

# Buffer Overflow

## CSE 351 Autumn 2023

### Guest Instructor:

Ellis Haker

### Teaching Assistants:

Afifah Kashif

Malak Zaki

Bhavik Soni

Naama Amiel

Cassandra Lam

Nayha Auradkar

Connie Chen

Nikolas McNamee

David Dai

Pedro Amarante

Dawit Hailu

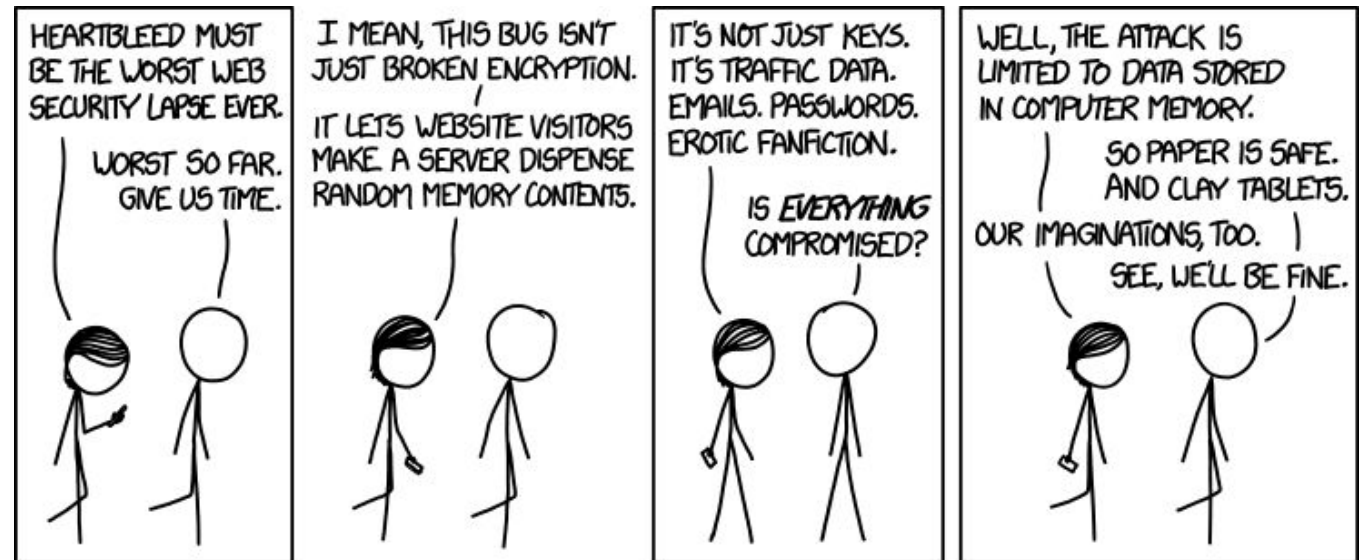
Renee Ruan

Eyoel Gebre

Simran Bagaria

Joshua Tan

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

- Lab 3 released today, due next Friday (11/10)
  - You will have everything you need by the end of this lecture
- Mid-Course Survey
  - Released on Sunday (11/5), closes Thursday (11/9)
- Midterm starts Thursday
  - Instructions will be posted on Ed Discussion
  - You are permitted to discuss high-level concepts and give hints, but are **not** allowed to solve the problems together
  - We will be available on Ed Discussion (private posts, please) and support hours to answer clarifying questions

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions. The text "Buffer Overflow" is overlaid in white on the center of the die.

