Lecture Outline

- Notion of binding time
- Object lifetime and storage management

- An aside: Stack Smashing 101
  - Slides courtesy of RPISEC/MBE
- Scoping
  - Static scoping
  - Dynamic scoping

Stack Frames

- In x86-64 RBP is fp and RSP is sp. Define the stack frame for the currently executing function
  - local variables
  - pointer to previous frame
  - return address

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jnz     short loc_30FB
```

note: for 64bit, each 'slot' is 8 bytes

```
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:
```

```
void foo() {
  long long x = 0x1337;
  char str[16];
  strcpy(str, "ABCDEFGH0123456");
}
```


```
                jnz     short loc_3130
                lea     rsi, aWannaCheatYes1 ; "wanna ch
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                lea     rax, [rbp+var_14]
```

What is corruption?

- So what happens if a programer makes a simple mistake:

```
char foo[64];
int money = 0;
gets(foo);
```

gets()?

```
NAME
       gets - get a string from standard input (DEPRECATED)

SYNOPSIS
       #include <stdio.h>

       char *gets(char *s);

DESCRIPTION
       Never use this function.

       gets() reads a line from stdin into the buffer pointed to by s until either a terminating new-
       line or EOF, which it replaces with a null byte ('\0').  No check for buffer overrun  is  per-
