# Network Interception in Rust
Building a MITM Tool from Scratch

Introduction

Agenda

- **Introduction**
- **0 - Training Goals**
- **0bis - pnet library**
- **1 - ARP Spoofing**
- **2 - Intercept IPv4 packets**
- **3 - TCP handshake**
- **4 - Intercept HTTP requests**
- **5 - Handle TLS handshake**
- **Bonus**
- **Real-life scenario**
- **Appendices**

Introduction

Who are we ?

Thomas HALIPRÉ

Corentin LIAUD

- Working for Synacktiv

- French offensive security company

- ~200 ninjas: pentest, reverse engineering, development, DFIR

- 6 locations in France: Paris, Rennes, Lyon, Toulouse, Bordeaux & Lille
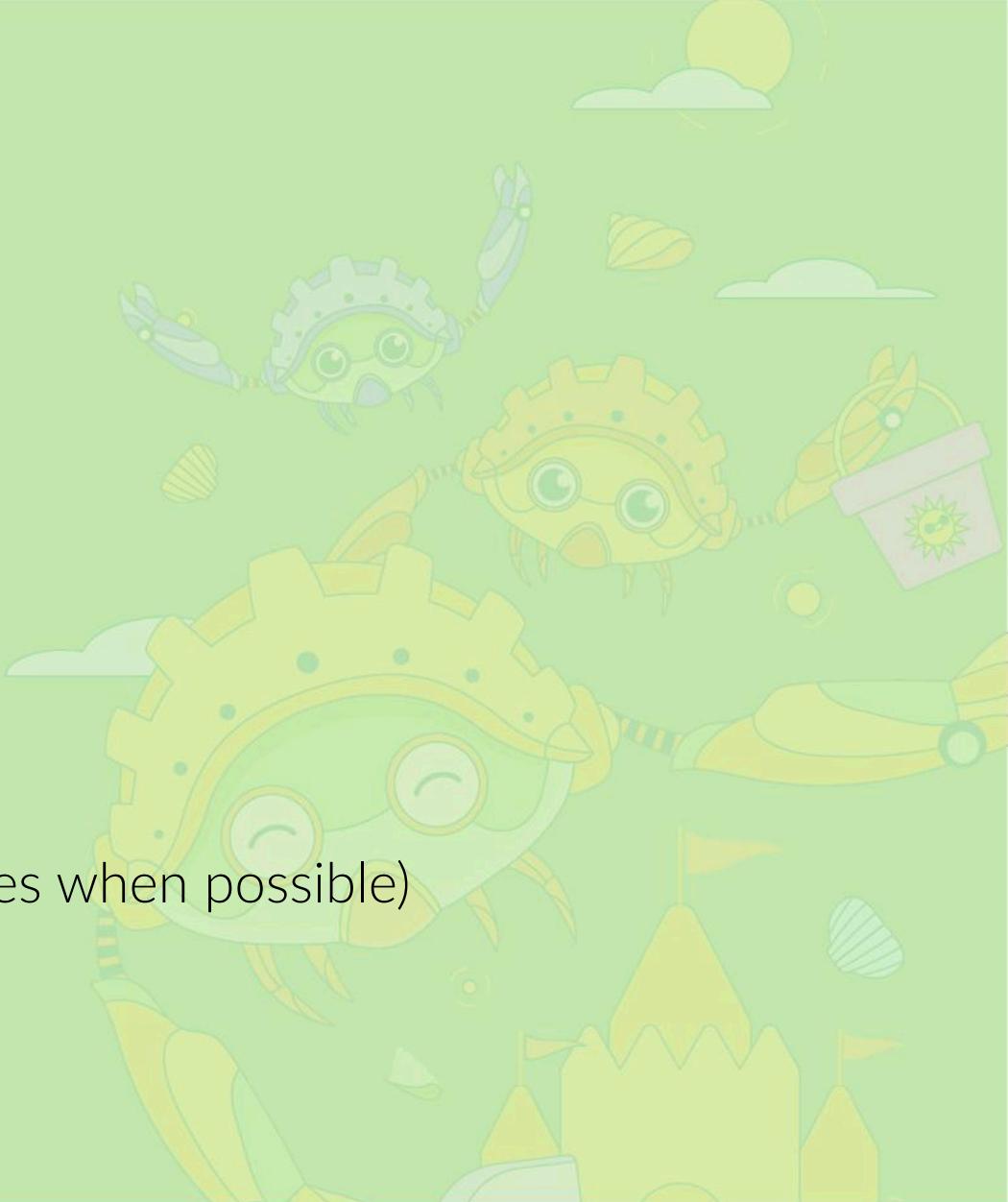
Introduction

What are we going to talk about?