# Buffer Overflow

# Lesson Summary (1/3)

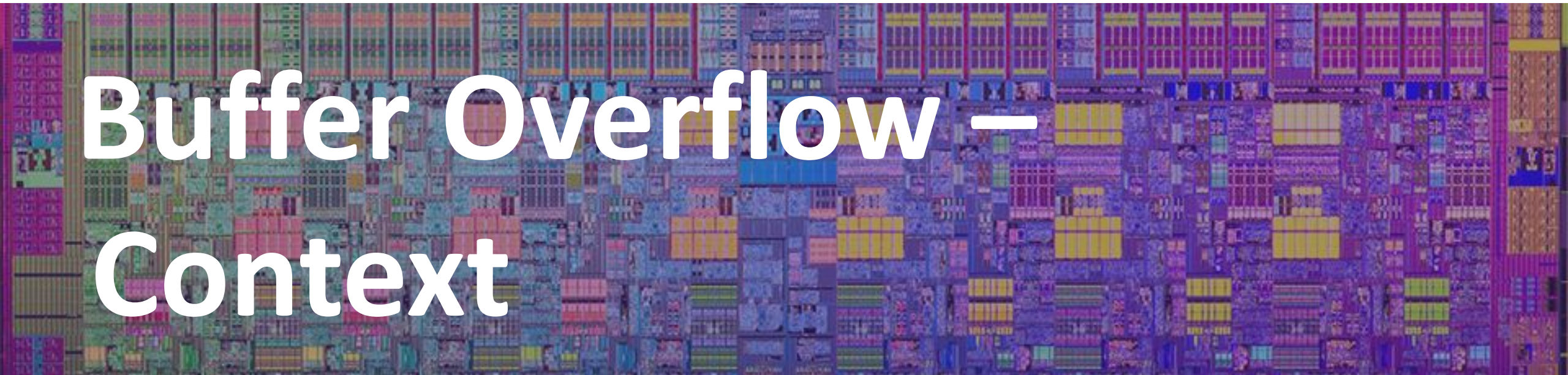
- Buffer overflow is a bug where more data is written to a buffer (array) than there is space for
  - Can be used to attack a system by overwriting important data
- Distressingly common in real programs
  - Most common technical source of security vulnerabilities
  - Programmers keep making the same mistakes 😞
  - Recent measures make these attacks much more difficult, but not impossible!
- Exploits based on buffer overflow
  - Stack smashing: Altering the execution of a program
  - Code injection: Run arbitrary code on target's computer

# Lesson Summary (2/3)

- Array bounds checking
  - In C, check manually or use safe library functions (eg: fgets)
  - Done automatically in most modern languages
- Stack canaries
  - Store a secret value in the stack before the return address, check that it wasn't overwritten before returning
- Non-executable memory regions
  - Prevent code from being executed on the stack
- Randomized stack offsets
  - Put a random amount of padding in memory before the stack

# Lesson Summary (3/3)

- Terminology:
  - Buffer, buffer overflow, stack smashing, code injection attack
- 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 at multiple levels (hardware, OS, software).
  - Perform stack smashing and code injection exploits.
- What lingering questions do you have from the lesson?

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions (purple, blue, yellow, green, red) representing different functional blocks and interconnects.

