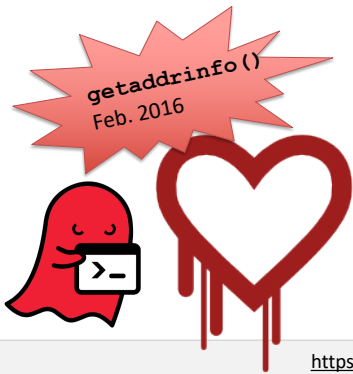




# Buffer Overflows



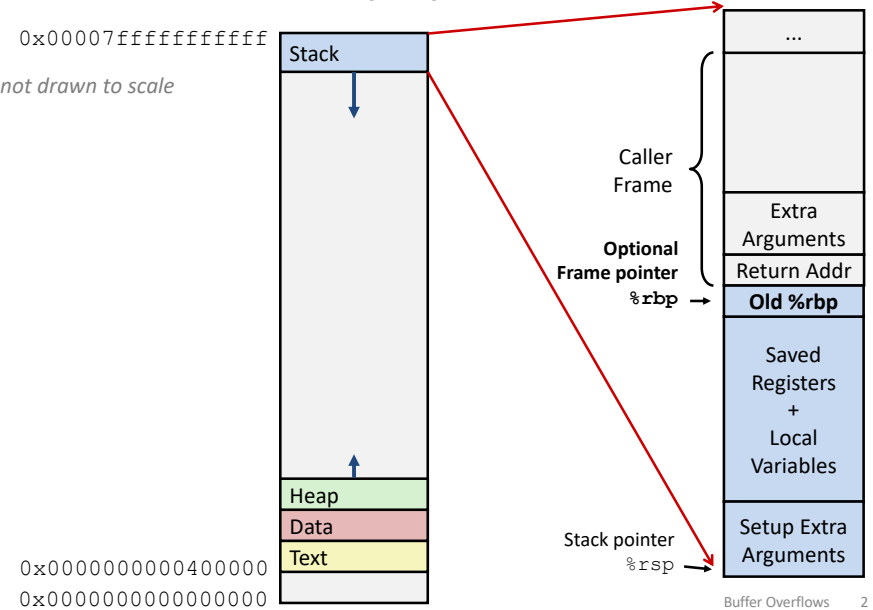
Address space layout  
the stack discipline  
+ C's lack of bounds-checking  
HUGE PROBLEM

<https://cs.wellesley.edu/~cs240/s20/>

## x86-64 Linux memory layout

0x00007fffffffffff

not drawn to scale



## String library code

C standard library function `gets()`

```
/* Get string from stdin */
char* gets(char* dest) {
    int c = getchar();
    char* p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return dest;
}
```

pointer to start of an array

same as:  
`*p = c;`  
`p = p + 1;`

What could go wrong in this code?

Same problem in many functions:

- strcpy:** Copies string of arbitrary length
- scanf, fscanf, sscanf,** when given `%s` conversion specification

## Vulnerable buffer code: C

```
/* Echo Line */
void echo() {
    char buf[4]; /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
int main() {
    printf("Type a string:");
    echo();
    return 0;
}
```

```
$ ./bufdemo
Type a string:123
123
```

```
$ ./bufdemo
Type a string: 0123456789012345678901234
Segmentation Fault
```

```
$ ./bufdemo
Type a string: 012345678901234567890123
012345678901234567890123
```

## Vulnerable buffer code: disassembled x86

**echo code**

```

0000000004006cf <echo>:
4006cf: 48 83 ec 18      sub    $24,%rsp
4006d3: 48 89 e7         mov    %rsp,%rdi
4006d6: e8 a5 ff ff ff  callq 400680 <gets>
4006db: 48 89 e7         mov    %rsp,%rdi
4006de: e8 3d fe ff ff  callq 400520 <puts@plt>
4006e3: 48 83 c4 18     add    $24,%rsp
4006e7: c3              retq
    
```

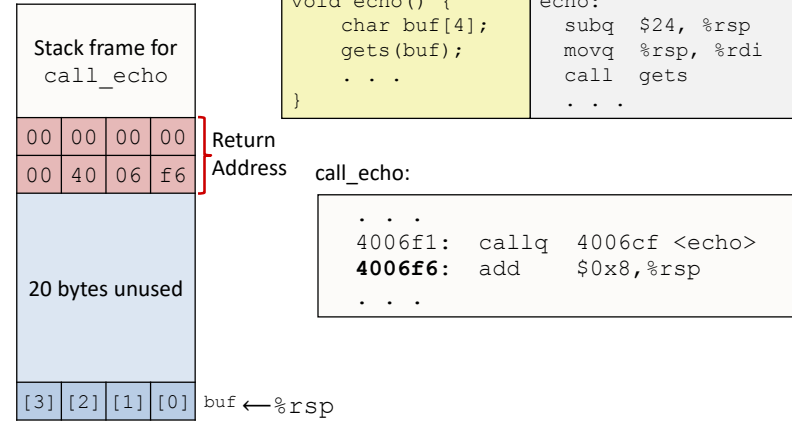
**caller code**

```

4006e8: 48 83 ec 08     sub    $0x8,%rsp
4006ec: b8 00 00 00 00  mov    $0x0,%eax
4006f1: e8 d9 ff ff ff  callq 4006cf <echo>
4006f6: 48 83 c4 08     add    $0x8,%rsp
4006fa: c3              retq
    
```