- Rust ! 🦀

- Low level packet interception

- Network protocol overviews

- Offensive security (and applicable countermeasures when possible)

Introduction

What aren't we going to talk about?

- Rust basics

- IPv6
    - Already have much to do with IPv4!

- Async

- Performance
    - Code skeleton is fully monothreaded

Introduction

Disclaimer(s)

- For educational purposes only
- Don't run this code on networks you aren't allowed to :)
  - ~~TSH Guest~~

Introduction

Discord server

https://discord.gg/Y2ae88HneZ

Introduction

(Expected) schedule

| SETUP + ARP	IP	Coffee	TCP 1/2	Lunch	TCP 2/2	HTTP	Coffee	Physical lab	TLS	Win !
09:15	10:30		11:30		14:30	15:00		16:30	optional	
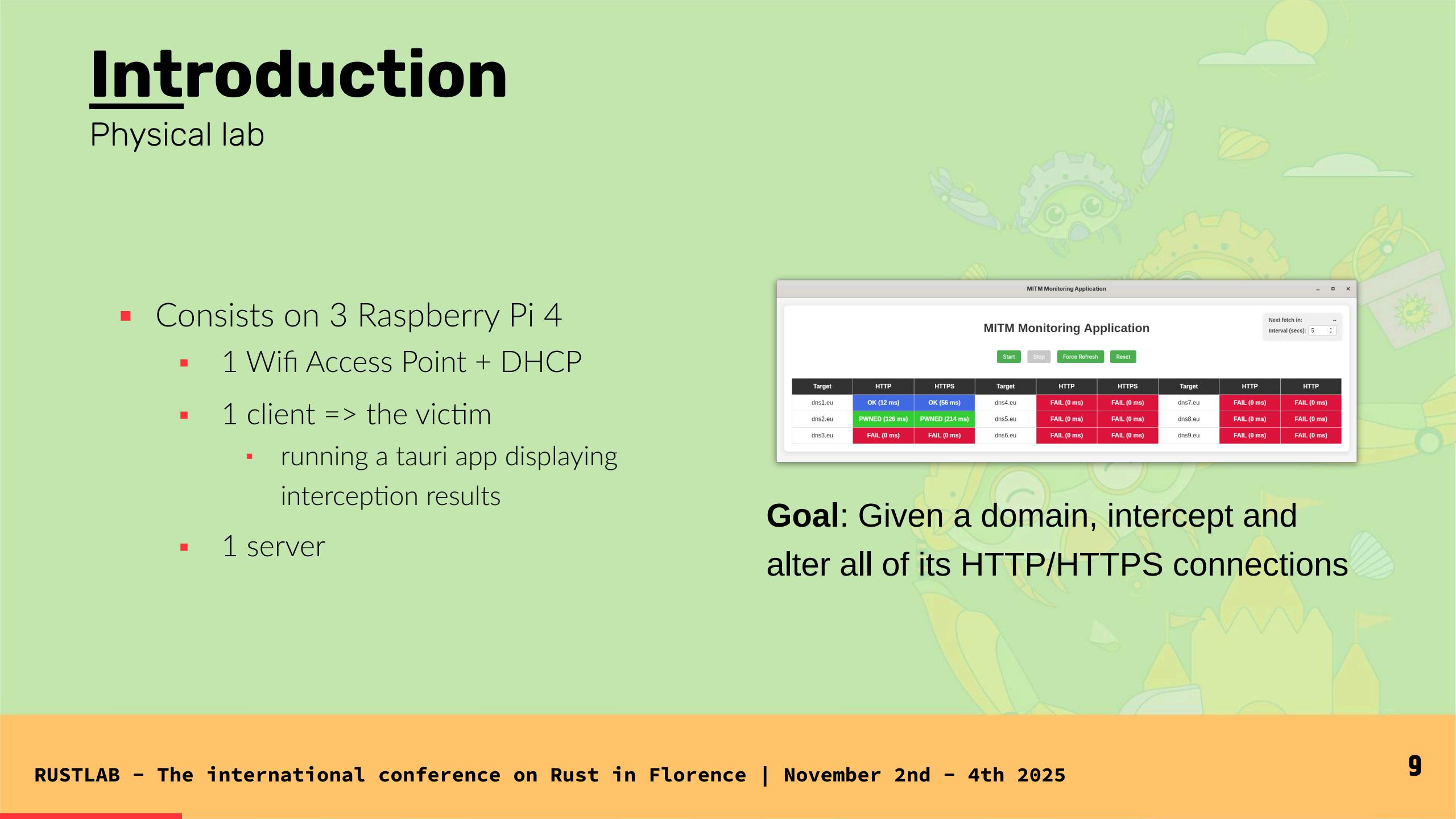
Introduction

