

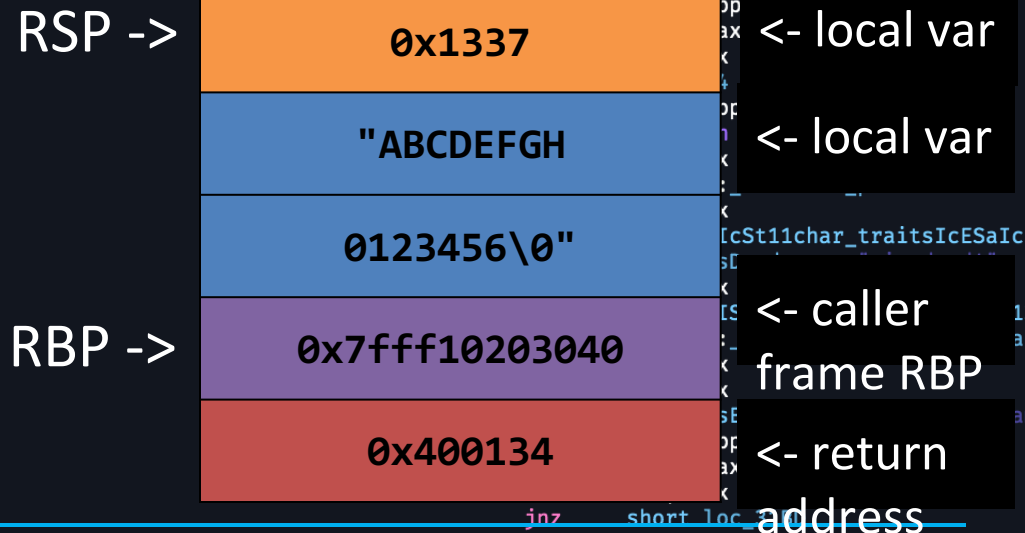
Lecture Outline

- 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

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



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

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

```
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
inc     short loc_30F8
```

loc_30FB:

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

The diagram illustrates a stack frame with five colored segments, each representing a different type of memory storage:

- Orange Segment (Top):** Contains the value `7`. It is labeled `<- local var` (local variable).
- Blue Segment:** Contains the text `FGH`. It is labeled `<- local var` (local variable).
- Light Blue Segment:** Contains the text `\0"`. It is labeled `<- caller` (caller's data).
- Purple Segment:** Contains the value `03040`. It is labeled `frame RBP` (frame pointer register).
- Red Segment (Bottom):** Contains the value `34`. It is labeled `<- return address` (return address).

At the very bottom, a green line is labeled `jmp short loc_30000000`, indicating a jump instruction.

```

    jnz     short loc_00401020 ; address_1
    lea     rsi, aWannaCheatYes1 ; "wanna cheat?"
    mov     rax, cs:_ZSt4cout_ptr ; address_2
    mov     rdi, rax
    call    _ZStlsISt11char_traitsIcEERSt11__va
    lea     rax, [rbp+var_14]

```

What is corruption?

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

```
char foo[64];
```

```
int money = 0;
```

```
gets(foo);
```

```
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: Zei
    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@13
    lea     rsi, aIsDead ; " is dead!"
    mov     rdi, rax
    call    __ZStlsISt11char_traitsIcEERSt13
    mov     rdx, cs:_ZSt4endlIcSt11char_tra
    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]
```

gets()?

```
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
```

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 performed (see BUGS below).

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

- So what happens if we give this program a bunch of A's?

