CMSC 414: Computer and Network Security

Spring 2017

Sections 0101 and 0201

A. Udaya Shankar

What is computer & network security

- Normally, we are concerned with correctness
  - The software should achieve the desired behavior

- Security is a bit different:
  - The software should prevent undesired behavior
  - Key difference: adversary

What are “undesired” behaviors?

- Reveals info users wish to hide (confidentiality)
  - Corporate secrets
  - Private data; personally identifying information (PII)

- Modifies information or functionality (integrity)
  - Destroys records
  - Changes data in-flight (think “the telephone game”)
  - Installs unwanted software (spambot, spyware, etc.)

- Denies access to a service (availability)
  - Crashing a website for political reasons
  - Denial of service attack
  - Variant: *fairness*

This is a subset

Can we make this precise?

- What does preventing undesired behavior mean

- Correctness of a program P:
  - for any intended input, P produces desired output

- P is secure: for any input, P produces desired output
  - if input unintended, P produces null/error output
  - any input: any bit string applied at any input point

What are possible input points for P

- Input statements in P: `read()`, `scanf()`, . . .
- Compiler that translates P to an executable Q:
- Shell environment variables: PATH, . . .
- Libraries that are linked with Q
- OS that loads and runs Q
  - protect Q's address space from other processes
- Origin of compiler and OS, . . .

Why are attacks common?

- Because attacks derive from **design flaws** and/or **implementation bugs**

- But all software has bugs: so what?

- A *normal user* never sees most bugs
  - Post-deployment bugs are usually rare corner cases

- Too expensive to fix every bug
  - Only fix what's likely to affect normal users

Why are attacks common?

Attackers are not normal users

- Normal users avoid bugs/flaws
- Adversaries seek them out and try to *exploit* them

This extends beyond software:

Attacks are possible even with perfect software

Why are attacks common?

