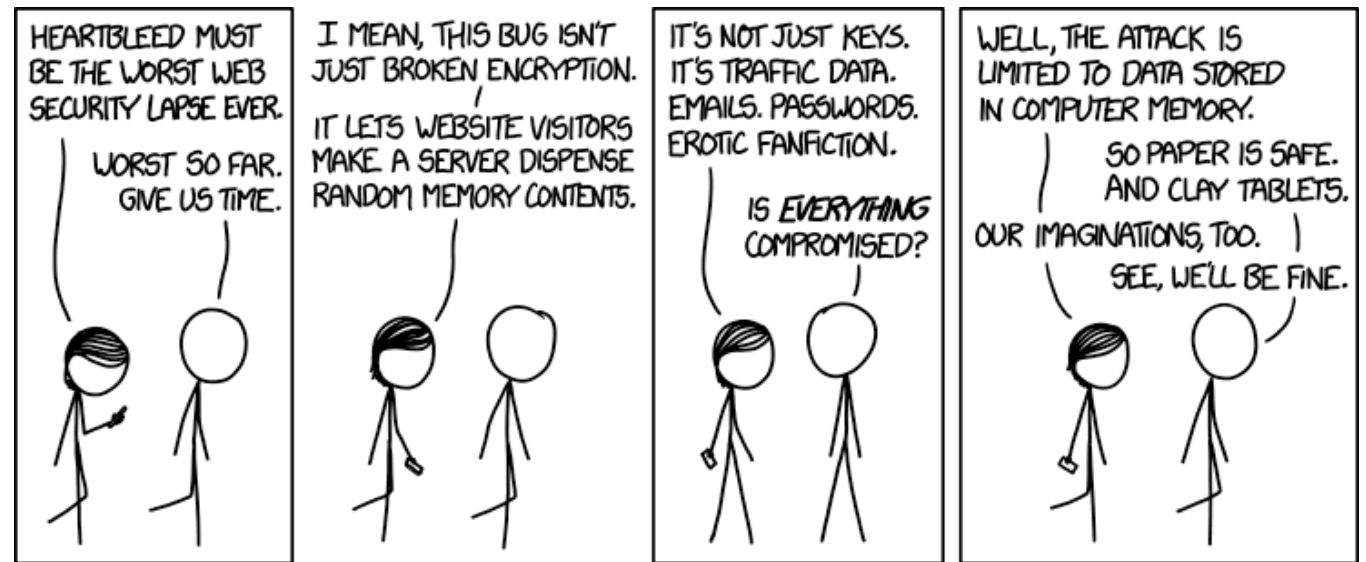# Buffer Overflow
CSE 351 Winter 2024

Instructor:

Justin Hsia

Teaching Assistants:

Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

Alt text: I looked at some of the data dumps from vulnerable sites, and it was … bad. I saw emails, passwords, password hints. SSL keys and session cookies. Important servers brimming with visitor IPs. Attack ships on fire off the shoulder of Orion, c-beams glittering in the dark near the Tannhäuser Gate. I should probably patch OpenSSL.
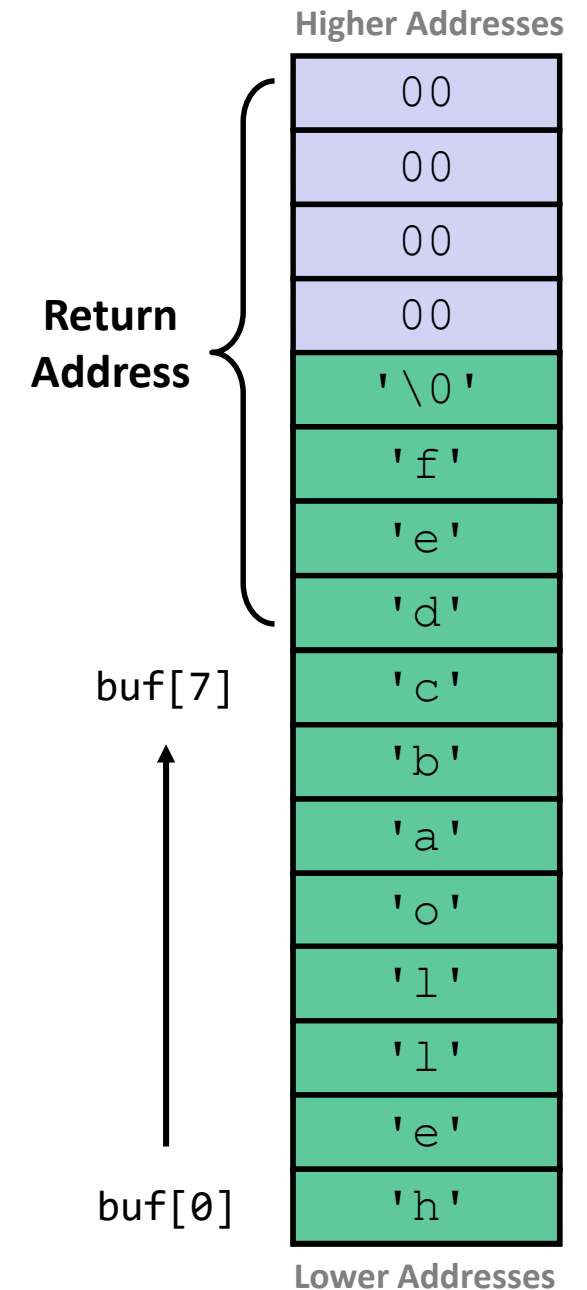
http://xkcd.com/1353/

Relevant Course Information

❖ Mid-quarter survey due tonight!