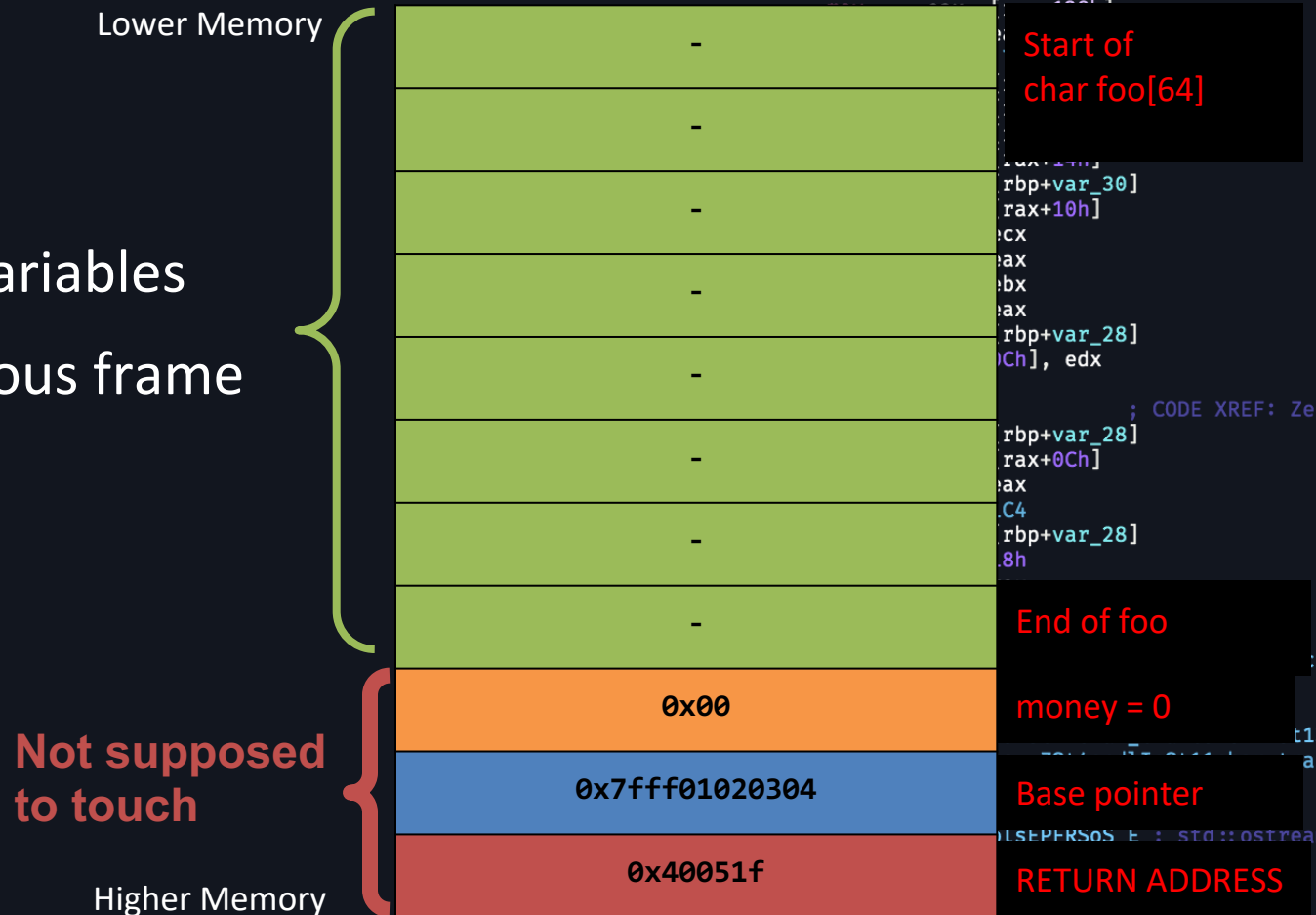
With **gets** we can give as many A's as we want!

```
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>_ZStlsIcSt11char_traitsIcESaIc>
lea      rsi, aIsDead ; " is dead!"
mov      rdi, rax
call     __ZStlsIcSt11char_traitsIcESaIc>_ZStlsIcSt11char_traitsIcESaIc>
mov      rdx, cs:_ZSt4endlIcSt11char_traitsIcESaIc>_ZSt4endlIcSt11char_traitsIcESaIc>
mov      rsi, rdx
mov      rdi, rax
call     __ZNSoIsEPFRSoS_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     __ZStlsIcSt11char_traitsIcESaIc>_ZStlsIcSt11char_traitsIcESaIc>
lea      rax, [rbp+var_14]
```