Because it's **profitable**

And because a system is
*only as secure as its **weakest link***

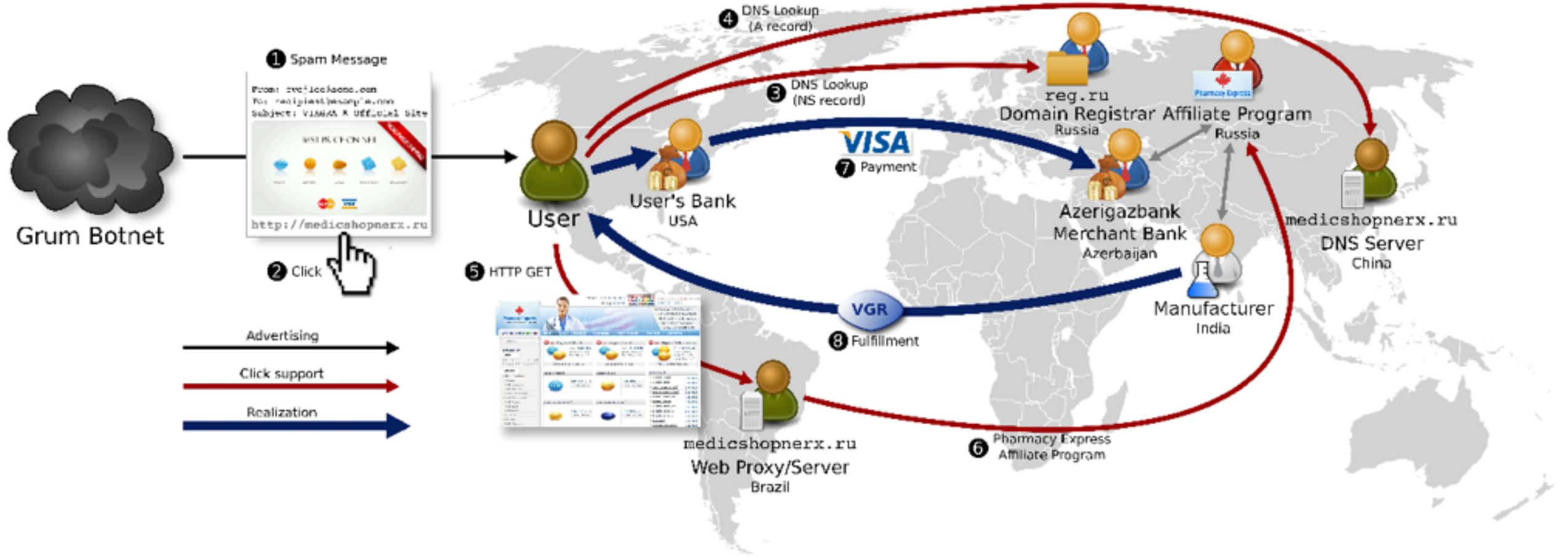
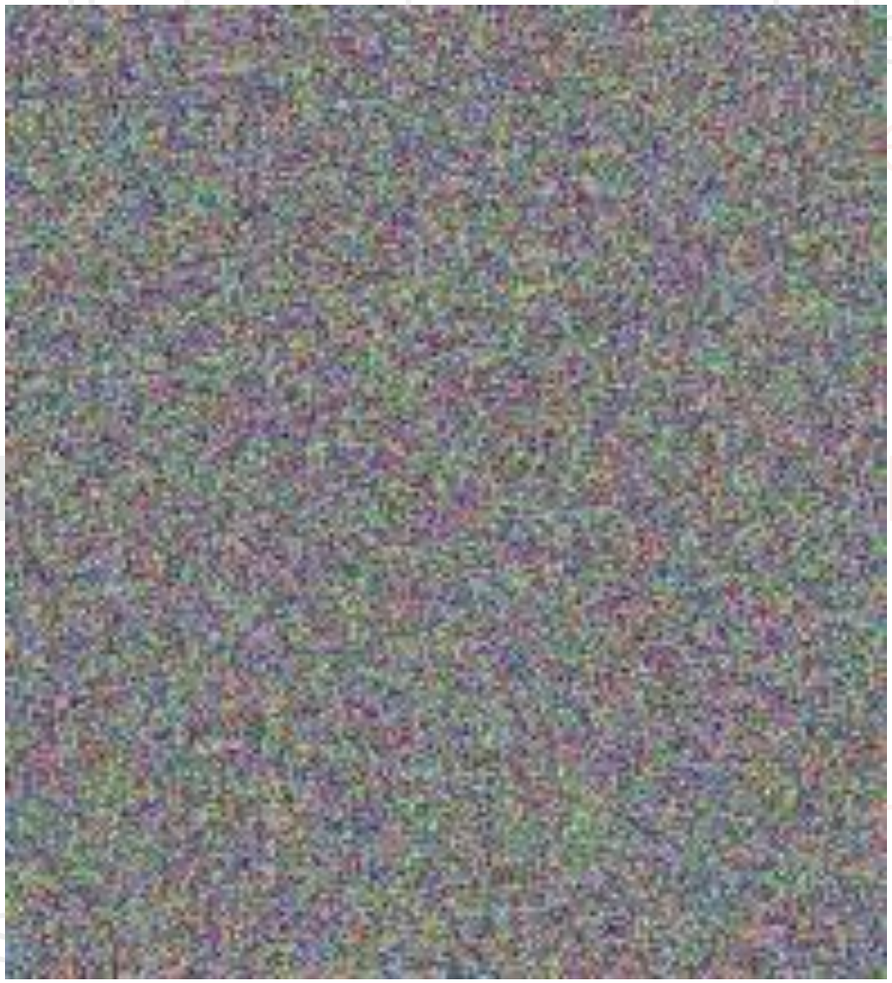


Figure 1: Infrastructure involved in a single URL's value chain, including advertisement, click support and realization steps.