       formed (see BUGS below).
```

- DO NOT **EVER** USE
  - scanf("%s", ...) as well

- So what happens if we give this program a bunch of A's?
  With gets we can give as many A's as we want!

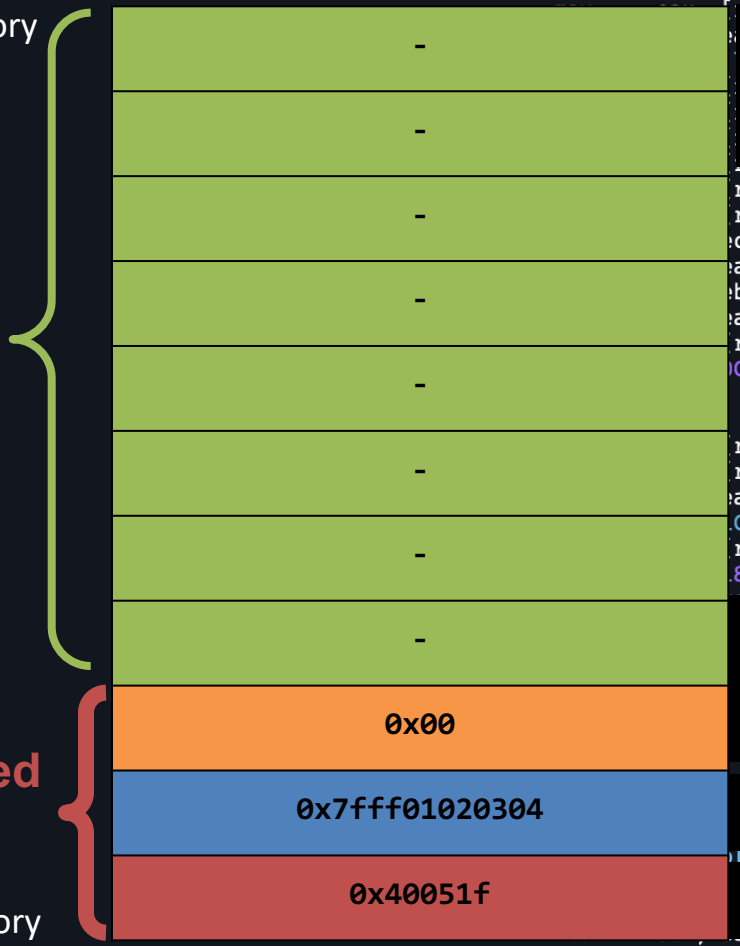
```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcESaIcE
                lea     rsi, aIsDead    ; " is dead!"
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcEERSt13
                mov     rdx, cs:_ZSt4endlIcSt11char_trai
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostream
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna ch
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt13
                lea     rax, [rbp+var_14]
```

Stack Smashing 101

main() has a

stack frame

- Contains local variables
- Pointer to previous frame
- Return address



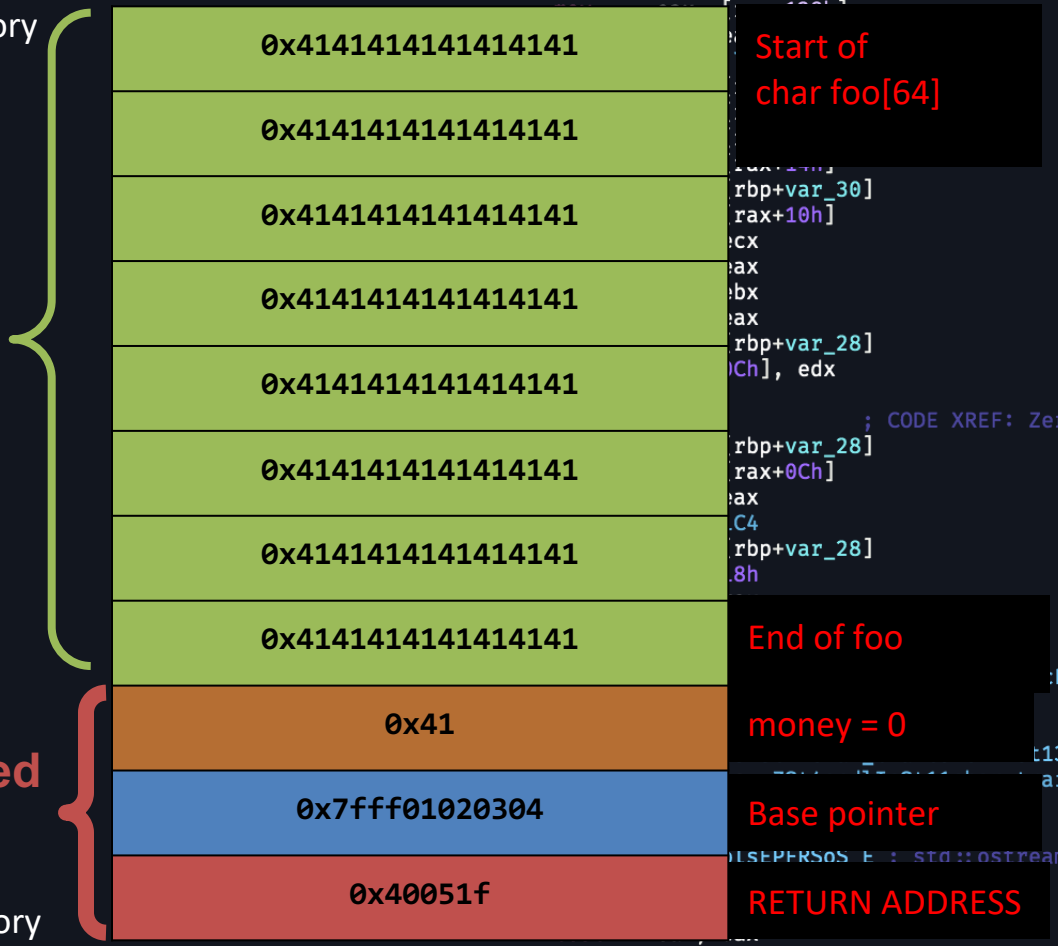
Stack Smashing 101

As gets() continues to read input, we fill up the 64 bytes allocated for buffer foo

Stack Smashing 101

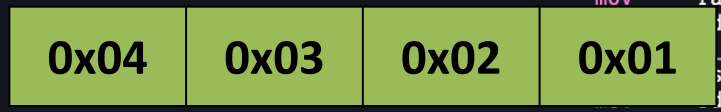
As gets() continues to read input, we fill up the 64 bytes allocated for foo

Go far enough, it corrupts important data!



Stack Smashing 101

- We can give ourselves money
- If we want to set money to 0x1337beef we need to know:
  - Most x86 machines are little endian (little byte goes first)
  - Meaning the byte order for numbers is "backwards" in memory
  - 0x01020304 would be




