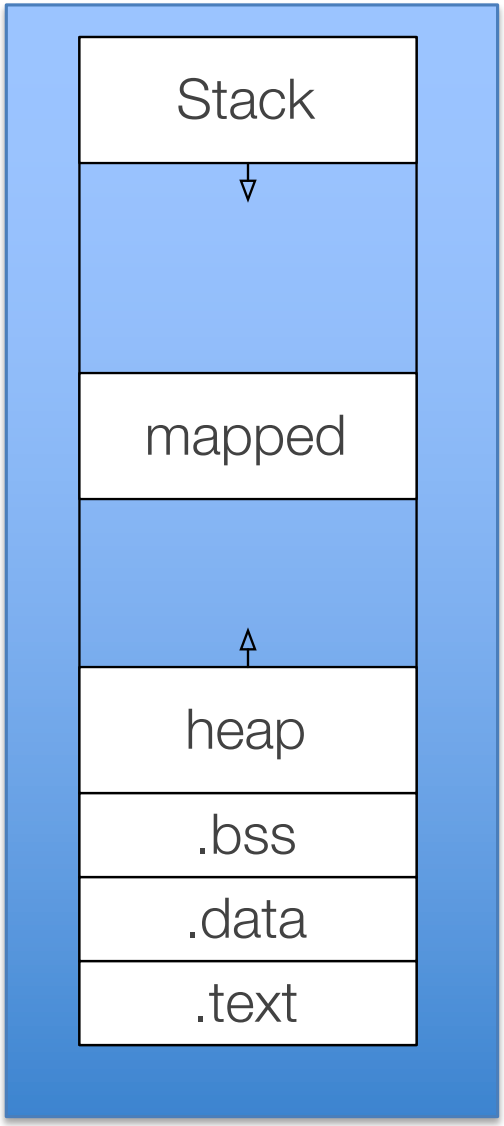
Attacking ASLR

- Guess!
  - May be feasible on 32-bit systems: e.g. 216 = 65,536 possible PaX code offsets
- Use a memory disclosure vulnerability to see parts of target memory
- Stack is a rich source of pointers to stack, heap, and text (code)

```
struct packet {
    unsigned int len;
    char data[256];
};

void buggy(const void * ptr, size_t len) {
    struct packet pkt;

    if (len < 256) {
        pkt.len = len;
        memcpy(pkt.data, ptr, len);
        send(&pkt, sizeof(pkt));
    }
}
```

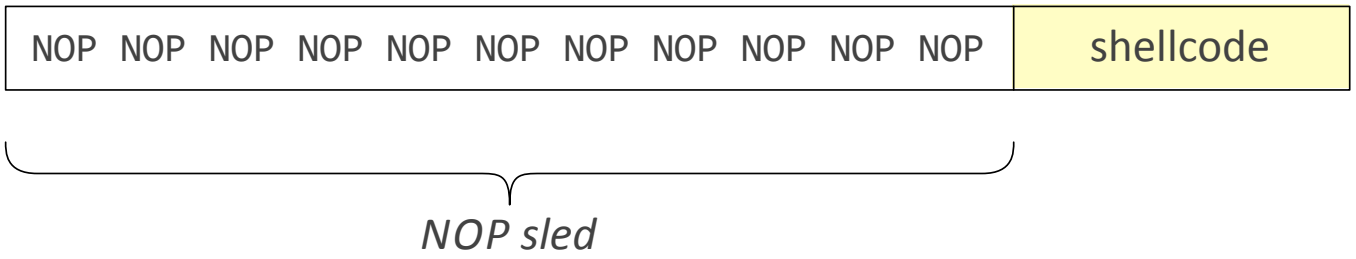
Heap Uncertainty

- Location of memory returned by malloc hard for attacker to predict: depends on sequence of previous calls to malloc and free
- Allocator uncertainty and ASLR make it hard to know exact location of heap shellcode