# Buffer Overflow – Context

# Example: the original Internet worm (1988)

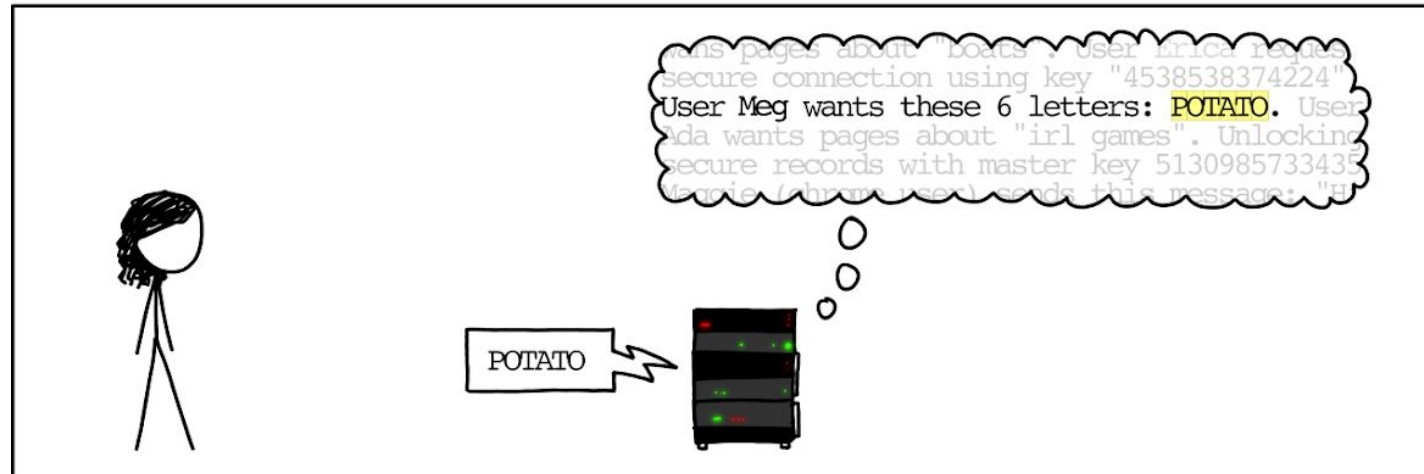
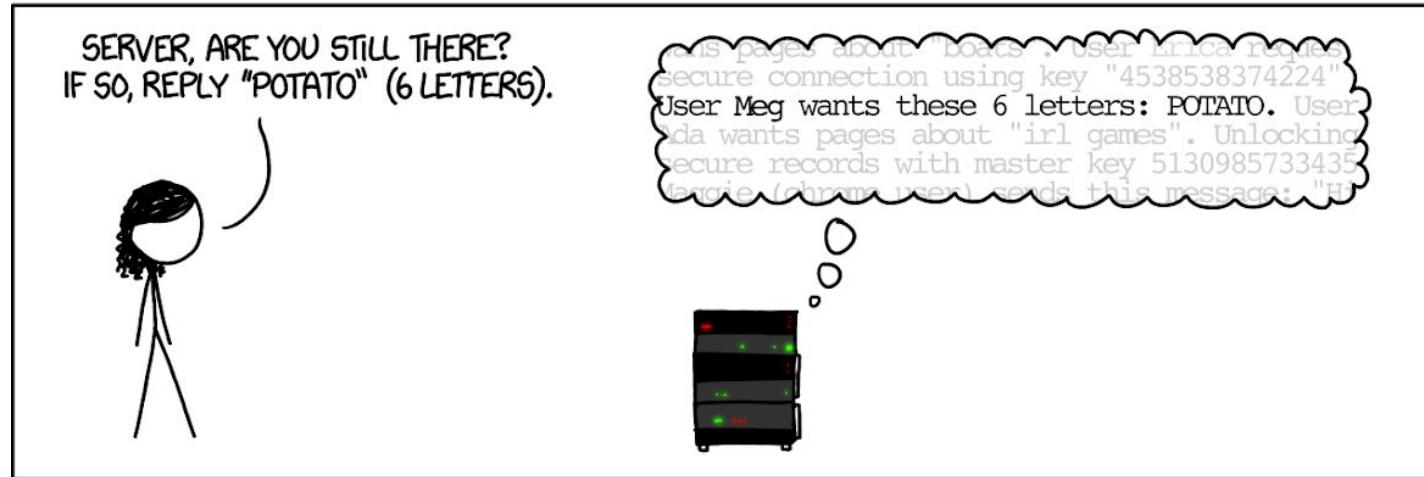
- Early versions of the finger server (`fingerd`) used `gets()` to read the argument sent by the client:
  - `finger droh@cs.cmu.edu`
- Worm attacked `fingerd` server with phony argument:
  - `finger "exploit-code padding new-return-addr"`
  - Exploit code: executed a root shell on the victim machine
- Scanned for other machines to attack
  - Invaded ~6000 computers in hours (10% of the Internet)
    - See June 1989 article in *Comm. of the ACM*
  - The author of the worm (Robert Morris) was prosecuted
    - First conviction under Computer Fraud and Abuse Act
    - Now an MIT professor