Stack Smashing 101

main() has a
stack frame

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



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

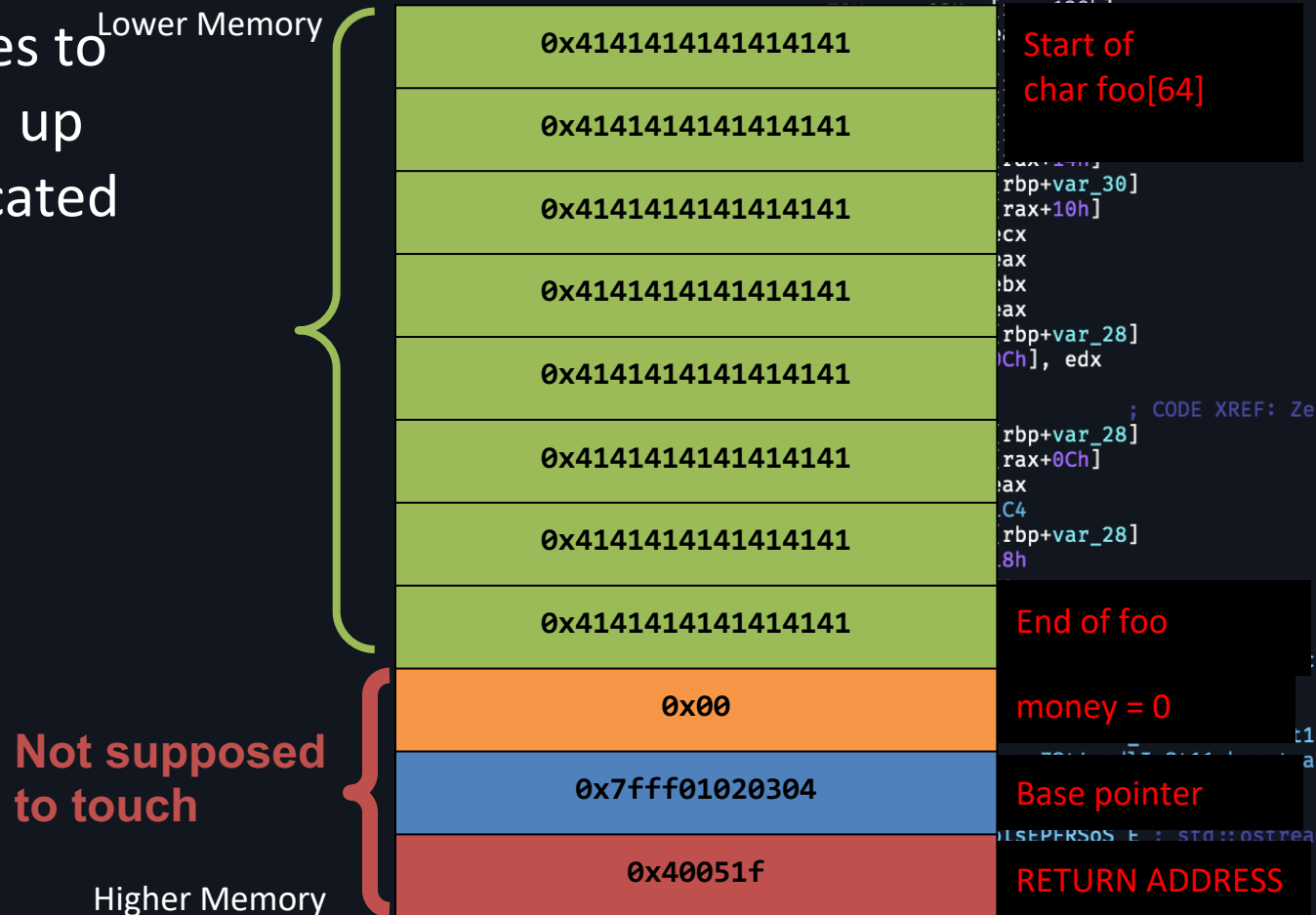
```
; __unwind {
    push    rbp
    mov     rbp, rsp
    push    rbx
    sub     rsp, 28h
    mov     [rbp+var_28], rdi
    mov     [rbp+var_30], rsi
    mov     rax, [rbp+var_28]
```

```
    mov     rbp+var_30]
    rax+10h]
    cx
    rax
    rbx
    rax
    rbp+var_28]
    Ch], edx
    ; CODE XREF: Zei
    rbp+var_28]
    rax+0Ch]
    rax
    C4
    rbp+var_28]
    8h
```

```
    jnz     short loc_318D
    lea     rsi, aWannaCheatYes1 ; "wanna ch
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    _ZStlsISt11char_traitsIcEERSt13
    lea     rax, [rbp+var_14]
```