❖ HW12 due tonight, HW13 due Wednesday, HW14 due Monday (2/12)

❖ Lab 3 released today, due next Friday (2/16)
- You will have everything you need by the end of this lecture

❖ Midterm starts Thursday (Friday lecture is extra support hour)
- Instructions will be posted on Ed Discussion
- Gilligan's Island Rule: discuss high-level concepts and give hints, but not solving the problems together
- Ed Discussion (private posts) and support hours to answer clarifying questions
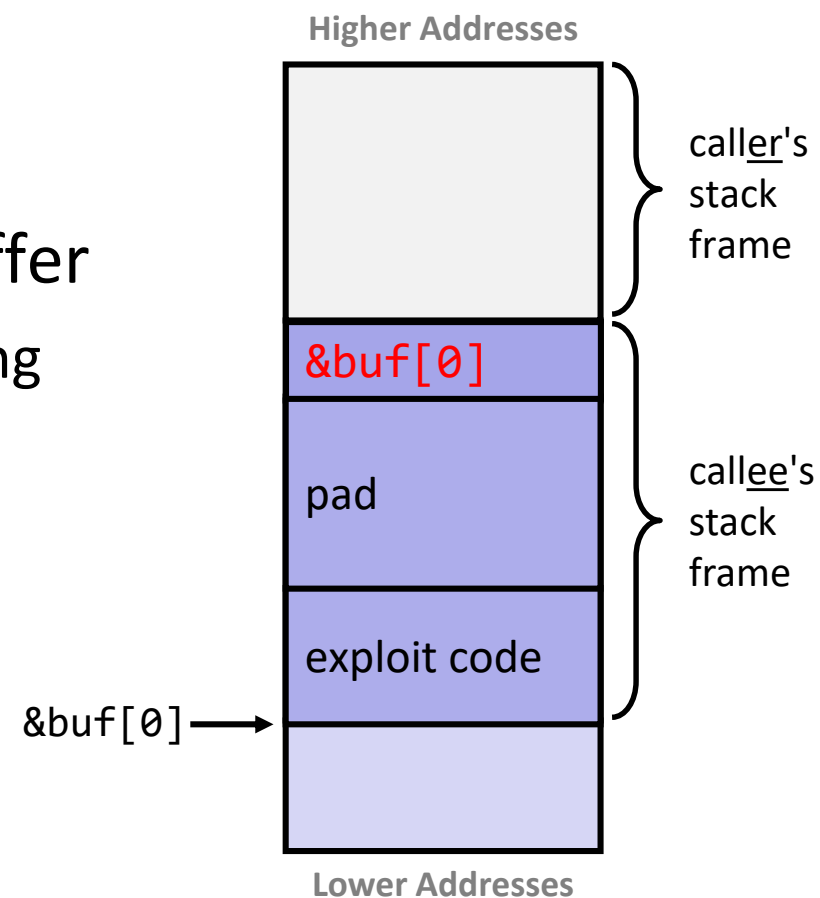
Buffer Overflow

Lesson Summary (1/3)

❖ A **buffer** is an array that holds temporary data (*e.g.*, user/file/network input)

❖ **Buffer overflow** is writing past the end of the buffer

- Common in C/Unix/Linux due to lack of bounds checking
- Vulnerable functions include `gets`, `strcpy`, `scanf`, `fscanf`, `sscanf`

❖ Buffer overflow exploit: **stack smashing**

- Overflow local array to alter stack contents
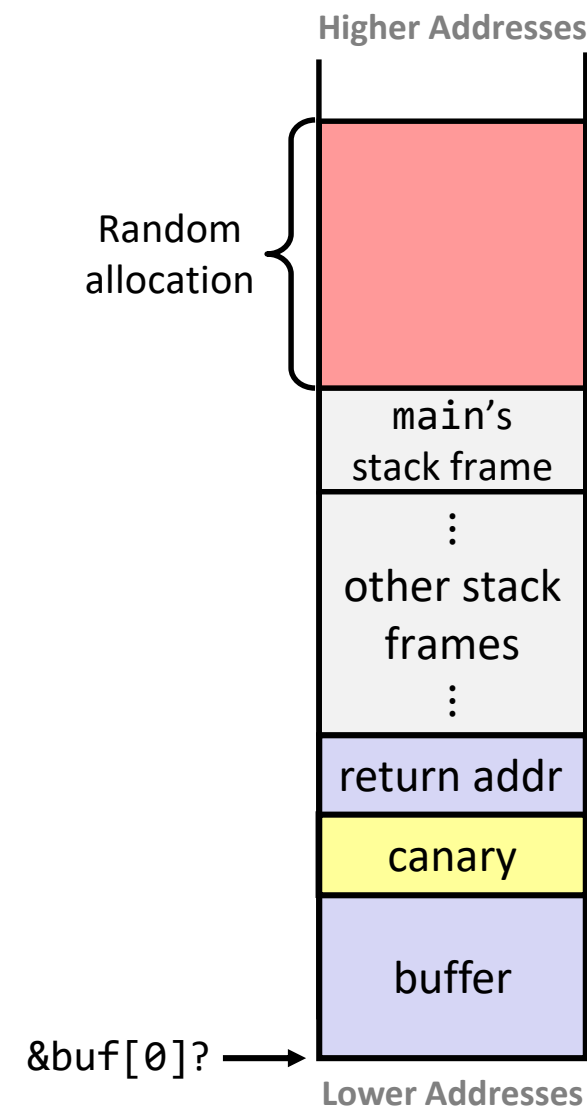- Commonly used to alter procedure return address