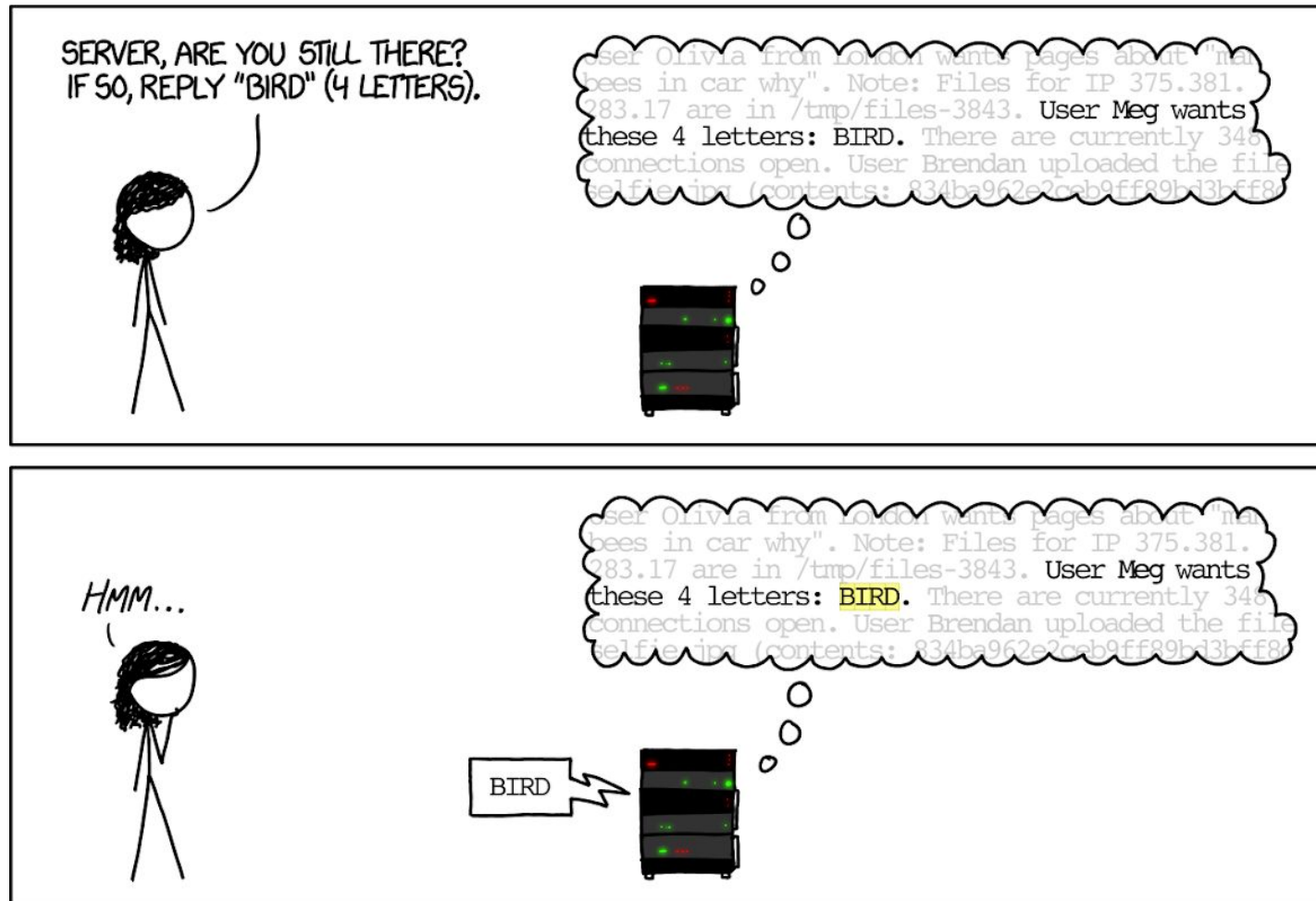


# Example: Heartbleed (2014)

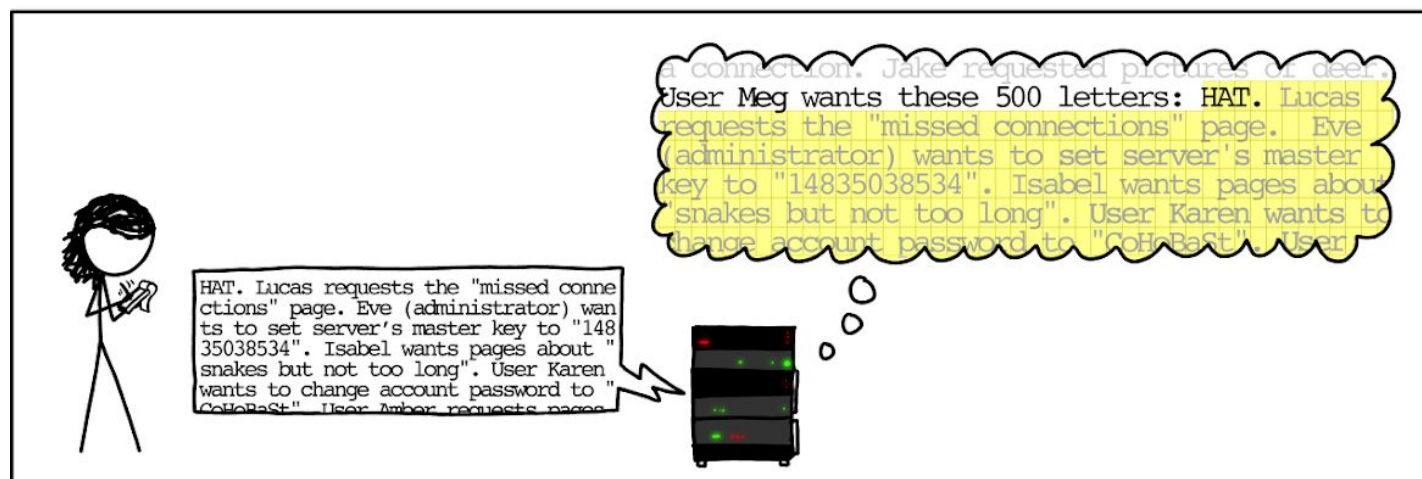
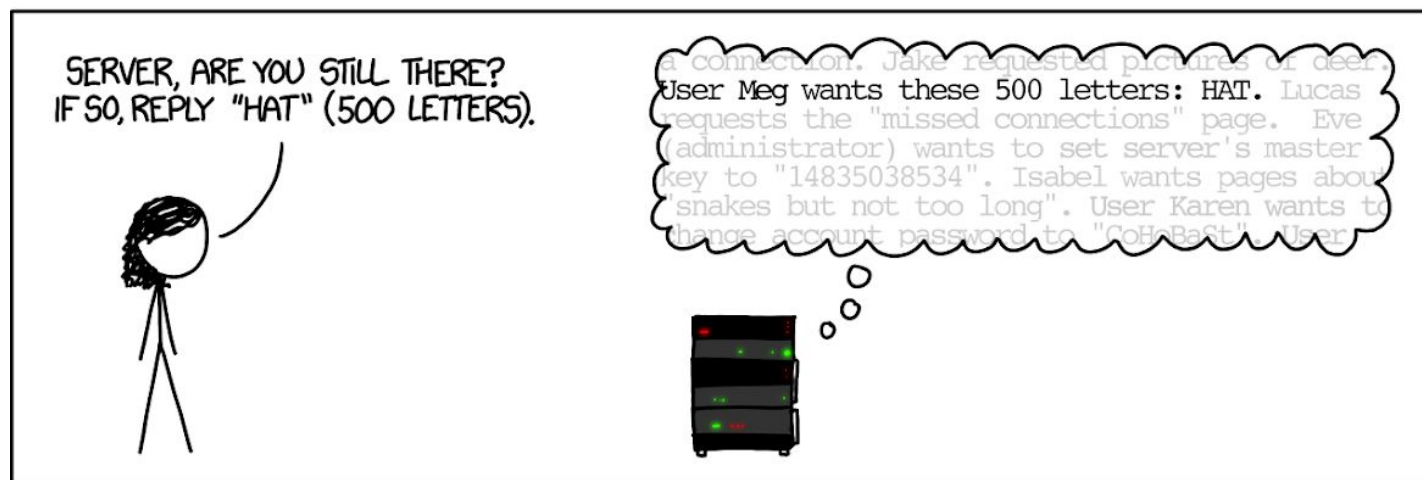