```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {

                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jnz     short loc_30FB
                mov     rax, [rbp+var_28]
                mov     edx, [rax+0Ch]
                mov     rax, [rbp+var_28]
                mov     ecx, [rax+14h]
                mov     rax, [rbp+var_30]
                mov     eax, [rax+10h]
                mov     ebx, ecx
                sub     ebx, eax
                mov     eax, ebx
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:                               ; CODE XREF: Ze
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcESaIcE
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                mov     esi, eax
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt11
                mov     rdx, cs:_ZSt4endlIcSt11char_trai
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostrean
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna ch
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt11
                lea     rax, [rbp+var_14]
```

Stack Smashing 201

- What else can we corrupt?
- What happens if you corrupt further? When does it segfault?
  - What was that about a return address?

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jnz     short loc_30FB
                mov     rax, [rbp+var_28]
                mov     edx, [rax+0Ch]
                mov     rax, [rbp+var_28]
                mov     ecx, [rax+14h]
                mov     rax, [rbp+var_30]
                mov     eax, [rax+10h]
                mov     ebx, ecx
                imul    edx, eax
                mov     eax, ebx
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:                               ; CODE XREF: Ze
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcESaIc
                lea     rsi, aIsDead    ; " is dead!"
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                mov     rdx, cs:_ZSt4endlIcSt11char_tra
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostrea
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna c
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                lea     rax, [rbp+var_14]
```

Stack Smashing 201

```
int func() {
      puts("Hello World");
      return 17;
}
int main() {
      int res = func();
      return 0;
}
```

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jnz     short loc_30FB
                mov     rax, [rbp+var_28]
                mov     edx, [rax+0Ch]
                mov     rax, [rbp+var_28]
                mov     ecx, [rax+14h]
                mov     rax, [rbp+var_30]
                mov     eax, [rax+10h]
                mov     ebx, ecx
                sub     ebx, eax
                mov     eax, ebx
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:                               ; CODE XREF: Ze
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    _ZStlsISt11char_traitsIcESaIcE
                lea     rsi, aIsDead    ; " is dead!"
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                mov     rdx, cs:_ZSt4endlIcSt11char_trai
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostrea
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna c
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                lea     rax, [rbp+var_14]
```

When **func()** is called, runtime stores the **return address** on the stack (i.e., the address of the instruction that immediately follows **call func** in **main**)

Stack Smashing 201

Before the call:

```
=> 0x40051a <main+13>:  call   0x4004f6 <func>
   0x40051f <main+18>:  mov    DWORD PTR [rbp-0x4],eax
   0x400522 <main+21>:  mov    eax,0x0
   0x400527 <main+26>:  leave
   0x400528 <main+27>:  ret
No argument
[-----------------------------------stack-------------
0000| 0x7fffffffe0f0 --> 0x7fffffffe1e0 --> 0x1
0008| 0x7fffffffe0f8 --> 0x0
0016| 0x7fffffffe100 --> 0x400530 (<__libc_csu_init>:
```

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jnz     short loc_30FB
                mov     rax, [rbp+var_28]
                mov     edx, [rax+0Ch]
                mov     rax, [rbp+var_28]
                mov     ecx, [rax+14h]
                mov     rax, [rbp+var_30]
                mov     eax, [rax+10h]
                mov     ebx, ecx
                sub     ebx, eax
                mov     eax, ebx
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:                               ; CODE XREF: Ze
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcESaIc
                lea     rsi, aIsDead    ; " is dead!"
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                mov     rdx, cs:_ZSt4endlIcSt11char_tra
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostrea
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna c
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                lea     rax, [rbp+var_14]
```