Higher Addresses

00
00
00
00
'\0'
'f'
'e'
'd'
'c'
'b'
'a'
'o'
'l'
'l'
'e'
'h'

Return Address { 00, 00, 00, 00, '\0', 'f', 'e', 'd' }

buf[7] → 'c'

buf[0] → 'h'

Lower Addresses

Lesson Summary (2/3)

❖ A **buffer** is an array that holds temporary data (*e.g.*, user/file/network input)

❖ **Buffer overflow** is writing past the end of the buffer

- Common in C/Unix/Linux due to lack of bounds checking
- Vulnerable functions include `gets`, `strcpy`, `scanf`, `fscanf`, `sscanf`

❖ Buffer overflow exploit: **code injection**

1) Put exploit/machine code in buffer
2) Pad to reach stack frame's return address
3) Replace return address with address of the buffer

Higher Addresses

&buf[0]

pad

exploit code

&buf[0] →

Lower Addresses

caller's stack frame
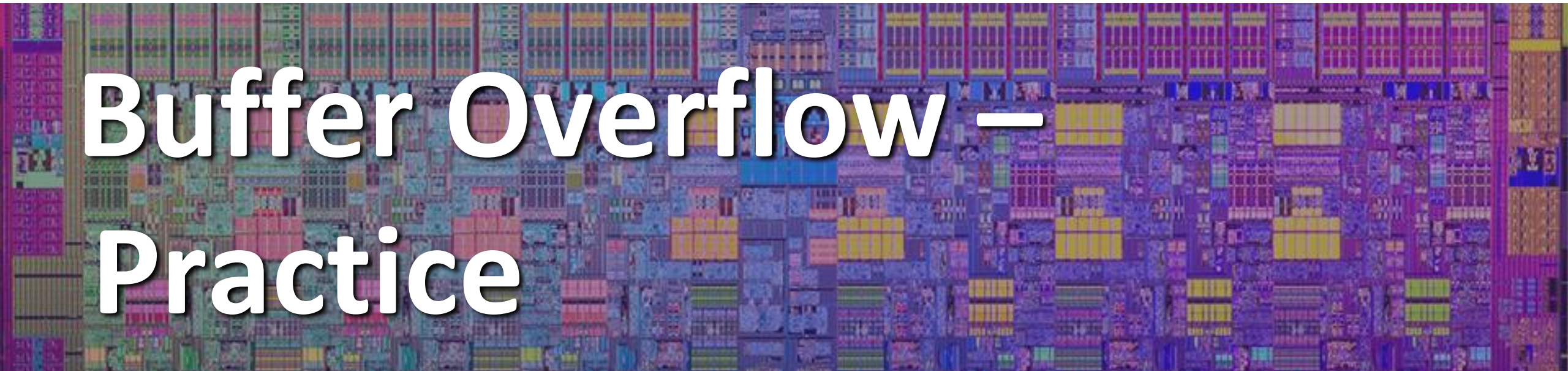
callee's stack frame

Lesson Summary (3/3)

❖ Dealing with buffer overflow attacks

 ▪ Use array bounds checking

  • Manually (*i.e.*, implement yourself) or automatically (*e.g.*, use safe functions or non-C language)

 ▪ Add a stack canary after the buffer

  • Secret value (changes on each execution) that shouldn't change

 ▪ Randomized stack offsets

  • Makes finding the address of exploit code more difficult

 ▪ Non-executable memory regions (*e.g.*, the stack)

  • Prevent exploit code from being placed and executed there

Higher Addresses

Random allocation

main's stack frame

⋮

other stack frames

⋮

return addr

canary

buffer

&buf[0]? →

Lower Addresses

Lesson Q&A

❖ Learning Objectives:

- Define buffer overflow and explain how it occurs.

- Identify elements of C programs that make them vulnerable to buffer overflow.

- Identify methods of protecting against buffer overflow.

- Perform stack smashing and code injection exploits.

❖ What lingering questions do you have from the lesson?

- Chat with your neighbors about the lesson for a few minutes to come up with questions

Buffer Overflow – Practice

Polling Question

- ❖ `smash_me` is vulnerable to stack smashing!

- ❖ What is the minimum number of characters that `gets` must read in order for us to change the return address to a stack address?