Stack Smashing 101

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



```
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]
```

```
rbp+var_30]
rax+10h]
cx
ax
bx
ax
rbp+var_28]
Ch], edx
CODE XREF: Ze
rbp+var_28]
rax+0Ch]
ax
C4
rbp+var_28]
8h
```

```
inzb short loc_318D
lea rsi, aWannaCheatYes1 ; "wanna ch
mov rax, cs:_ZSt4cout_ptr
mov rdi, rax
call _ZStlsISt11char_traitsIceERSt13
lea rax, [rbp+var_14]
```

Stack Smashing 101

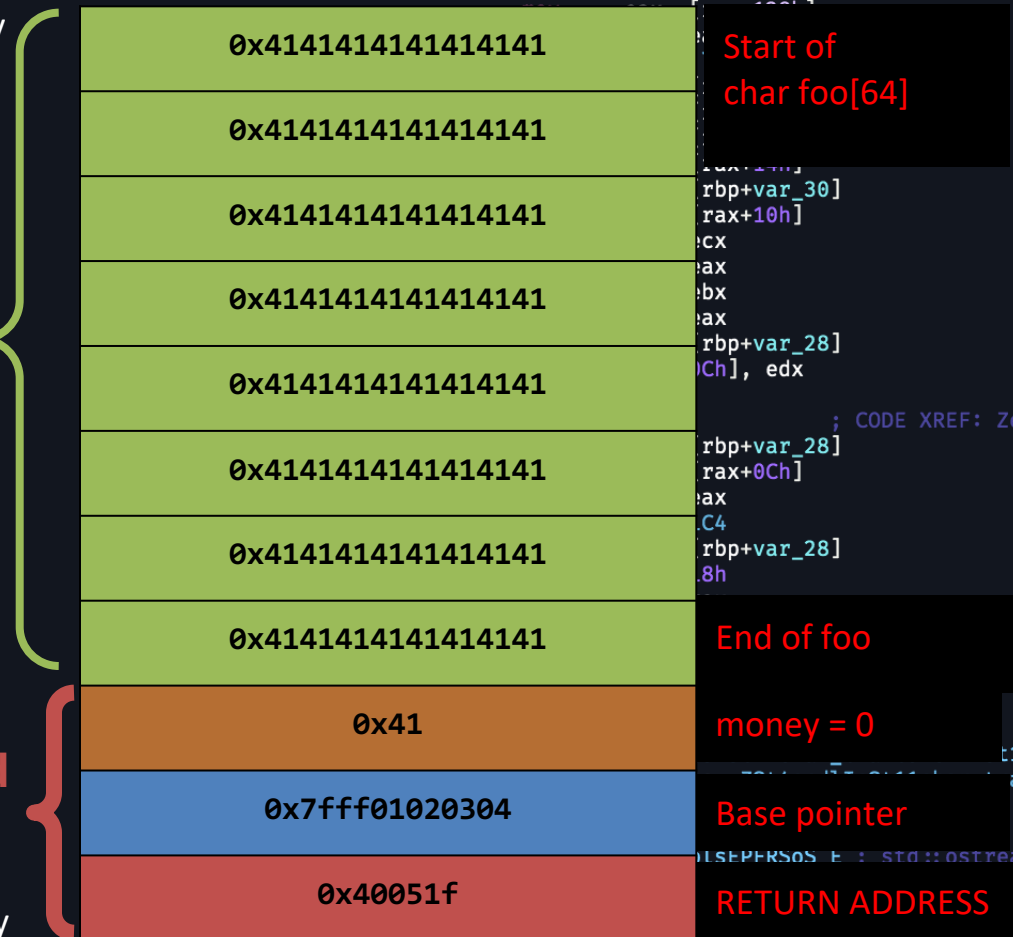
As gets() continues to read input, we fill up the 64 bytes allocated for **foo**

Go far enough, it corrupts important data!