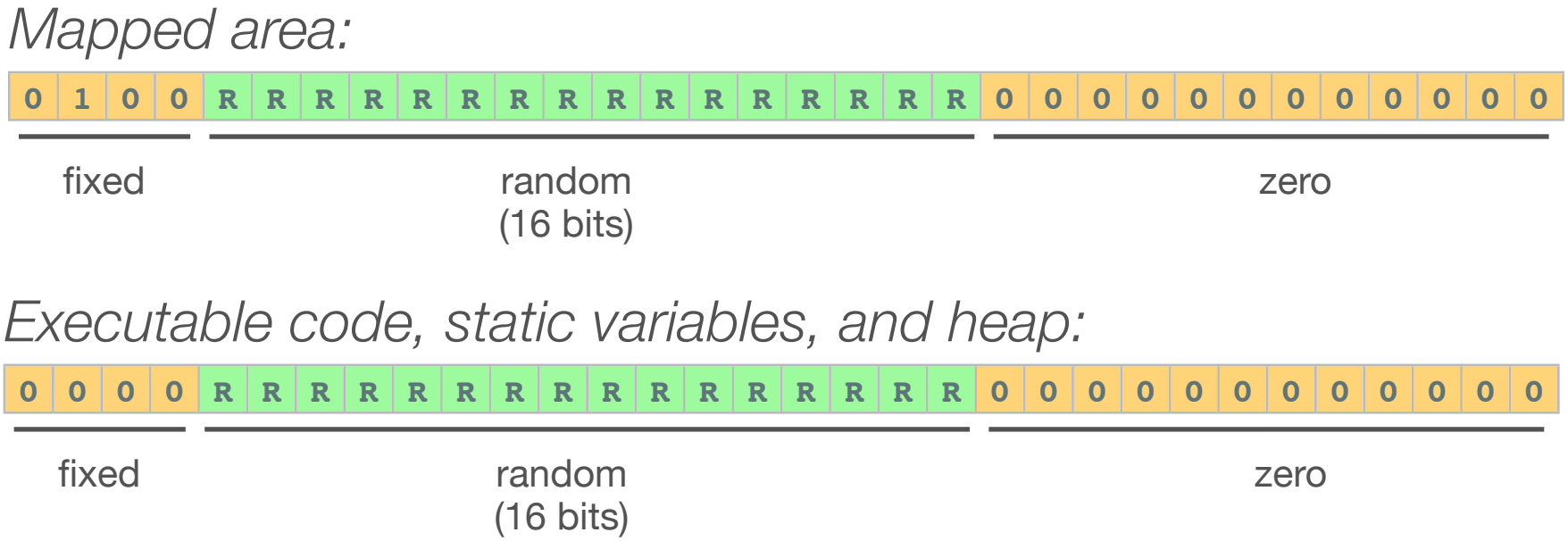


Heap Spray

- Allocate many instances of shellcode on the heap
- Prefix shellcode with NOP sled so that jump into NOP region will reach start of shellcode
- NOP sled and spray (many objects) compensate for address uncertainty by creating larger "target" for jump