Crypto

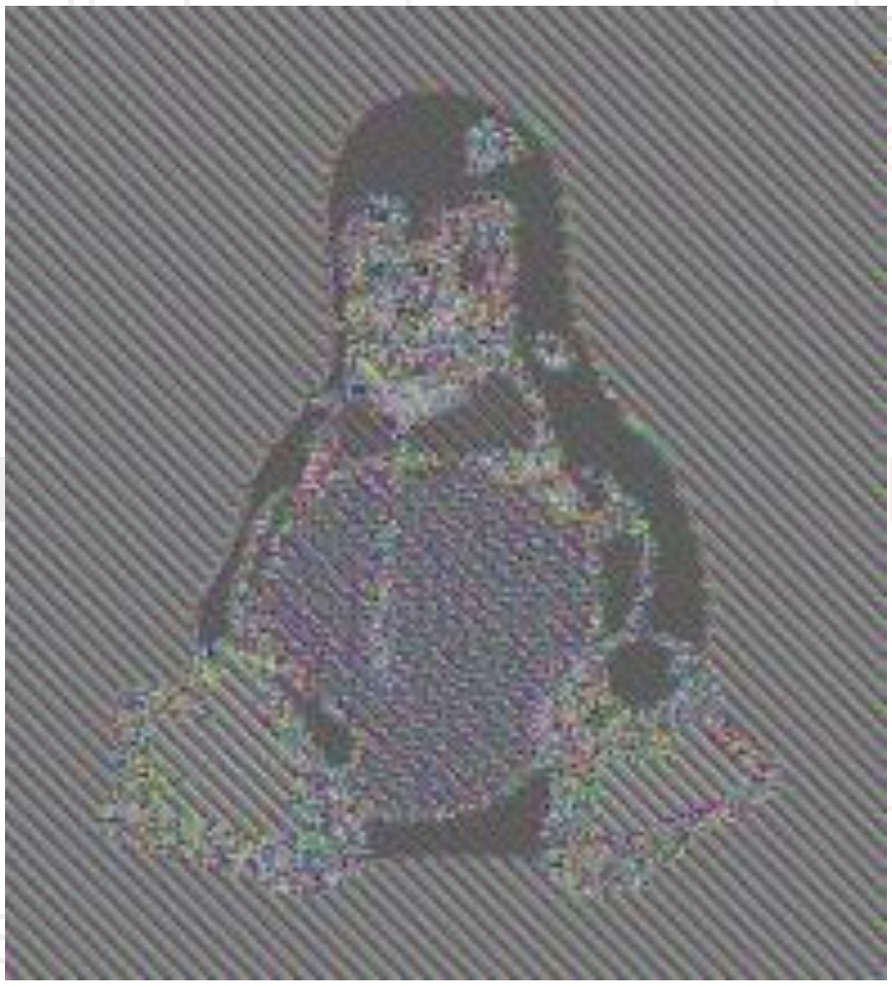
- Fundamental to detecting invalid input strings
  - invalid password, invalid executable, . . .
- Good crypto is based on sophisticated mathematics
  - Don’t make up your own crypto
- Crypto by itself cannot overcome all attacks
  - if Q’s address space is exposed, attacker can get key

Widespread misuse of crypto



This is an encrypted image

Widespread misuse of crypto



This is an encrypted image

50% of Android apps that use crypto encrypt in this manner

In order to achieve security, we must:

Be able to eliminate bugs and design flaws
and/or make them harder to exploit.

Be able to think like attackers.

Develop a foundation for deeply understanding
the systems we use and build.

Software **Hardware** **Protocols**

Users **Law** **Economics**

Administrative

- Syllabus, resources, office hours, and all this on class page: http://www.cs.umd.edu/class/spring2017/cmsc414
- People
  - Me: A. Udaya Shankar (shankar@cs.umd.edu)
  - TAs: Nishant Rodrigues
    Jacob Hammontree
    Ashton Webster
    Stephan Kostreski
- Piazza
- Grades:
  - Projects P1-P4: 50%
  - 2 midterms: 15% each
  - Final: 20%

Read the syllabus

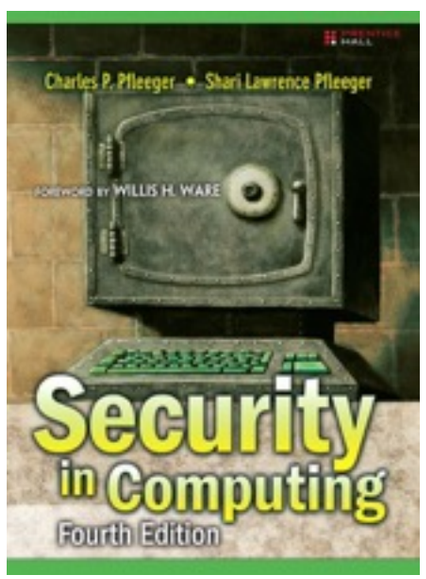
- Late policy
- Good-faith effort requirement
- Excused absences
- Academic integrity