## Buffer overflow example: before input

Before call to gets



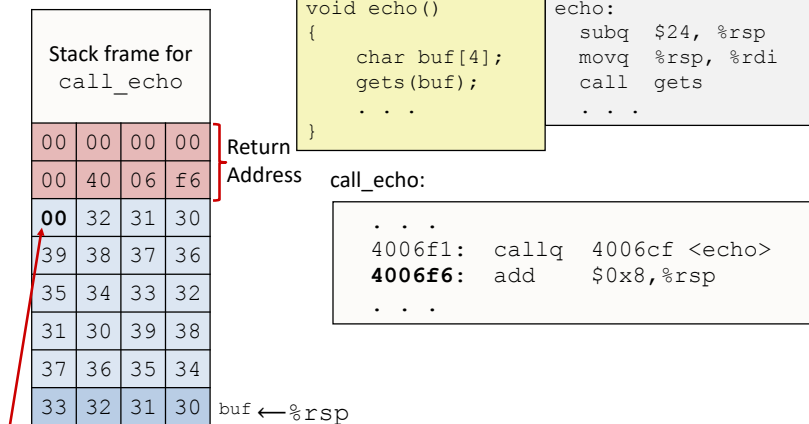
```

void echo() {
    char buf[4];
    gets(buf);
    ...
}

echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    ...
    
```

## Buffer overflow example: input #1

After call to gets



Null Terminator

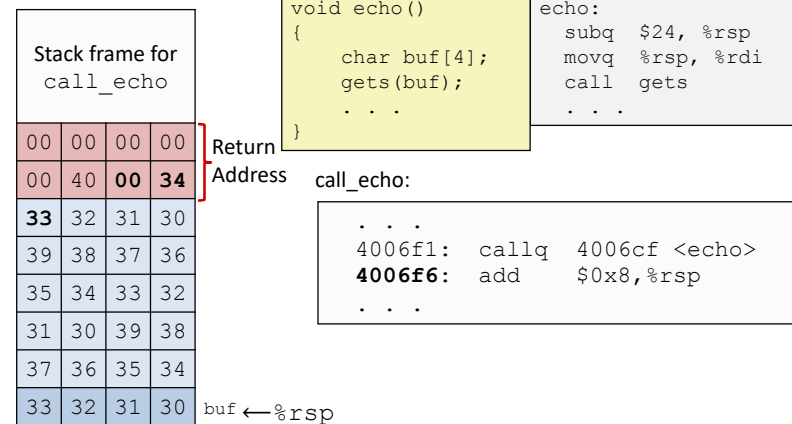
```

$ ./bufdemo
Type a string: 01234567890123456789012
01234567890123456789012
    
```

Overflowed buffer, but did not corrupt state

## Buffer overflow example: input #2

After call to gets



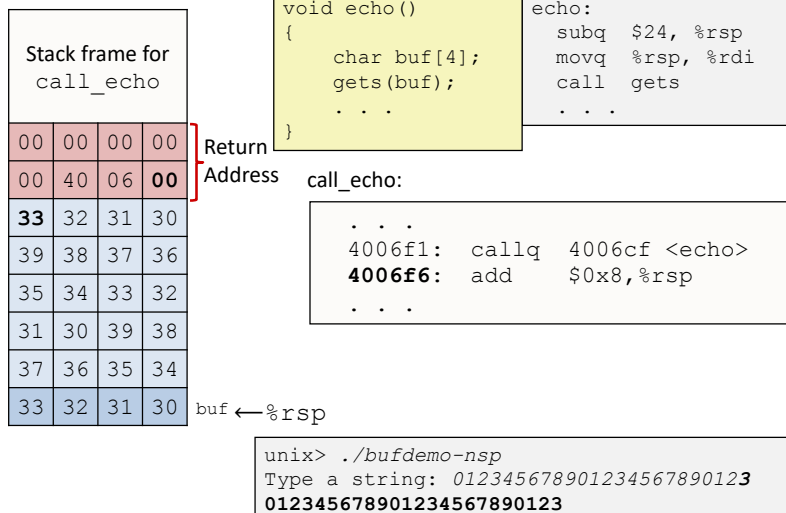
```

unix> ./bufdemo
Type a string: 0123456789012345678901234
Segmentation Fault
    
```