Stack Smashing 201

Before the call:

```
=> 0x40051a <main+13>:  call   0x4004f6 <func>
   0x40051f <main+18>:  mov    DWORD PTR [rbp-0x4],eax
   0x400522 <main+21>:  mov    eax,0x0
   0x400527 <main+26>:  leave
   0x400528 <main+27>:  ret
No argument
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe0f0 --> 0x7fffffffe1e0 --> 0x1
0008| 0x7fffffffe0f8 --> 0x0
0016| 0x7fffffffe100 --> 0x400530 (<__libc_csu_init>:
```

After the call:

```
=> 0x4004f6 <func>:     push   rbp
   0x4004f7 <func+1>:   mov    rbp,rsp
   0x4004fa <func+4>:   lea    rdi,[rip+0xb3]
   0x400501 <func+11>:  call   0x4003f0 <puts@plt>
   0x400506 <func+16>:  mov    eax,0x11
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe0e8 --> 0x40051f (<main+18>:   mov
0008| 0x7fffffffe0f0 --> 0x7fffffffe1e0 --> 0x1
0016| 0x7fffffffe0f8 --> 0x0
```

Return address points back to where it left off in main

Stack Smashing 201

Returning just takes whatever is on the top of the stack, and jumps there, equivalently: `pop rip`

About to return:

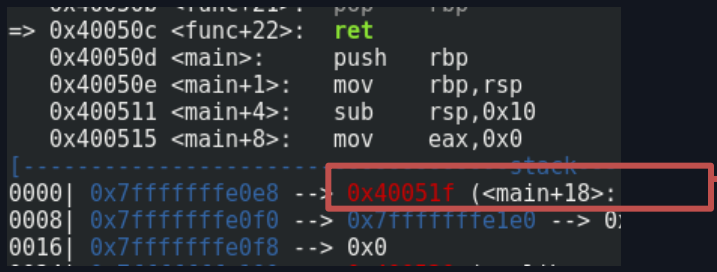
```
=> 0x40050c <func+22>:  ret
   0x40050d <main>:     push   rbp
   0x40050e <main+1>:   mov    rbp,rsp
   0x400511 <main+4>:   sub    rsp,0x10
   0x400515 <main+8>:   mov    eax,0x0
[-------------------------------------stack-------]
0000| 0x7fffffffe0e8 --> 0x40051f (<main+18>:
0008| 0x7fffffffe0f0 --> 0x7fffffffe1e0 --> 0
0016| 0x7fffffffe0f8 --> 0x0
```

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
                mov     eax, [rax+128h]
                test    eax, eax
                jz      short loc_30FB
                mov     rax, [rbp+var_28]
                mov     edx, [rax+0Ch]
                mov     rax, [rbp+var_28]
                mov     ecx, [rax+14h]
                mov     rax, [rbp+var_30]
                mov     eax, [rax+10h]
                mov     ebx, ecx
                sub     ebx, eax
                mov     eax, ebx
                add     edx, eax
                mov     rax, [rbp+var_28]
                mov     [rax+0Ch], edx

