CSCI2467: Systems Programming Concepts

Bonus target!

Spring 2020

THE UNIVERSITY of
NEW ORLEANS

DEPARTMENT OF
COMPUTER SCIENCE

Getting started

- Copy `btarget` from `/home/CSCI2467/labs/bonustarget/btarget`.
- Look at code on page 2 of activity PDF (see schedule page).

Introduction to `solve()`

Let's look at `solve()` in the `src/activity.c` file.

What is it doing?

Is it possible for the program to call `win()`?

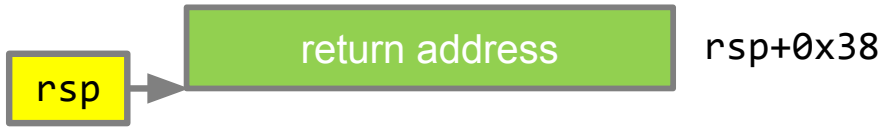
```
void solve(void) {
    long before = 0xb4;
    char buf[16];
    long after = 0xaf;

    Gets(buf);

    if (before == 0x3331323531)
        win(0x15213);

    if (after == 0x3331323831)
        win(0x18213);
}
```

Diving into assembly

- Look at the disassembly of `solve()`.
- Try drawing a stack diagram.
  - How large is the stack frame?
  - Where is the saved return address?
  - Where are `before`, `buf`, and `after`?
- **Which variable will be overwritten if we perform a buffer overflow, `before` or `after`?**

Drawing the stack diagram

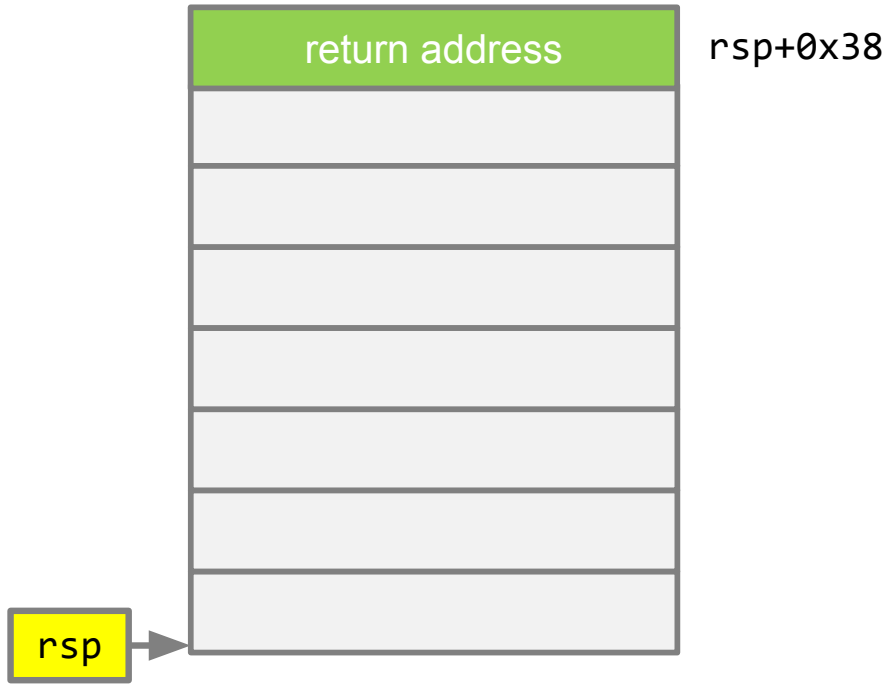
```
=> 0x4006b5 <+0>:     sub    $0x38,%rsp
```