Overflowed buffer and corrupted return pointer

## Buffer overflow example: input #3

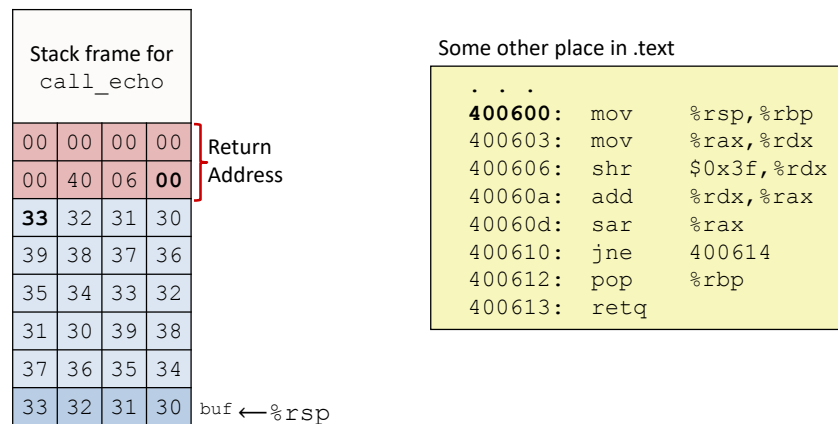
After call to gets



Overflowed buffer, corrupted return pointer, but program seems to work! 9

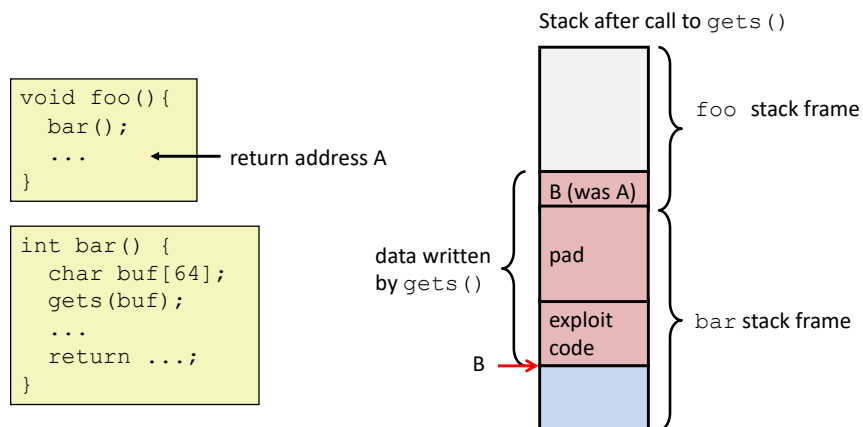
## Buffer overflow example: input #3

After call to gets



“Returns” to unrelated code  
Lots of things happen, without modifying critical state  
Eventually executes retq back to main

## Exploiting buffer overflows



Input string contains byte representation of executable code  
Overwrite return address A with address of buffer (need to know B)  
When bar () executes ret, will jump to exploit code (instead of A)

## Exploits in the wild

*Buffer overflow bugs allow remote attackers to execute arbitrary code on machines running vulnerable software.*

1988: Internet worm

Early versions of the finger server daemon (fingerd) used gets () to read the argument sent by the client:

```
finger somebody@cs.wellesley.edu
```

*commandline facebook of the 80s!*

Attack by sending phony argument:

```
finger "exploit-code padding new-return-address"
```

...

Still happening

"Ghost:" 2015



gethostname ()

getaddrinfo ()  
Feb. 2016

# Heartbleed (2014)

optional

Buffer over-read in OpenSSL  
Widely used encryption library (https)

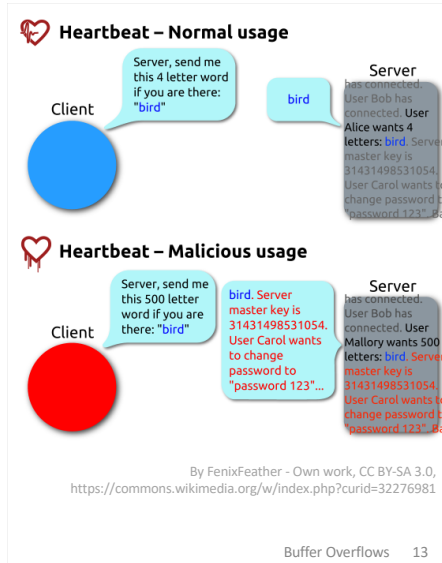
## “Heartbeat” packet

Specifies length of message  
Server echoes that much back  
Library just “trusted” this length  
Allowed attackers to read contents of memory anywhere they wanted

~17% of Internet affected

“Catastrophic”

Github, Yahoo, Stack Overflow, Amazon AWS, ...



# Avoiding overrun vulnerabilities

1. Use a memory-safe language (not C)!
2. If you have to use C, use library functions that limit string lengths.

**fgets** instead of **gets**

```
/* Echo Line */
void echo() {
    char buf[4]; /* Way too small! */
    fgets(buf, 4, stdin);
    puts(buf);
}
```

**strncpy** instead of **strcpy**

Don't use **scanf** with **%s** conversion specification

Use **fgets** to read the string

Or use **%ns** where **n** is a suitable integer

*Other ideas?*

# System-level protections

Available in modern OSs/compilers/hardware  
(We disabled these for buffer assignment.)

1. Randomize stack base, maybe frame padding
2. Detect stack corruption  
save and check stack "canary" values
3. Non-executable memory segments  
stack, heap, data, ... everything except text  
hardware support

Helpful, not foolproof!

Return-oriented programming, over-reads, etc.

not drawn to scale