loc_30FB:                               ; CODE XREF: Ze
                mov     rax, [rbp+var_28]
                mov     eax, [rax+0Ch]
                test    eax, eax
                jns     loc_31C4
                mov     rax, [rbp+var_28]
                add     rax, 18h
                mov     rsi, rax
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsIcSt11char_traitsIcESaIc
                lea     rsi, aIsDead    ; " is dead!"
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                mov     rdx, cs:_ZSt4endlIcSt11char_tra
                mov     rsi, rdx
                mov     rdi, rax
                call    __ZNSolsEPFRSoS_E ; std::ostrea
                mov     rax, [rbp+var_28]
                mov     eax, [rax+8]
                test    eax, eax
                jnz     short loc_31BD
                lea     rsi, aWannaCheatYes1 ; "wanna c
                mov     rax, cs:_ZSt4cout_ptr
                mov     rdi, rax
                call    __ZStlsISt11char_traitsIcEERSt1
                lea     rax, [rbp+var_14]
```

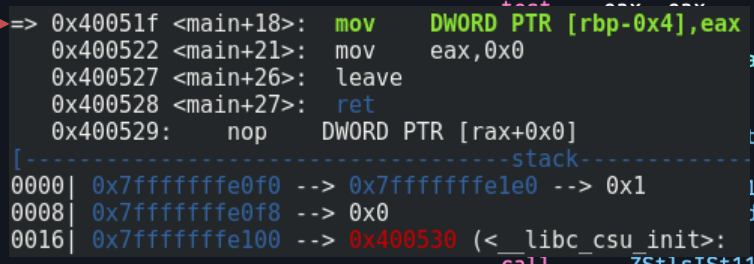
Stack Smashing 201

Returning just takes whatever is on the top of the stack, and jumps there, equivalently: `pop rip`

About to return:



Returned back to main:



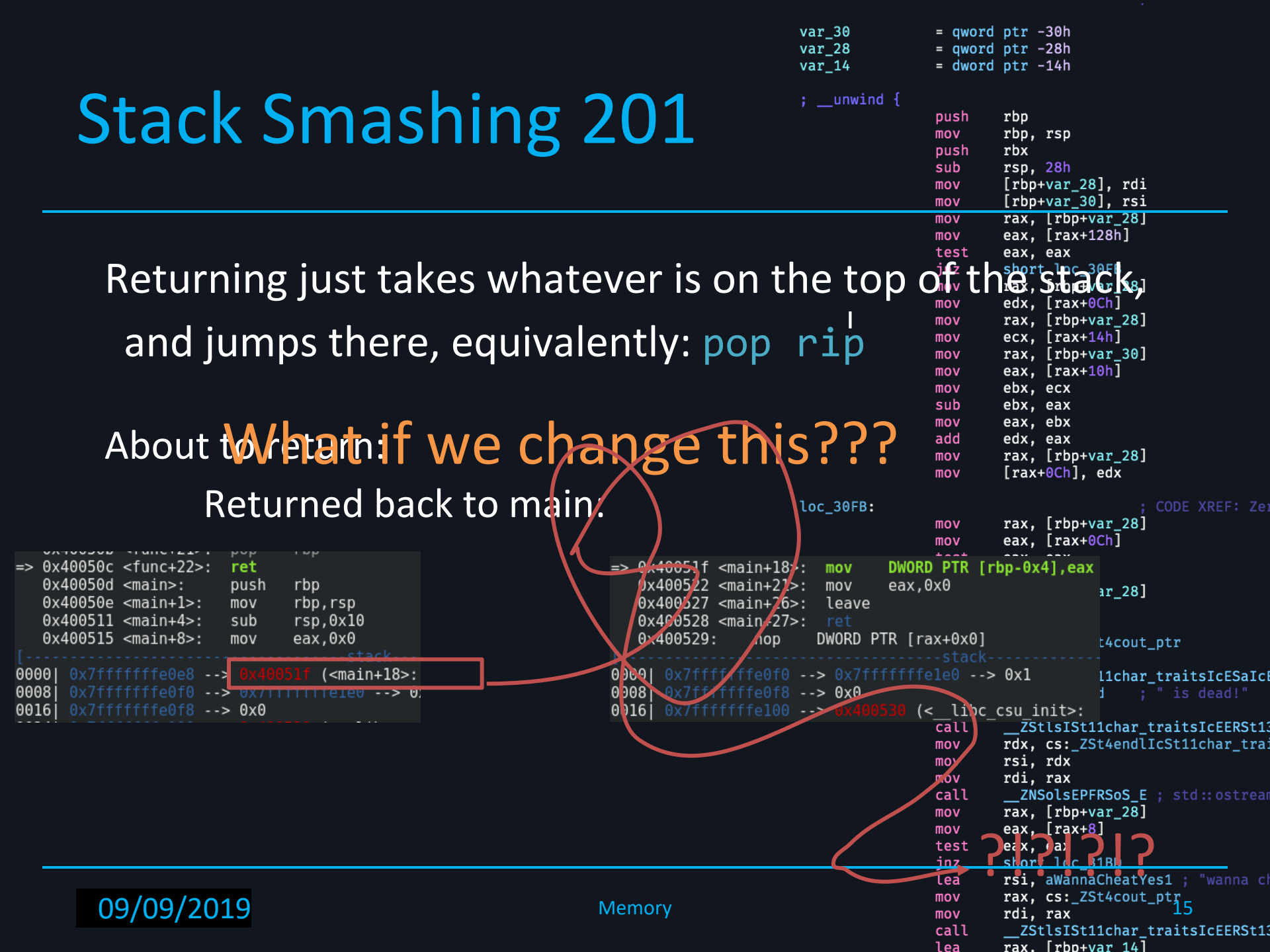
Stack Smashing 201

Returning just takes whatever is on the top of the stack, and jumps there, equivalently: `pop rip`

About to return:

Returned back to main:

What if we change this???



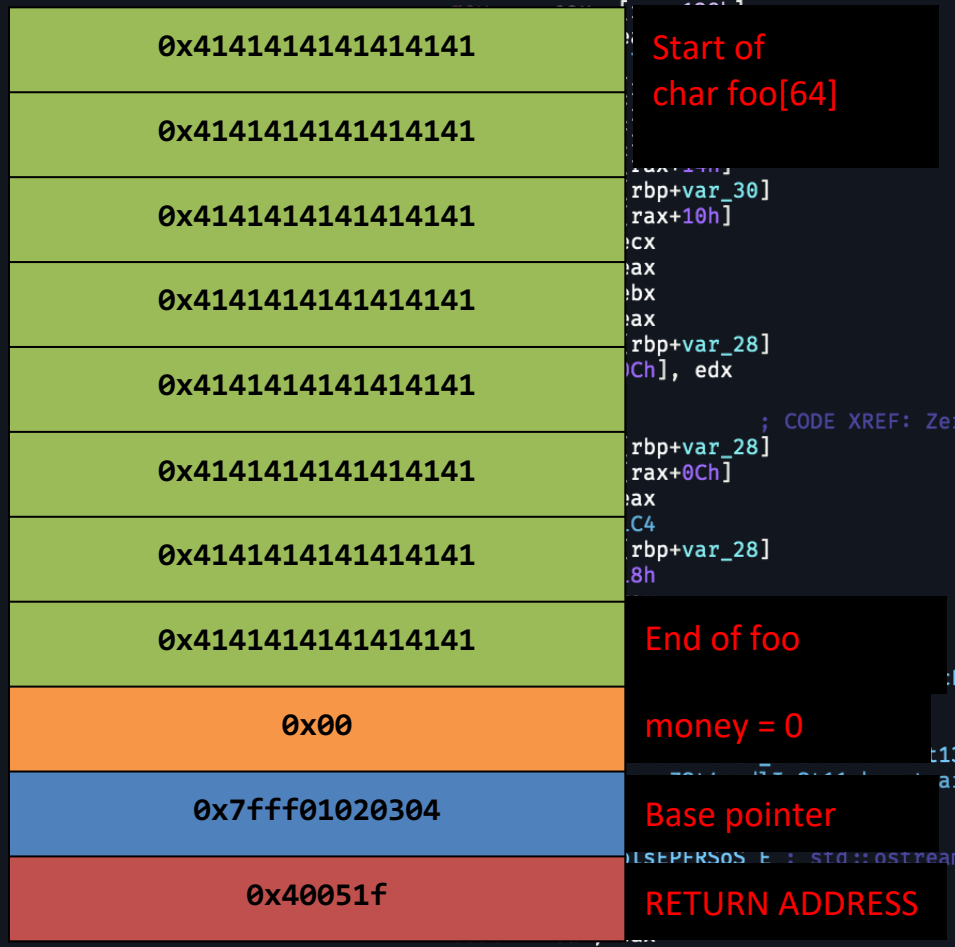
Stack Smashing 201