Physical lab

- Consists on 3 Raspberry Pi 4
  - 1 Wifi Access Point + DHCP
  - 1 client => the victim
    - running a tauri app displaying interception results
  - 1 server
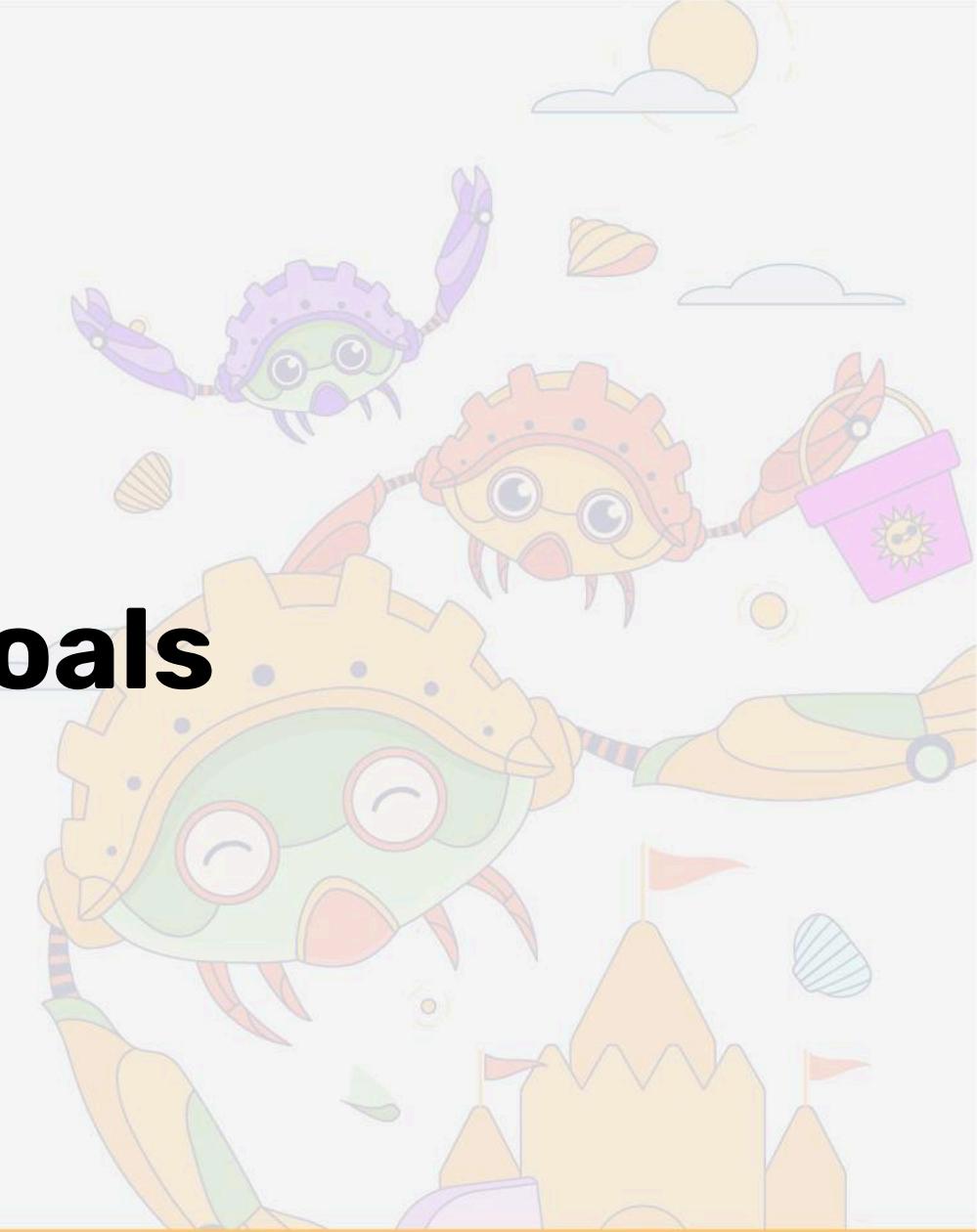
Goal: Given a domain, intercept and alter all of its HTTP/HTTPS connections