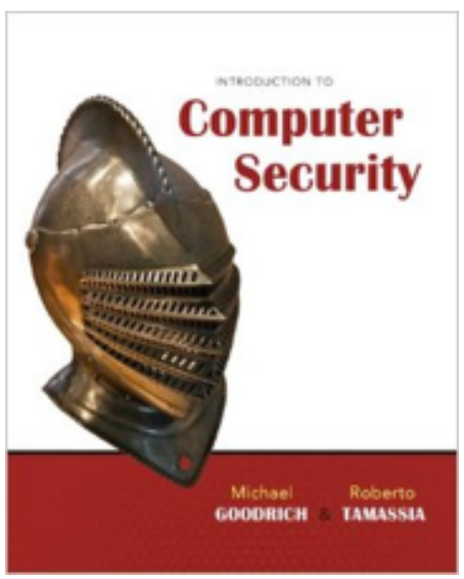
Administrative

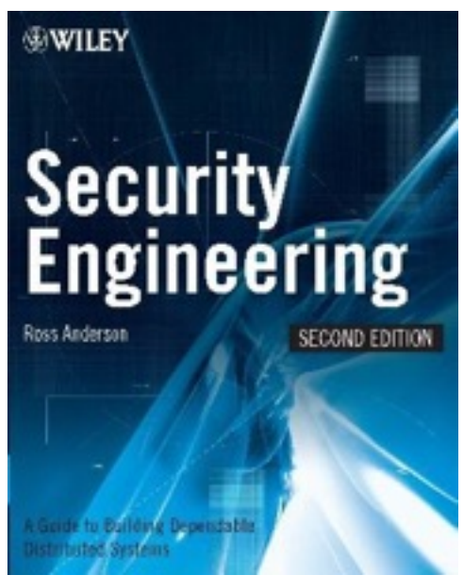
Textbooks

- None required
  - Mostly in-class and papers posted on website

- Recommended texts, if you are so inclined
  - "Security in Computing", Pfleeger & Pfleger
  - "Introduction to Computer Security", Goodrich & Tamassia
  - "Security Engineering", Ross Anderson
    - Free online: http://www.cl.cam.ac.uk/~rja14/book.html







Administrative

Outside reading

- The best way to learn is to reinforce
- *Lots* of security resources (something is always breaking).
  - Krebs on security
  - Bruce Schneier's blog
  - reddit.com/r/netsec
  - Any other favorites? Let us know on Piazza

What’s in this course?

What's in this course?

Software Security

How do we build software that is secure?

Memory safety

Malware

Web security

Static analysis

Design principles

What's in this course?

Software Security

Crypto

What it is, and how to use it responsibly

A black-box approach to crypto

Designing protocols that *use* crypto

Authentication: proving who you are

Anonymity: hiding who you are

What's in this course?

Software Security

Crypto

Network Security

Attacks on TCP & DNS

Botnets

Underground spam economies

How to build secure networked systems.

What's in this course?

Software Security

How do we build software that is secure?

Crypto

What it is, and how to use it responsibly.

Network Security

How to build secure networked systems.

Attacks and defenses across all of these

Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate "On the Cusp" page.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)

Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate "On the Cusp" page.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)

Ethics and legality

- You will be learning about (and implementing and *launching*) attacks, many of which are in active use today.

- ***This is not an invitation to use them without the explicit written consent of all parties involved***

- If you want to try something out, then *let me know* and I will try to help create a safe environment

- This is not just a question of ethics; to do otherwise would risk violating UMD policies and MD/USA laws

Prerequisite knowledge

- You should be reasonably proficient in C and Unix
- You should also be creative and resourceful (those who try to attack your systems will be!)
- Otherwise, this course **won't** require any prior knowledge in networking or crypto

Trusting Trust

Is anything really “secure”?

Is anything really “secure”?