## HOW THE HEARTBLEED BUG WORKS:



# Example: Heartbleed (2014)

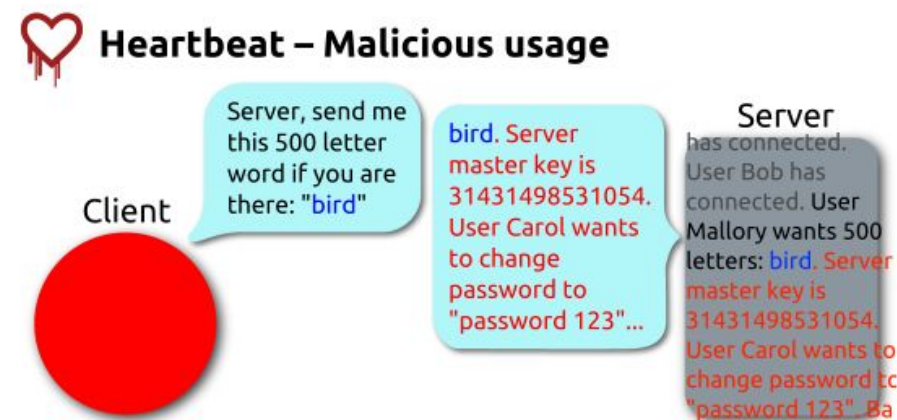
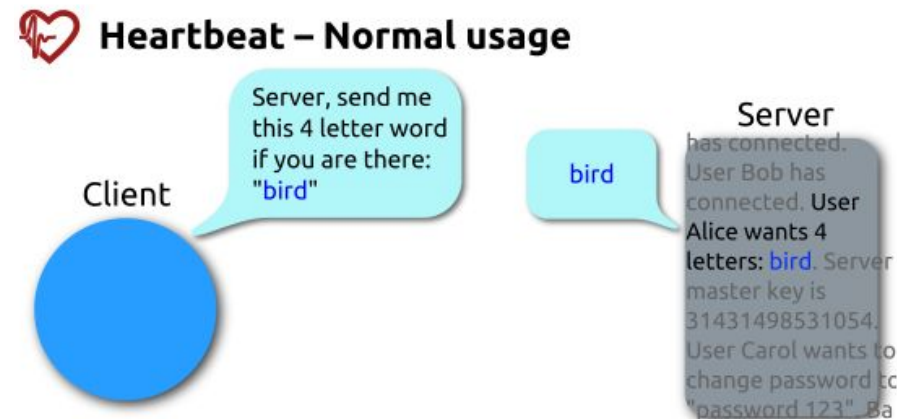