0 - Training Goals

0 - Training Goals
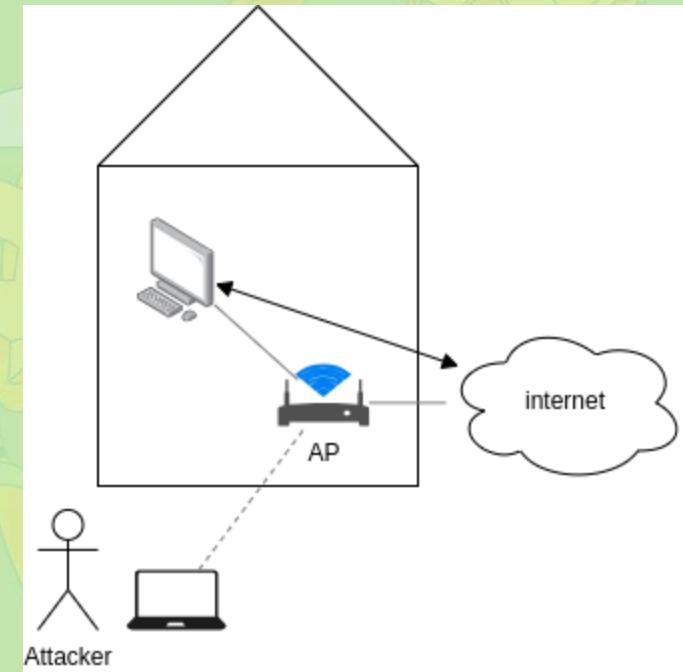
Initial Situation

- Two machines communicating on the same network (Ethernet or WIFI)
- They exchange data and assume their connection is private

As the attacker, we assume that we are connected to the same network.

Example

Your network at home!
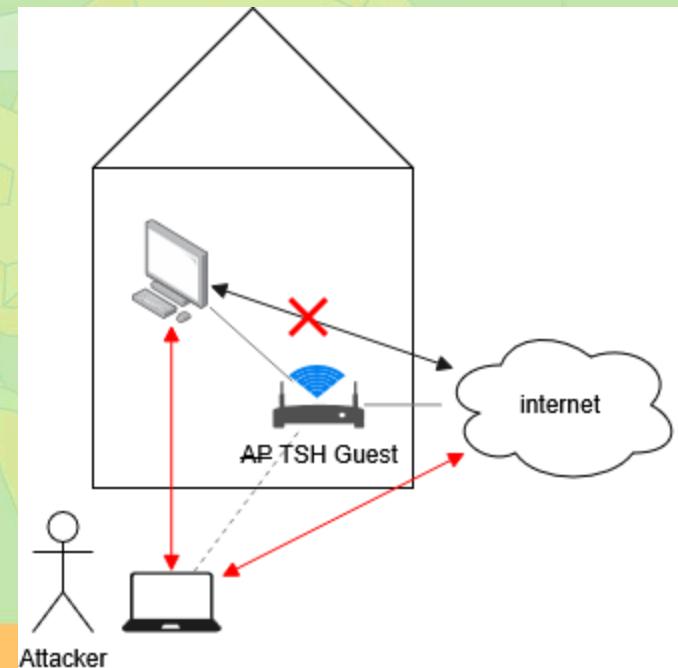
0 - Training Goals

Objectives

Example

Your "hacked" network at ~~home~~ RustLab 2025!