  - For example: (0x 00 00 7f ff ca fe f0 0d)

Previous stack frame			
00	00	00	00
00	40	05	d1
. . .			
			[0]
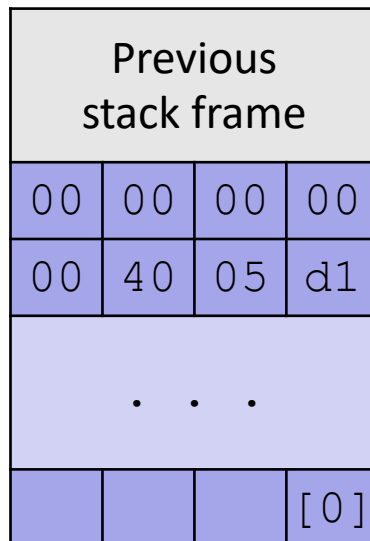
```
smash_me:
   subq   $0x40, %rsp
   ...
   leaq   16(%rsp), %rdi
   call   gets
   ...
```

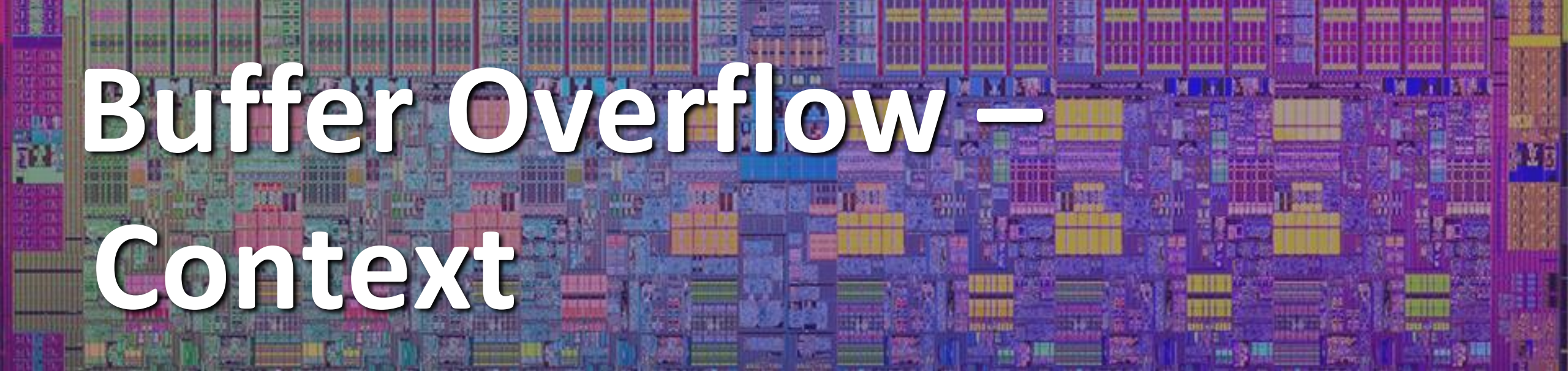A. **27**
B. **30**
C. **51**
D. **54**
E. **We're lost...**

Lab Setup/Demo

❖ Printable vs. nonprintable characters (asciitable.com)

- Input from terminal generally restricted to printable characters (~ 0x20 – 0x7E)
- Need full range of 1-byte character values (0x00 – 0xFF) for exploits

❖ Lab 3 workflow:

- Design exploit string for task
- Type out exploit string as printable characters in `.txt` file
- Use `sendstring` to convert `.txt` → `.bytes` file
- Pass `.bytes` file into `bufbomb` to be read by `Gets()`
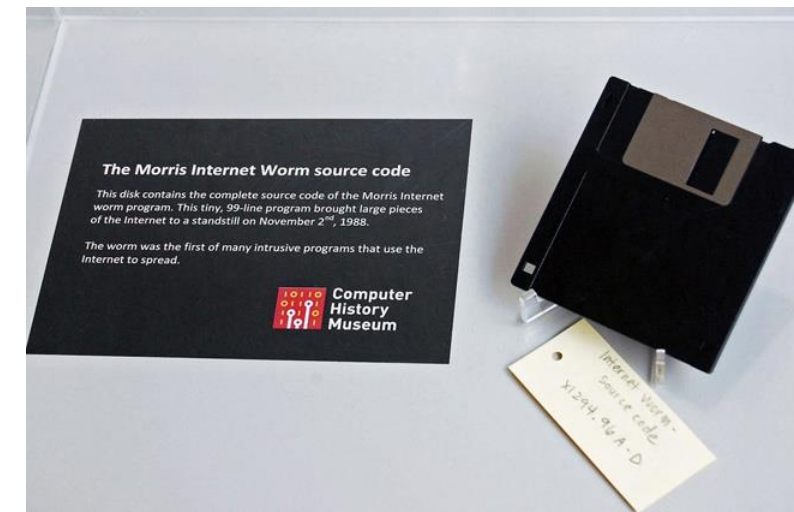- Use GDB to verify that stack is modified as desired

Buffer Overflow – Context

Exploits Based on Buffer Overflows

- ⚠ Buffer overflow bugs can allow attackers to execute arbitrary code on victim machines ☠
  - Most commonly executing a "root shell" – terminal with elevated privileges