Not supposed to touch

Higher Memory

Lower Memory



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

0x04	0x03	0x02	0x01
------	------	------	------

```
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
    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      edi, rax
    call     _ZStlsIcSt11char_traitsIcESaIc>@8
    mov      rdx, cs:_ZSt4endlIcSt11char_traitsIcESaIc>@8
    mov      rdi, rax
    call     __ZNSt11__ostream_t&lt;std::basic_ostream<char, std::basic_ostreambuf_iterator<char, std::allocator<char>>>>>::operator<<>(std::basic_ostream_t&, const char_t*)@8
    mov      rax, [rbp+var_28]
    mov      eax, [rax+8]
    test     eax, eax
    jnz      short loc_318D
    lea      rsi, aWannaCheatYes1 ; "wanna cheat?"
    mov      rax, cs:_ZSt4cout_ptr
    mov      rdi, rax
    call     _ZStlsIcSt11char_traitsIcESaIc>@8
    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
    mov     esi, eax
    mov     ebx, ebx
    add     edx, eax
    mov     rax, [rbp+var_28]
    mov     [rax+0Ch], edx
```

```
loc_30FB:
    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>@13
    lea     rsi, aIsDead ; " is dead!"
    mov     rdi, rax
    call    __ZStlsISt11char_traitsIcEERSt13
    mov     rdx, cs:_ZSt4endlIcSt11char_traitsIcESaIc>@13
    mov     rsi, rdx
    mov     rdi, rax
    call    __ZNSolsEPFRSoS_E@13 ; std::ostream
    mov     rax, [rbp+var_28]
    mov     eax, [rax+8]
    test    eax, eax
    jnz     short loc_31BD
    lea     rsi, aWannaCheatYes1 ; "wanna cheat?"
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    __ZStlsISt11char_traitsIcEERSt13
    lea     rax, [rbp+var_14]
```

Stack Smashing 201

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

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**)

```
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: Zei  
    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    __ZSt4cout@12@@@std::ostream_traitsIcESaIc>::operator<<@12@@@std::ostream_traitsIcESaIc>  
    lea     rsi, aIsDead ; " is dead!"  
    mov     rdi, rax  
    call    __ZSt4cout@12@@@std::ostream_traitsIcESaIc>::operator<<@12@@@std::ostream_traitsIcESaIc>  
    mov     rdx, rsi  
    mov     rsi, rdx  
    mov     rdi, rax  
    call    __ZNSoIsEPFRSoS_E ; std::ostream_traitsIcESaIc>::operator<<@12@@@std::ostream_traitsIcESaIc>  
    mov     rax, [rbp+var_28]  
    mov     eax, [rax+8]  
    test    eax, eax  
    jnz     short loc_31BD  
    lea     rsi, aWannaCheatYes1 ; "wanna cheat?"  
    mov     rax, cs:_ZSt4cout_ptr  
    mov     rdi, rax  
    call    __ZSt4cout@12@@@std::ostream_traitsIcESaIc>::operator<<@12@@@std::ostream_traitsIcESaIc>  
    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| 0x7fffffffef0f --> 0x7fffffffef10 --> 0x1
0008| 0x7fffffffef08 --> 0x0
0016| 0x7fffffffef10 --> 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: Zei
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>@PLT
lea         rsi, aIsDead ; " is dead!"
mov         rdi, rax
call        __ZStlsISt11char_traitsIcEERSt13
mov         rdx, cs:_ZSt4endlIcSt11char_tra
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_318D
lea         rsi, aWannaCheatYes1 ; "wanna ch
mov         rax, cs:_ZSt4cout_ptr
mov         rdi, rax
call        __ZStlsISt11char_traitsIcEERSt13
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| 0x7fffffffef0 --> 0x7fffffffef0 --> 0x1
0008| 0x7fffffffef8 --> 0x0
0016| 0x7fffffffef100 --> 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| 0x7fffffffef0e8 --> 0x40051f (<main+18>: mov     var_28]
0008| 0x7fffffffef0f0 --> 0x7fffffffef100 --> 0x1
0016| 0x7fffffffef0f8 --> 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
[-----]
0000| 0x7fffffffef0e8 --> 0x40051f (<main+18>:
0008| 0x7fffffffef0f0 --> 0x7fffffffef0f0
0016| 0x7fffffffef0f8 --> 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 ecx, [rax+14h]
mov rax, [rbp+var_30]
mov eax, [rax+10h]
mov ebx, ecx
sub ebx, eax
add edx, eax
mov rax, [rbp+var_28]
mov [rax+0Ch], edx
```

```
loc_30FB: ; CODE XREF: Zei
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_traitsIcEERSt13
mov rdx, cs:_ZSt4endlIcSt11char_tra
mov rsi, rdx
mov rdi, rax
call __ZNSoIsEPFRSoS_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 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