- **Read** the traffic between the two machines
  - Dump sensitive data (passwords, personal data...)
  - List requested websites
- **Modify** the data they are exchanging

<u>0</u> - Training Goals
Steps

1. **ARP Spoofing**: Achieving the Man-in-the-Middle position.
2. **MitM on IP**: Intercepting network level packets.
3. **MitM on TCP**: Intercepting transport level packets.
4. **MitM on HTTP**: Reading and modifying unencrypted web traffic.
5. **MitM on TLS (Optional)**: Attempting to intercept encrypted HTTPS traffic.

7. Application (HTTP, FTP, SMTP, DNS)

6. Presentation (SSL/TLS)

5. Session (NetBIOS, RPC)

4. Transport (TCP, UDP)

3. Network (IP, ICMP)

2. Data Link (Ethernet)

1. Physical (Cables, signals, fibers)

Obis - pnet library

Cross-platform, low level networking in Rust[1]

- Sending, receive and craft packets

- Protocol parsing (Ethernet, IPv4, TCP, ARP, ...)

- Memory safety & idiomatic Rust API

- Supported platforms
    - Linux using raw `AF_PACKET` sockets
    - macOS/BSD using BPF (**B**erkeley **P**acket **F**ilter)
    - Windows using `winpcap` [2] or `npcap` [3] (its successor)

[1] pnet library - https://github.com/libpnet/libpnet
[2] winpcap library - https://www.winpcap.org
[3] npcap library - https://npcap.com

Obis - pnet library

Two different ways to manipulate packets:

> ℹ️ **Documentation**
>
> `pnet` documentation is a must-have for this training

Using "structures"

```rust
let arp = Arp {/* ARP params */};

let mut arp_data = vec![0u8; ArpPacket::packet_size(&arp)];

let mut arp_pkt = MutableArpPacket::new(&mut arp_data)
    .expect("cannot build arp packet");
arp_pkt.populate(&arp);

let ethernet = Ethernet {
    /* Ethernet params */
    payload: arp_data,
};

let mut eth_data = vec![0u8; EthernetPacket::packet_size(&ethernet)];

let mut eth_pkt = MutableEthernetPacket::new(&mut eth_data)
    .expect("cannot build ethernet packet");
eth_pkt.populate(&ethernet);
```