Addresses
increase towards
the top of the slide

Drawing the stack diagram

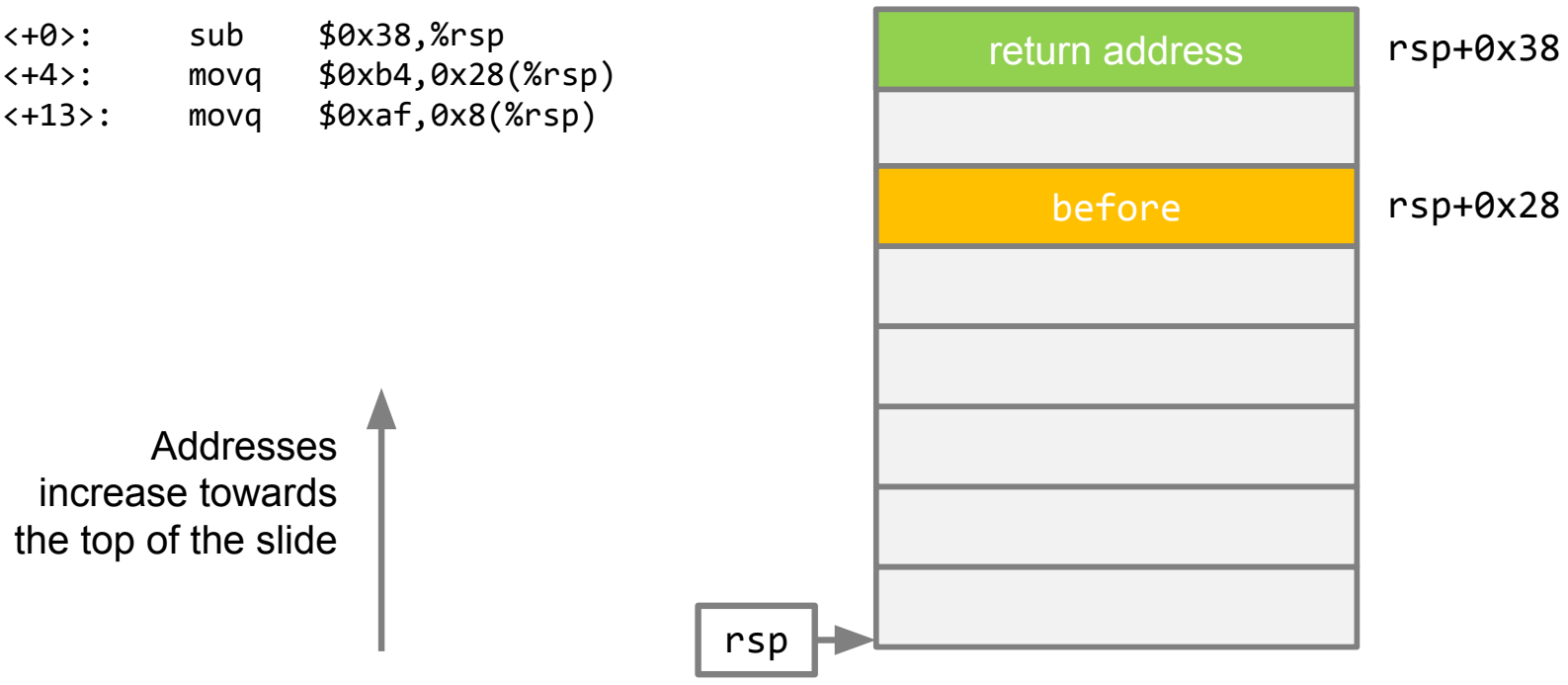
```
   0x4006b5 <+0>:     sub    $0x38,%rsp
=> 0x4006b9 <+4>:     movq   $0xb4,0x28(%rsp)
```



Addresses
increase towards
the top of the slide

Drawing the stack diagram

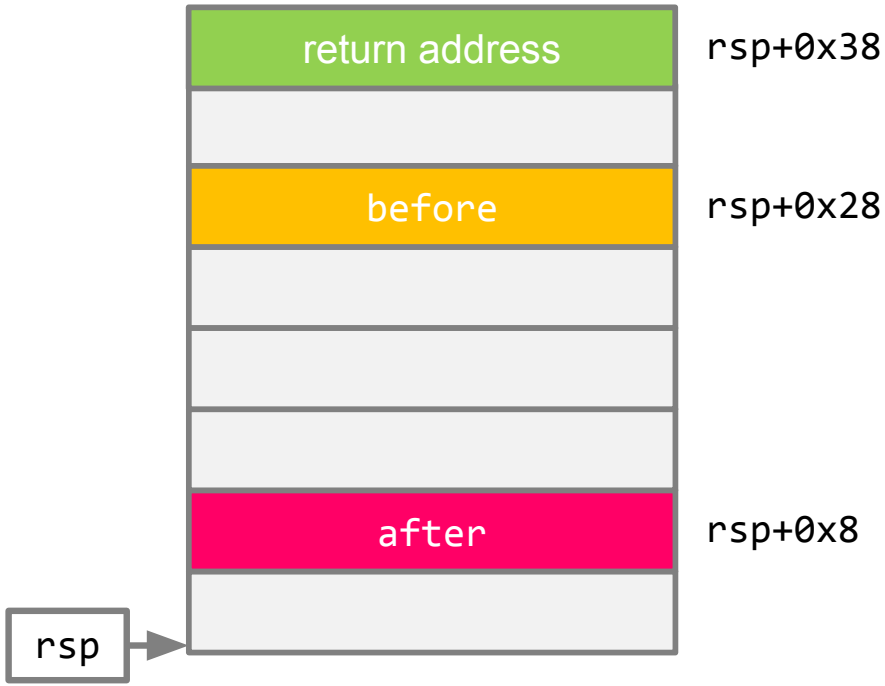
```
   0x4006b5 <+0>:     sub    $0x38,%rsp
   0x4006b9 <+4>:     movq   $0xb4,0x28(%rsp)
=> 0x4006c2 <+13>:    movq   $0xaf,0x8(%rsp)
```