- Security requires context
  - What is the ***threat model***? What can the attacker do?
  - What are the ***assets*** you seek to protect?
  - Whom and what do you ***trust***?

Is anything really “secure”?

- Security requires context
  - What is the ***threat model***? What can the attacker do?
  - What are the ***assets*** you seek to protect?
  - Whom and what do you ***trust***?

- “Trust no one!”
  - That’s the spirit!
  - But how did you compile your code again?
  - Who built your OS? Your hardware?…

Is anything really “secure”?

- Security requires context
  - What is the ***threat model***? What can the attacker do?
  - What are the ***assets*** you seek to protect?
  - Whom and what do you ***trust***?

- “Trust no one!”
  - That’s the spirit!
  - But how did you compile your code again?
  - Who built your OS? Your hardware?…

Required reading

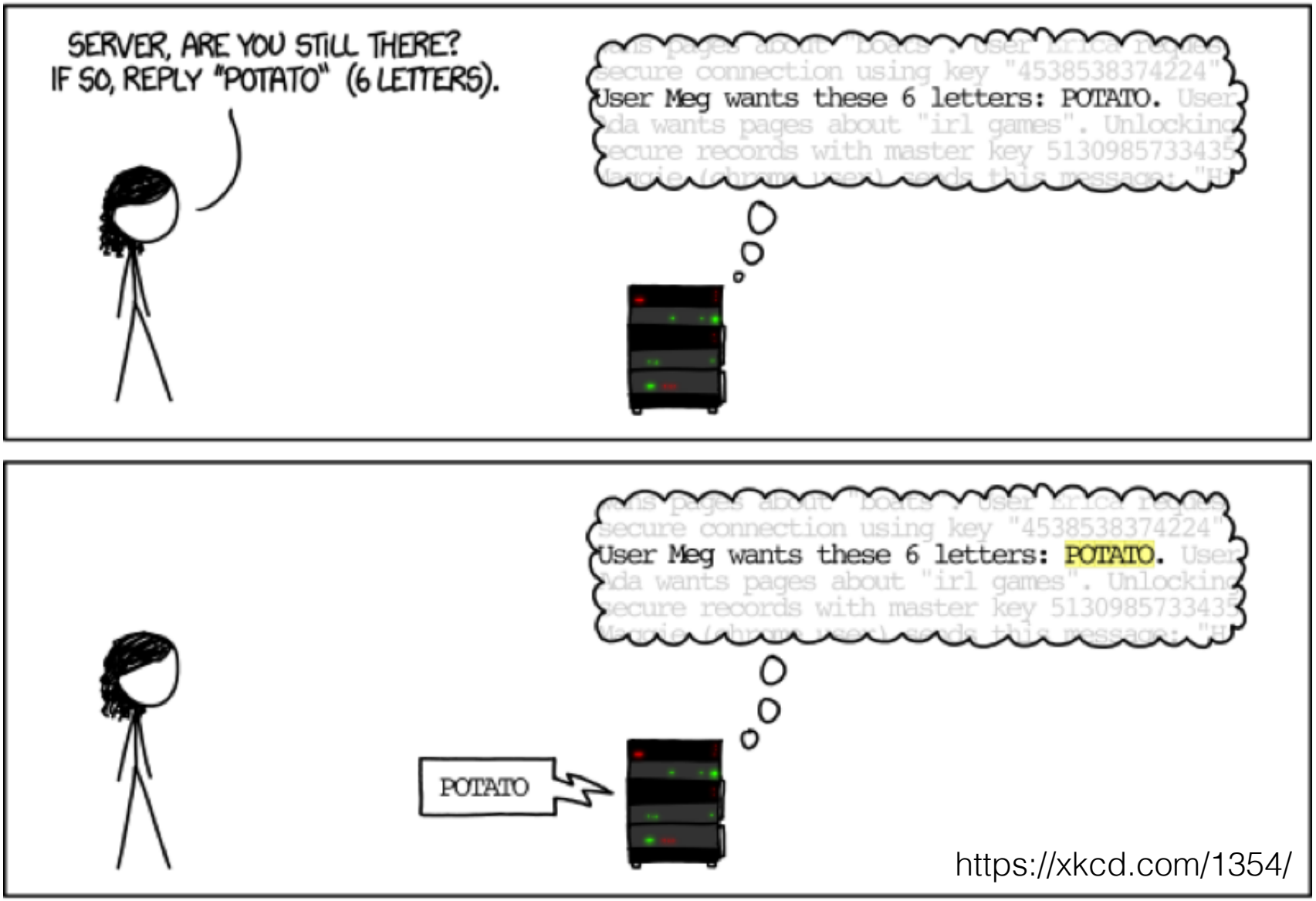
“Reflections on Trusting Trust”
Ken Thompson