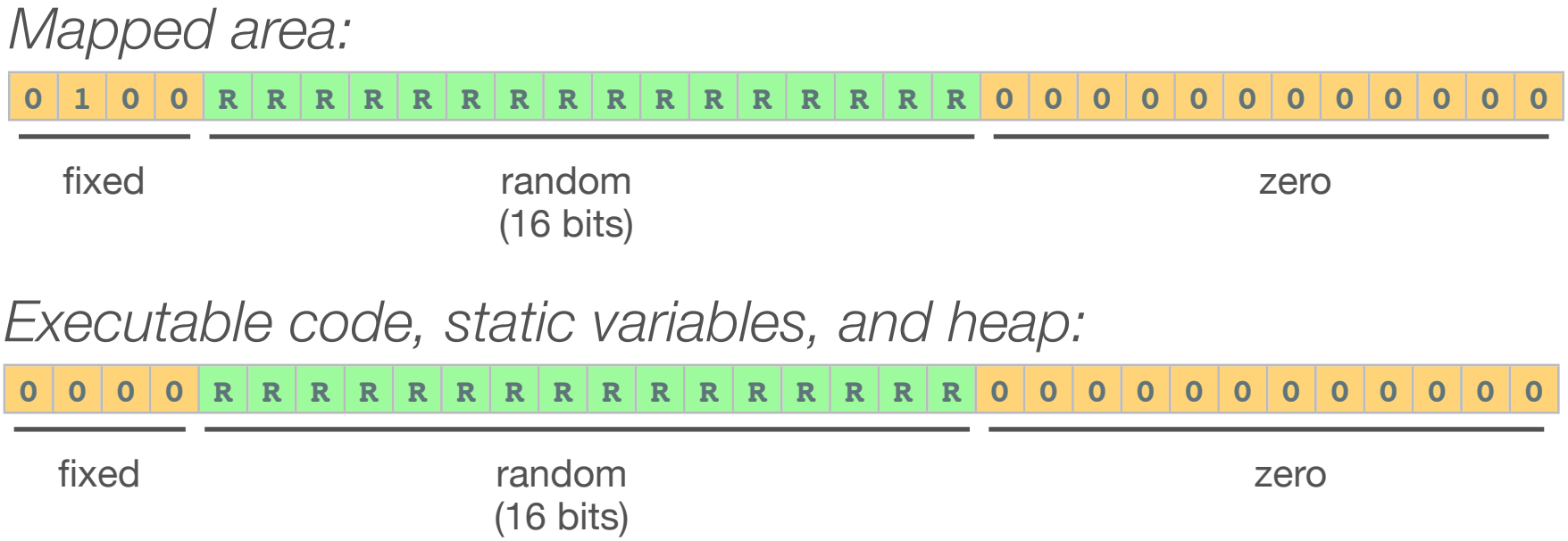


Heap Spray



- Max heap size is 1GB in 32-bit mode (PaX ASLR)
  - Lowest heap addr: `0x00000000`
  - Highest heap addr: `0x3fffffff`

Heap Spray



- Max heap size is 1GB in 32-bit mode (PaX ASLR)
- Allocate 750 objects of 1MB in size
  - Each object is NOP sled + shellcode
- A jump to 0x20000000 will land in NOP sled with high probability

Heap Spray

- Heap spray requires attacker to be able to allocate objects on the heap — how?
  - Provide as input to target program
  - User input loaded into a heap-allocated buffer
- Browsers are popular targets:
  - Victim users visits malicious Web site
  - Malicious site serves JavaScript to browser
  - Heap objects allocated from within JavaScript
  - Use control flow vulnerability to jump to heap

Control Flow Hijacking Attacks

- **Altering control flow** of a target program to cause it **to do what attacker wants**
- Strategy:
  1. **Identify a code pointer that is eventually loaded into PC**
  2. **Overwrite code pointer (memory write vulnerability)**
  3. **Divert PC to code that will do useful work (for attacker)**
- ASLR blocks step 3 because attacker does not know where the shellcode (or ROP gadgets) are located **— everywhere in heap!**
- ALSR may also block step 2 if memory write vulnerability requires knowing a specific destination address to write to **— spray!**

Heap Spray

- Ok, but what about DEP?
  - –Prevents us from executing code on the heap
- ROPspray? — not a thing :(
  - –Overwritten code ptr must point to existing code
- Memory disclosure vulnerabilities to get text segment or libc location

Just-in-Time Compilation

- Just-in-time (JIT) code compilation is used to improve JavaScript performance in browsers
  - JIT compiler output is executable native code
- Output is in executable memory!
  - JavaScript code is untrusted, JIT generated native code is not arbitrary native code
- Can't write shellcode in JavaScript because JavaScript limited in what it is allowed to do

JIT Spray

- Compiler-generated code may have unintended sequences of native code
  - E.g. from JavaScript string and integer literals
- Carefully crafted JavaScript can produce native NOP sled + shellcode after JIT!
- Create many instances of JITed code
- Use control flow hijack to jump to this code
- Used to exploit Web browsers

The Eternal War in Memory…

- **Altering control flow** of a target program to cause it **to do what attacker wants**

1. **Identify a code pointer that is eventually loaded into PC**
   - **Return address on stack, virtual function pointers**
2. **Overwrite code pointer (memory write vulnerability)**
   - **classic buffer overflows**
   - **stack canaries, finding + fixing bugs in code**
   - **Non-stack buffer overflows and other memory write vuln**
3. **Divert PC to code that will do useful work (for attacker)**
   - **Data Execution Prevention (DEP)**
   - **Return-to-libc and return-oriented programming (ROP)**
   - **Address Space Layout Randomization (ASLR)**
   - **Memory disclosure vulnerabilities**
   - **Heap Spray and JIT Spray**

The Eternal War in Memory…

- Anyhow, time to move on.