- Distressingly common in real programs
  - Original "Internet worm" (1988)
  - Heartbleed (2014, affected 17% of servers) & Cloudbleed (2017)
  - Hacking embedded devices (e.g., cars, smart home devices, Internet of Things)
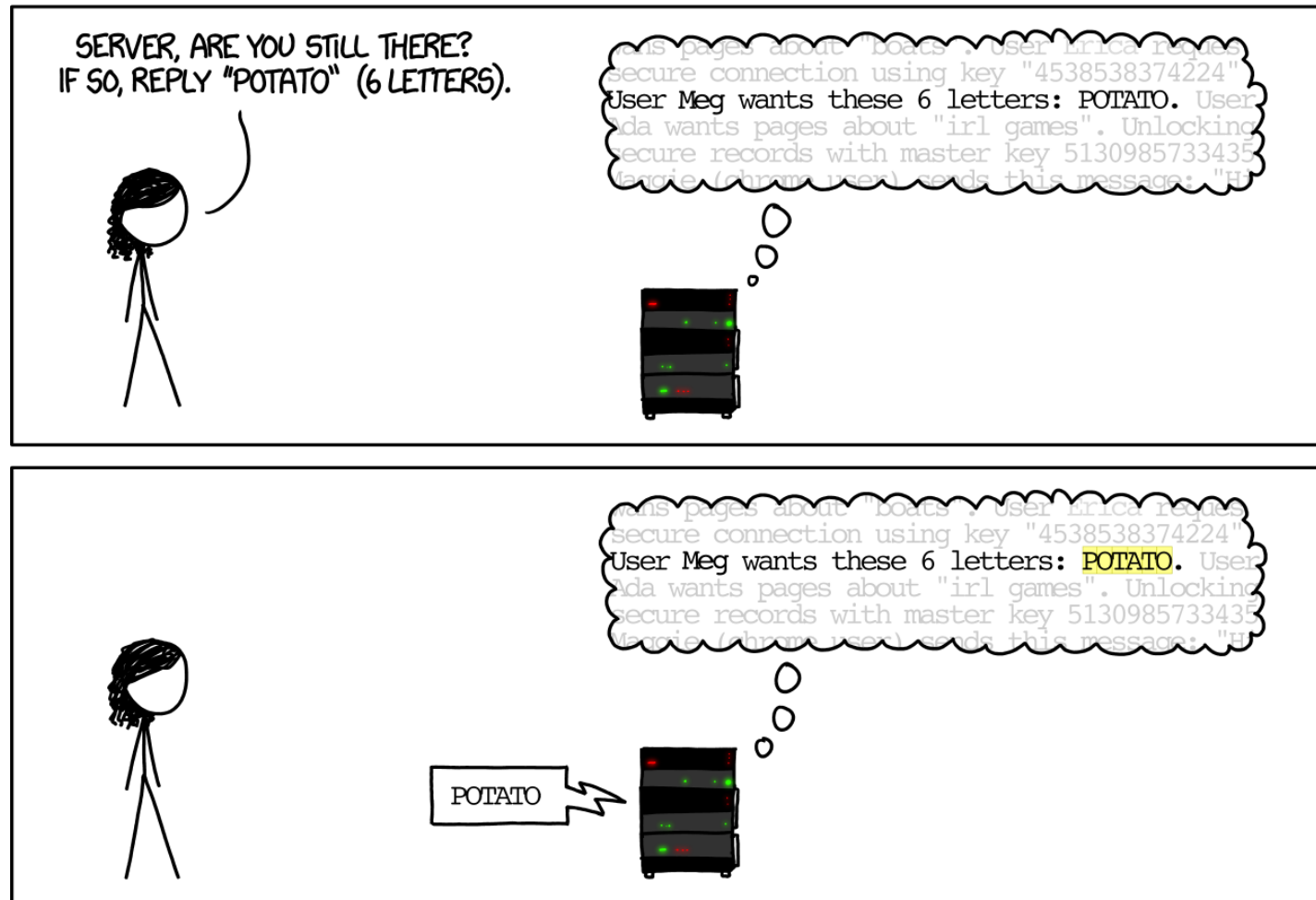
The Morris Worm (1988)

❖ Early versions of the finger server (<u>fingerd</u>) used `gets` to read the argument sent by the client

  ▪ *e.g.,* `finger` <u>droh@cs.cmu.edu</u>

❖ The Morris Worm attacked `fingerd` server with phony argument:

  ▪ `finger "exploit-code padding new-return-addr"`

  ▪ Exploit code executed a root shell on the victim machine, then scanned for other machines to attack

❖ Fallout/legacy (<u>1989 article</u>)

  ▪ Invaded ~6000 computers in hours (10% of the Internet)