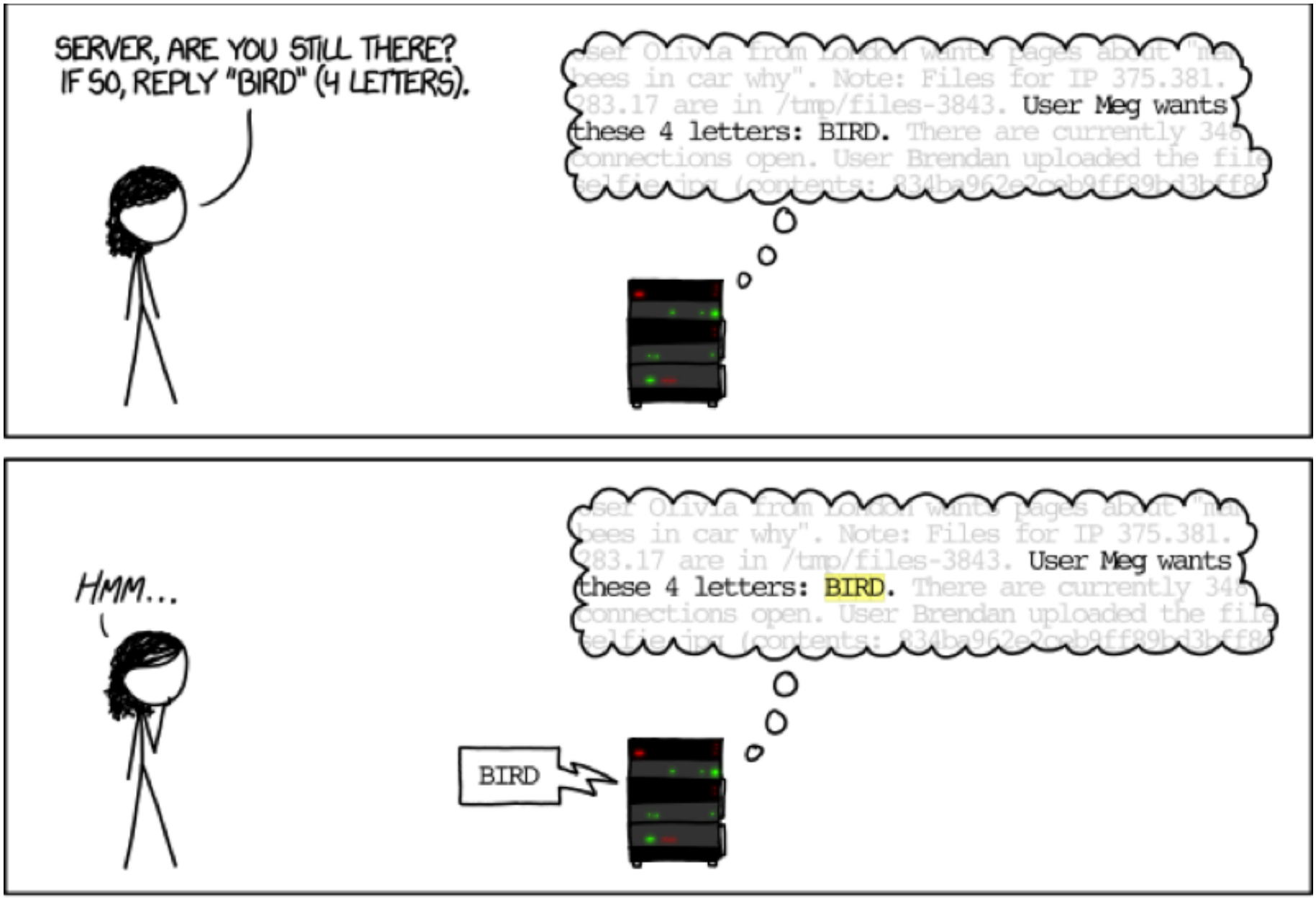
Case study: Heartbleed

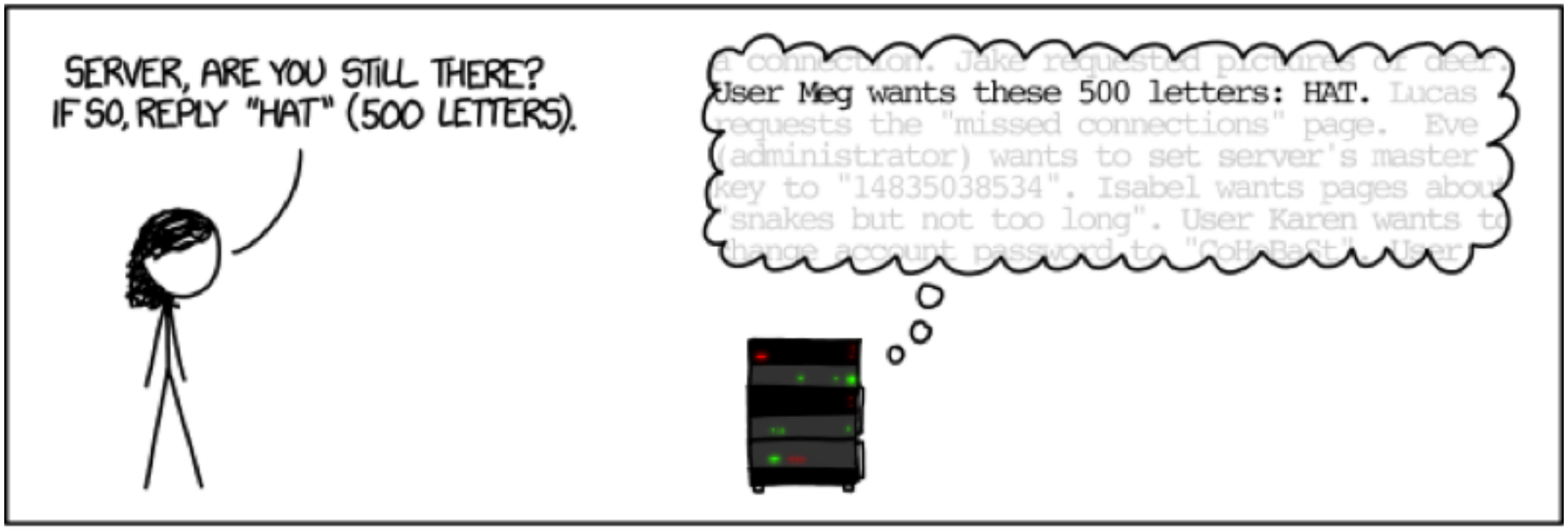
- SSL is the main protocol for secure (encrypted) online communication

- Heartbleed was a vulnerability in the most popular SSL server

HOW THE HEARTBLEED BUG WORKS:




SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Meg wants these 4 letters: BIRD.
HMM...
BIRD


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT.

Case study: Heartbleed

- SSL is the main protocol for secure (encrypted) online communication
- Malformed packet allows you to see server memory
  - Passwords, keys, emails, visitor logs …..
- Fix: Don't let the user tell you how much data to send back!
  - This is a *design* flaw

RSA breach, 2011

1. Flash exploit: When run by vulnerable Flash player version, allows arbitrary code exec.
2. Excel embed: Runs automatically when spreadsheet is opened.
3. Spear phishing: Spreadsheet attached to email claiming to be from trusted party, about relevant content
   - Any “From” address can be forged

Next time

We will begin our 1st section:

Software Security

By investigating

Buffer overflows

and other memory safety vulnerabilities

To prepare: you may want to brush up on your C

Particularly if this seems foreign to you:

```
char buf[32];
unsigned *ptr = (unsigned*) (buf + 12);
*ptr += 0x1a;
```