Drawing the stack diagram

```
   0x4006b5 <+0>:     sub    $0x38,%rsp
   0x4006b9 <+4>:     movq   $0xb4,0x28(%rsp)
   0x4006c2 <+13>:    movq   $0xaf,0x8(%rsp)
   0x4006cb <+22>:    lea    0x10(%rsp),%rdi
=> 0x4006d0 <+27>:    callq  0x40073f <Gets>
```

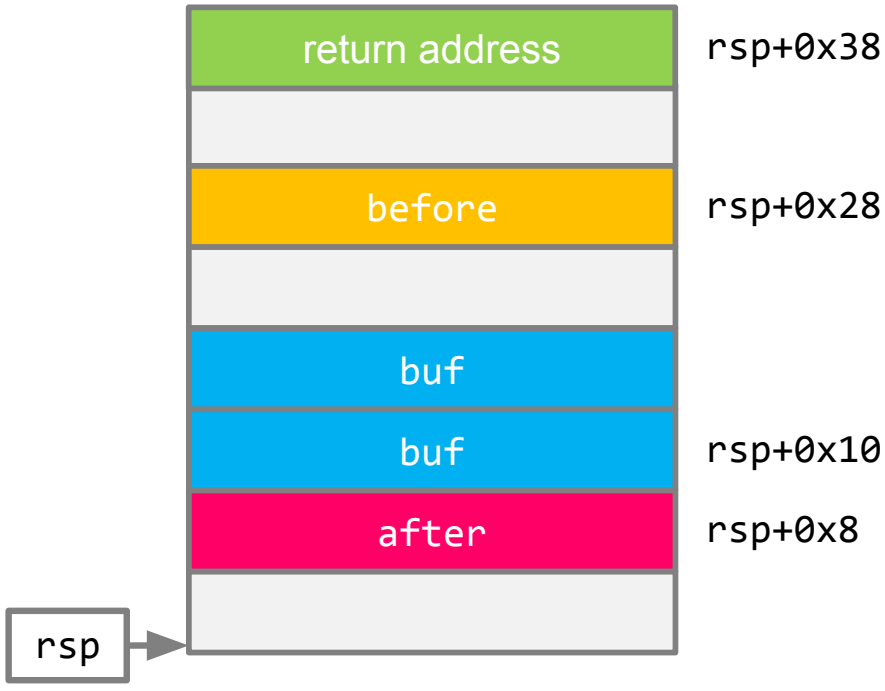
Addresses increase towards the top of the slide



Drawing the stack diagram

```
   0x4006b5 <+0>:     sub    $0x38,%rsp
   0x4006b9 <+4>:     movq   $0xb4,0x28(%rsp)
   0x4006c2 <+13>:    movq   $0xaf,0x8(%rsp)
   0x4006cb <+22>:    lea    0x10(%rsp),%rdi
   0x4006d0 <+27>:    callq  0x40073f <Gets>
=> 0x4006d5 <+32>:    mov    0x28(%rsp),%rdx
```