```
var_30          = qword ptr -30h
var_28          = qword ptr -28h
var_14          = dword ptr -14h

; __unwind {
                push    rbp
                mov     rbp, rsp
                push    rbx
                sub     rsp, 28h
                mov     [rbp+var_28], rdi
                mov     [rbp+var_30], rsi
                mov     rax, [rbp+var_28]
```

Without corruption:

- At the end of the function, it returns
- 0x40051f is popped off the stack and stored in rip
- Control goes to that address

We want to change this



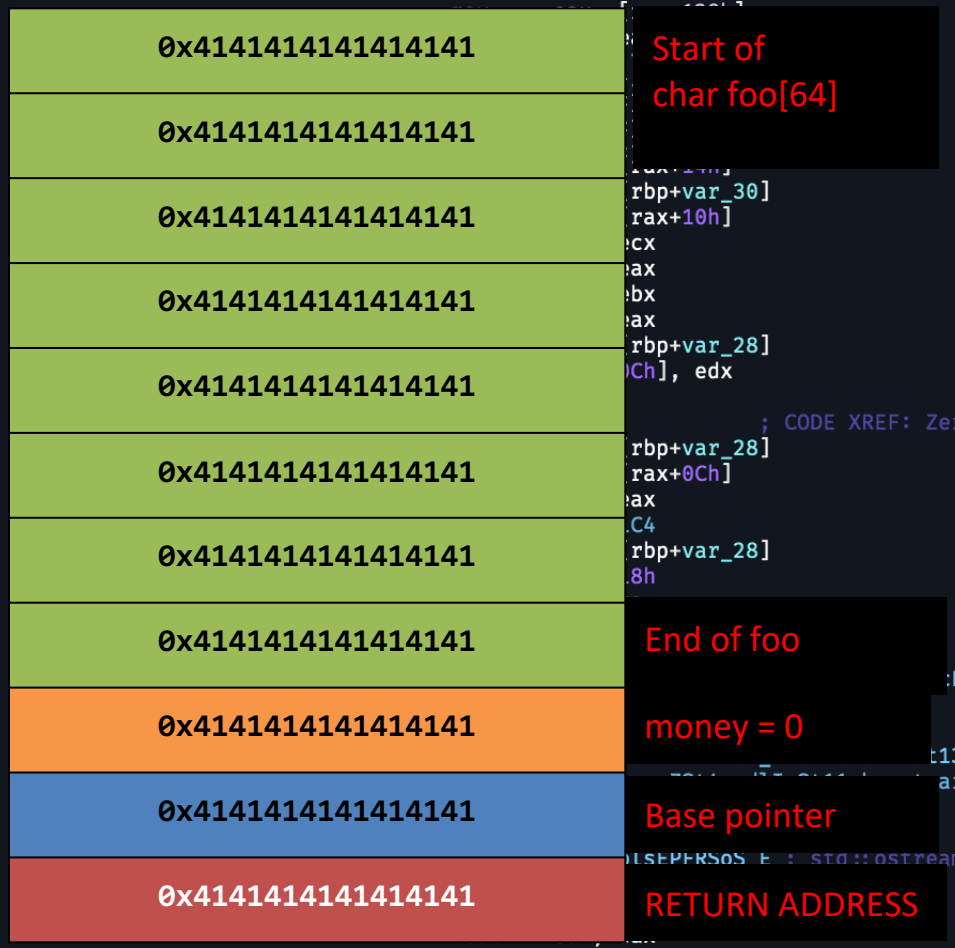
Stack Smashing 201

Lower Memory

Corrupted:

- At the end of the function, it returns
- 0x4141414141414141 is popped off the stack and stored in rip
- Control goes to that address
- but it's invalid memory...

Segmentation fault



Higher Memory