```
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_30F5
    mov     ecx, [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     eax, ecx
    mov     rax, [rbp+var_28]
    mov     [rax+0Ch], edx
```

```
loc_30FB:                                     ; CODE XREF: Zei
    mov     rax, [rbp+var_28]
    mov     eax, [rax+0Ch]
```

```
=> 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| 0x7fffffffef08 --> 0x40051f (<main+18>:
0008| 0x7fffffffef0f --> 0x7fffffffef0f --> 0;
0016| 0x7fffffffef08 --> 0x0
```

```
=> 0x40051f <main+18>:  mov     DWORD PTR [rbp-0x4], eax
0x400522 <main+21>:    mov     eax, 0x0
0x400527 <main+26>:    leave
0x400528 <main+27>:    ret
0x400529:             nop     DWORD PTR [rax+0x0]

[-----stack-----]
0000| 0x7fffffffef0f --> 0x7fffffffef0f --> 0x1
0008| 0x7fffffffef08 --> 0x0
0016| 0x7fffffffef10 --> 0x400530 (<_libc_csu init>:
    call    __ZStlsISt11char_traitsIcEERSt13
    mov     rdx, cs:_ZSt4endlIcSt11char_trai
    mov     rsi, rdx
    mov     rdi, rax
    call    __ZNSoIsEPFRSoS_E ; std::ostream
    mov     rax, [rbp+var_28]
    mov     eax, [rax+8]
    test    eax, eax
    jnz     short loc_318D
    lea     rsi, aWannaCheatYes1 ; "wanna ch
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    __ZStlsISt11char_traitsIcEERSt13
    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: **What if we change this???**
Returned back to main:

```
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     ecx, [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     edx, eax
    add     edx, ebx
    mov     rax, [rbp+var_28]
    mov     [rax+0Ch], edx
    loc_30FB:
    mov     rax, [rbp+var_28]
    mov     eax, [rax+0Ch]
    test    eax, eax
    jnz     short loc_30FB
    mov     DWORD PTR [rbp+0x4], eax
    mov     eax, 0x0
    leave   [rbp+var_28]
    ret
0x400529:  nop     DWORD PTR [rax+0x0]
    t4cout_ptr
    stack
0x000| 0x7fffffffef08 --> 0x40051f (<main+18>:
0x008| 0x7fffffffef0f --> 0x7fffffffef0f --> 0
0x016| 0x7fffffffef0f --> 0x0
    stack
0x000| 0x7fffffffef0f --> 0x7fffffffef0f --> 0x1
0x008| 0x7fffffffef0f --> 0x0
0x016| 0x7fffffffef10 --> 0x400530 (<_libc_csu init>:
    call    __ZStlsISt11char_traitsIcEERSt13
    mov     rdx, cs:_ZSt4endlIcSt11char_trai
    mov     rsi, rdx
    mov     rdi, rax
    call    __ZNSoIsEPFRSoS_E ; std::ostream
    mov     rax, [rbp+var_28]
    mov     eax, [rax+8]
    test    eax, eax
    jnz     short loc_30FB
    lea     rsi, aWannaCheatYes1 ; "wanna ch
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    __ZStlsISt11char_traitsIcEERSt13
    lea     rax, [rbp+var_14]
```

??!?!?

Stack Smashing 201

- 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

Lower Memory

Higher Memory

0x4141414141414141	Start of char foo[64]
0x4141414141414141	
0x4141414141414141	
0x4141414141414141	
0x4141414141414141	
0x4141414141414141	
0x4141414141414141	
0x4141414141414141	End of foo
0x00	money = 0
0x7fff01020304	Base pointer
0x40051f	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]
```

```
    jnz     short loc_318D
    lea     rsi, aWannaCheatYes1 ; "wanna cheat?"
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    _ZStlsISt11char_traitsIcEERSt13basic_stringbuf
    lea     rax, [rbp+var_14]
```


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]
```

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

Lower Memory

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

0x4141414141414141

Higher Memory

Start of
char foo[64]

End of foo

money = 0

Base pointer

RETURN ADDRESS

```
    jnz     short loc_318D
    lea     rsi, aWannaCheatYes1 ; "wanna ch
    mov     rax, cs:_ZSt4cout_ptr
    mov     rdi, rax
    call    _ZStlsISt11char_traitsIcEERSt13
    lea     rax, [rbp+var_14]
```