Addresses increase towards the top of the slide

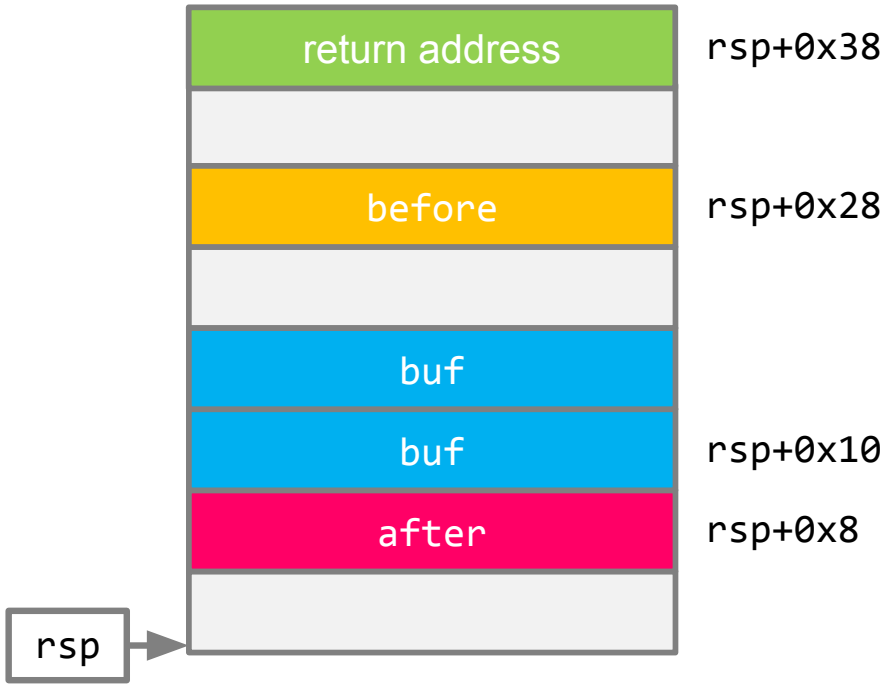


Drawing the stack diagram

we drew with the actual values on the stack after `Gets()` returns.

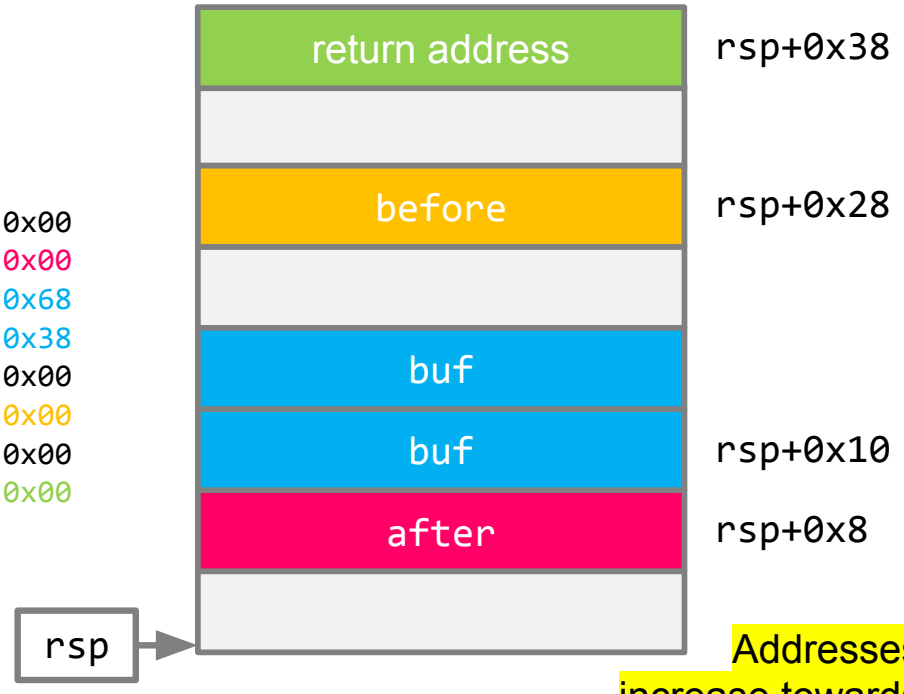
```
   0x4006d0 <+27>:    callq  0x40073f <Gets>
=> 0x4006d5 <+32>:    mov    0x28(%rsp),%rdx
```

```
(gdb) break *0x4006d5
(gdb) run
Starting program: act1
abcdefgh12345678
(gdb) x/8gx $rsp
(gdb) x/64bx $rsp
```



Drawing the stack diagram

```
0x602030:    0x6867666564636261    0x3837363534333231
0x602040:    0x0000000000000000    0x00000000000000b4
0x602050:    0x0000000000000000    0x0000000000400783

(gdb) x/64bx $rsp
0x602020:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x602028:  0xaf  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x602030:  0x61  0x62  0x63  0x64  0x65  0x66  0x67  0x68
0x602038:  0x31  0x32  0x33  0x34  0x35  0x36  0x37  0x38
0x602040:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x602048:  0xb4  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x602050:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x602058:  0x83  0x07  0x40  0x00  0x00  0x00  0x00  0x00
```



Addresses increase towards **bottom of the slide**

Addresses increase towards **top of the slide**

Summary so far

- Buffer overflows can **overwrite** parts of the stack frame, include other local variables
- Stack frames may include *padding*, so looking at the assembly is crucial to drawing a correct diagram
- GDB prints output starting at the *lowest* address, but our (and most) stack diagrams start at the *highest*

Students: this part is for you!

- Can you get `btarget` to print:

  `You win 1 cookie!  Great start!` ?
- that is, can you make it call `win(0x15213)` ?
- More importantly, can you show the class and explain?
- One ★ is at stake!

Ready for more of a challenge?

- Can you get `btarget` to print:

  `You win 2 cookies!  Woohoo! ?`
- More importantly, can you show the class and explain?
- Two ★★ are at stake!

- How about 3 cookies? 4 cookies?? How far does this go?