  ▪ The author, <u>Robert Morris</u>, was prosecuted

    • First conviction under 1986 Computer Fraud and Abuse Act
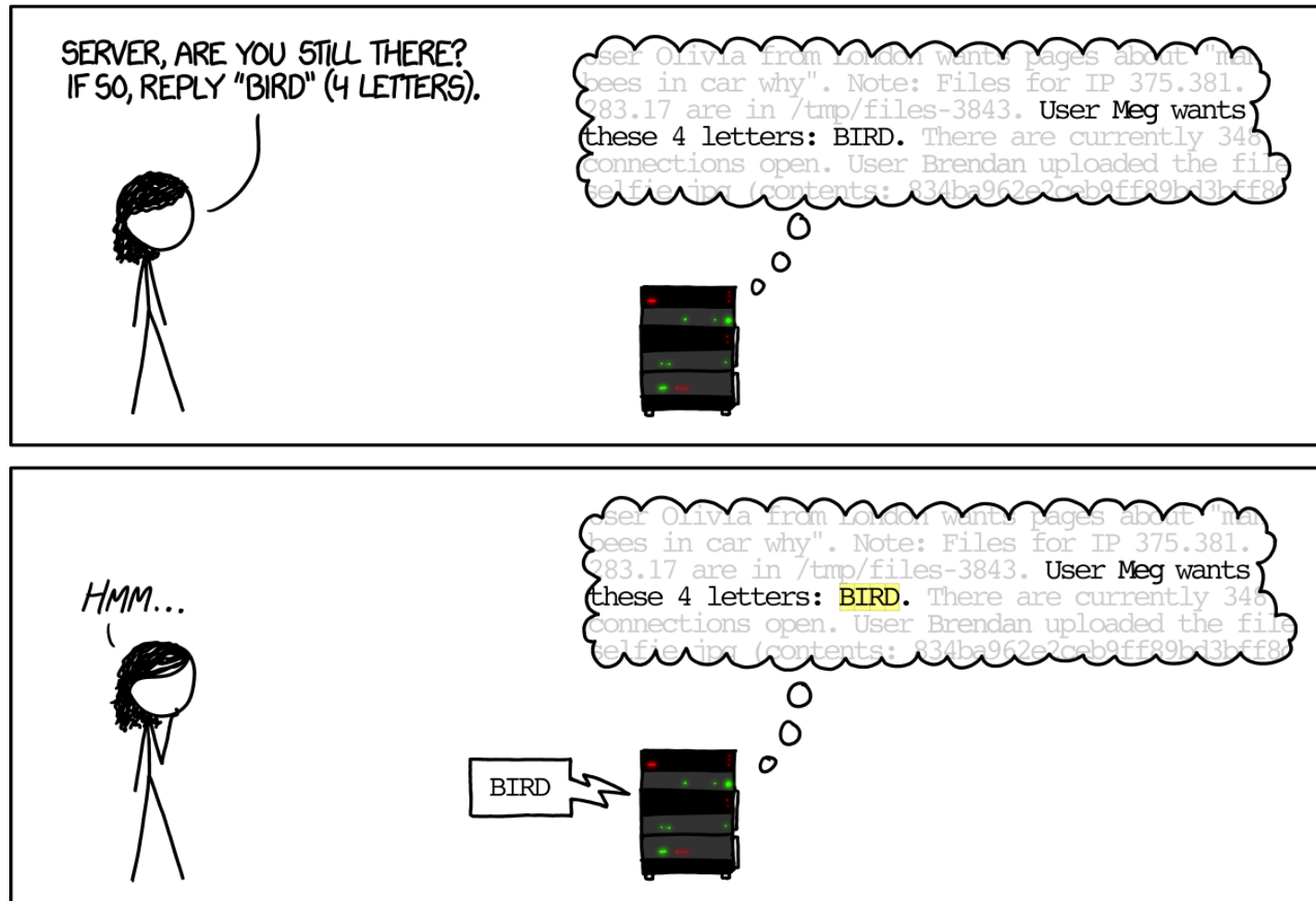
    • Now an MIT professor