# Example: Heartbleed (2014)



# Heartbleed Details

- Buffer over-read in OpenSSL
  - Open source security library
  - Bug in a small range of versions
- “Heartbeat” packet
  - Specifies length of message
  - Server echoes it back
  - Library just “trusted” this length
  - Allowed attackers to read contents of memory anywhere they wanted
- Est. 17% of Internet affected
  - “Catastrophic”
  - Github, Yahoo, Stack Overflow, Amazon AWS, ...



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

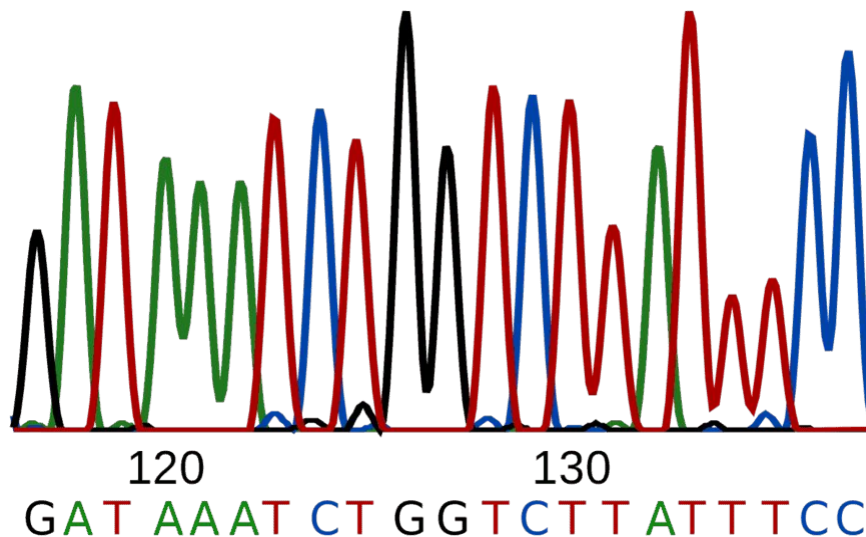


# Hacking DNA Sequencing Tech (2017)

## Computer Security and Privacy in DNA Sequencing

Paul G. Allen School of Computer Science & Engineering, University of Washington

- Potential for malicious code to be encoded in DNA!
- Attacker can gain control of DNA sequencing machine when malicious DNA is read
- Ney et al. (2017): <https://dnasec.cs.washington.edu/>



# 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
- If they're so well-known, why do buffer overflow attacks still happen?
  - Why do we still use unsafe languages like C?
  - What kinds of incentives dissuade tech companies from prioritizing security?

# Group Work Time

- During this time, you are encouraged to work on the following:
  - If desired, continue your discussion
  - Work on the lesson problems (solutions at the end of class)
  - Work on the homework problems
- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support



# Practice Question

- `buggy` 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: (0x00 00 7f ff ca fe f0 0d)

Previous stack frame			
00	00	00	00
00	40	05	d1
. . .			
			[0]

```
buggy:
  subq $0x40, %rsp
  ...
  leaq 16(%rsp), %rdi
  call gets
  ...
```

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

# Think this is cool?

- You'll love Lab 3 😊
  - Released Today, due next Friday (11/10)
  - Some parts *must* be run through GDB to disable certain security features
- 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>