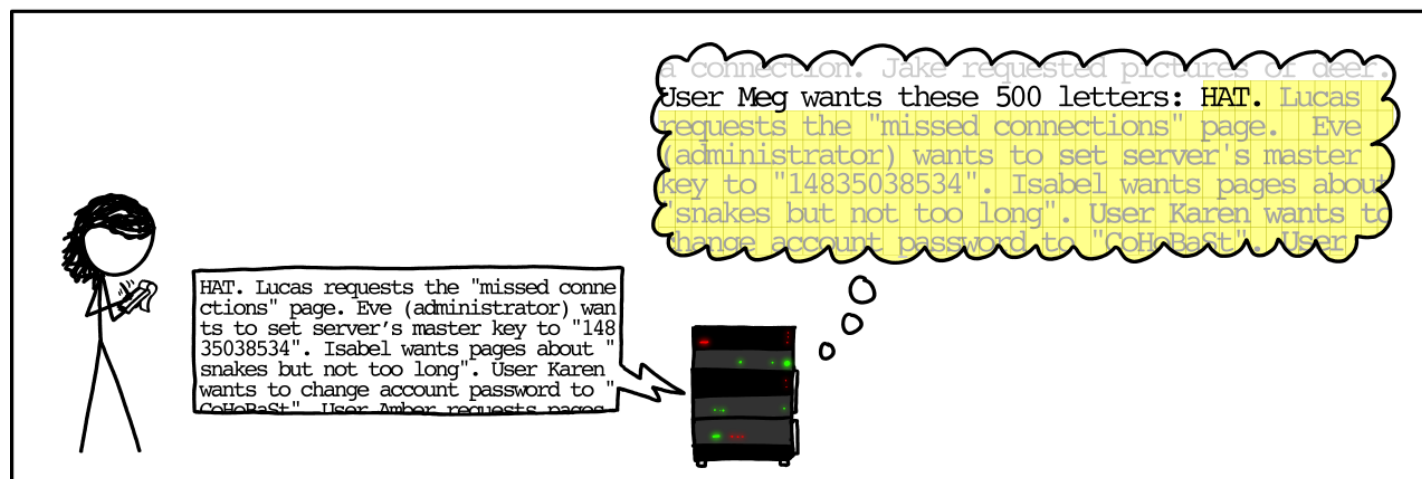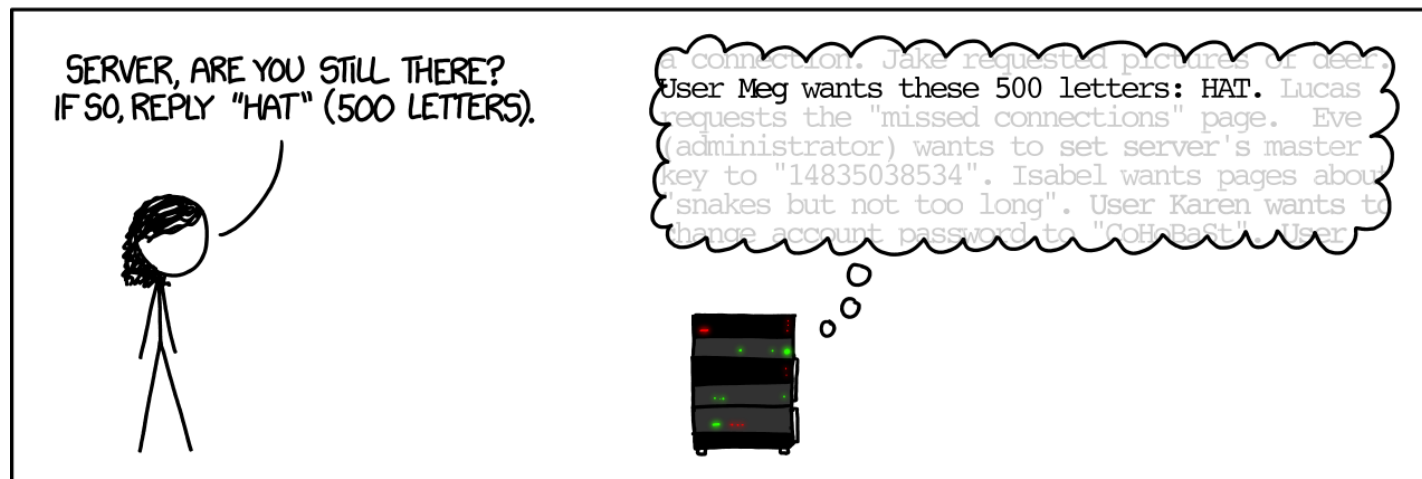The Morris Internet Worm source code

This disk contains the complete source code of the Morris Internet worm program. This tiny, 99-line program brought large pieces of the Internet to a standstill on November 2nd, 1988.

The worm was the first of many intrusive programs that use the Internet to spread.

Computer History Museum

Heartbleed (2014)

Heartbleed (2014)

Heartbleed (2014)

Heartbleed Details

❖ Buffer over-read in OpenSSL

- Open source security library
- Bug in a small range of versions

❖ "Heartbeat" packet: message & length

- Server echoes back data to match length
- Allowed attackers to read contents of memory

❖ ~17% of Internet affected

- *e.g.*, Github, Yahoo, Stack Overflow, Amazon Web Services
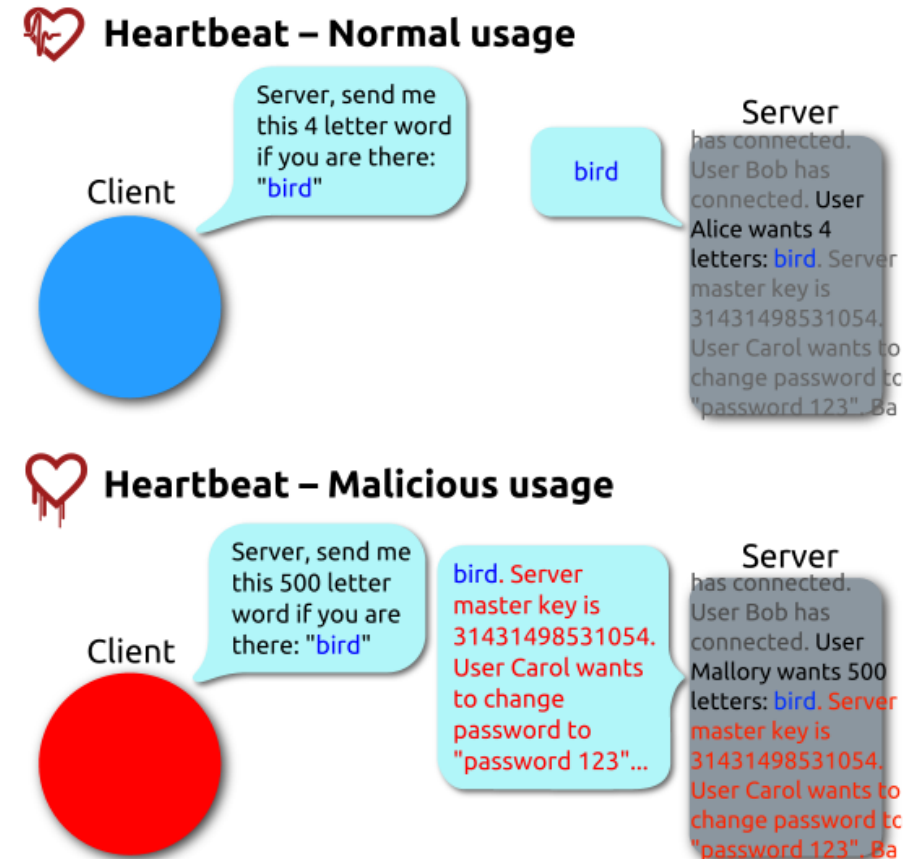
By FenixFeather - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=32276981

Hacking Cars (2010)

❖ UW CSE research demonstrated wirelessly hacking a car using buffer overflow

  ▪ http://www.autosec.org/pubs/cars-oakland2010.pdf

❖ Overwrote the onboard control system's code

  ▪ Disable brakes, unlock doors, turn engine on/off

Hacking DNA Sequencing Tech (2017)

❖ UW CSE project: Computer Security and Privacy in DNA Sequencing Ney et al. (2017): https://dnasec.cs.washington.edu/

  ▪ Potential for malicious code to be encoded in DNA!

  ▪ Attacker can gain control of DNA sequencing machine when malicious DNA is read
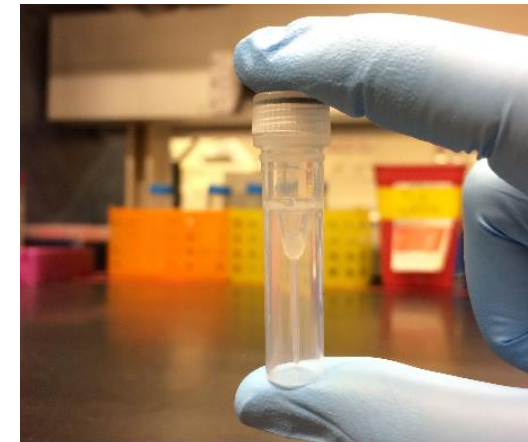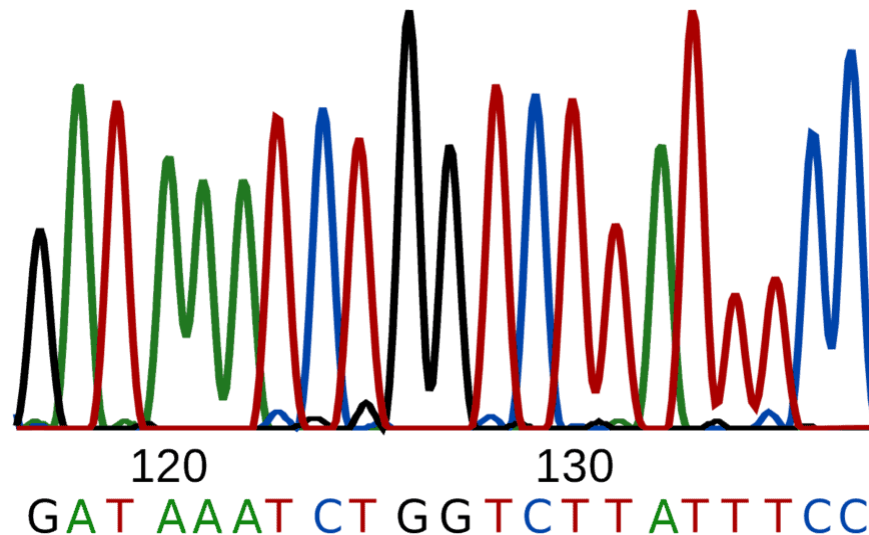

Figure 1: Our synthesized DNA exploit

Think this is cool?

- Take CSE 484 (Security)
  - Several different kinds of buffer overflow exploits
  - Many ways to counter them

- Nintendo fun!
  - Using glitches to rewrite code: https://www.youtube.com/watch?v=TqK-2jUQBUY
  - Flappy Bird in Mario: https://www.youtube.com/watch?v=hB6eY73sLV0

Discussion Questions

❖ Discuss the following question(s) in groups of 3-4 students

- I will call on a few groups afterwards so please be prepared to share out
- Be respectful of others' opinions and experiences

❖ Code injection attacks are a form of "*hacking*" that takes advantage of a *security vulnerability* (*e.g.,* buffer overflow). "Hacking" is now commonplace in society and media.

- What are some of the possible <u>consequences</u> & <u>objectives</u> of hacking (*i.e.,* to what ends might someone engage in hacking)?
- What are some reasons why vulnerable systems keep getting connected to the Internet?