Using builder pattern

```rust
let mut arp_data = vec![0u8; ArpPacket::minimum_packet_size()];

let mut arp_pkt = MutableArpPacket::new(&mut arp_data)
    .expect("cannot build arp packet");
arp_pkt.set_hardware_type(ArpHardwareTypes::Ethernet);
arp_pkt.set_operation(ArpOperations::Reply);
/* keep building... */

let mut eth_data = vec![0u8;
    EthernetPacket::minimum_packet_size() + arp_data.len()];

let mut eth_pkt =
    MutableEthernetPacket::new(&mut eth_data)
        .expect("cannot build ethernet packet");
/* build ethernet packet... */
eth_pkt.set_payload(&mut arp_data);
```
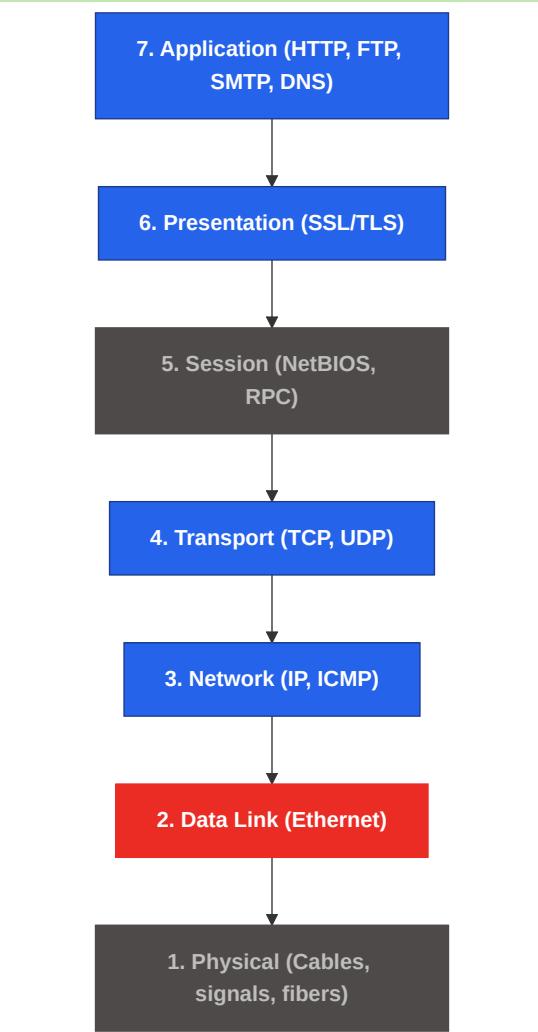
1 - ARP Spoofing
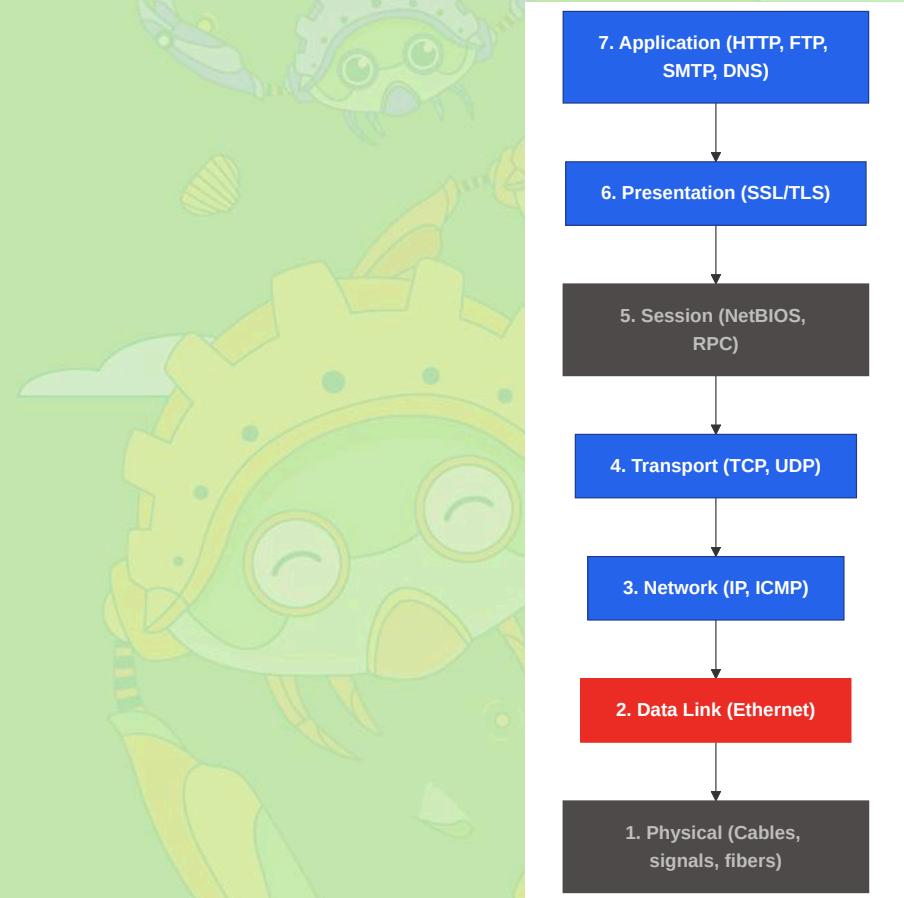
1 - ARP Spoofing
Network stack

1 - ARP Spoofing

Ethernet protocol [1]
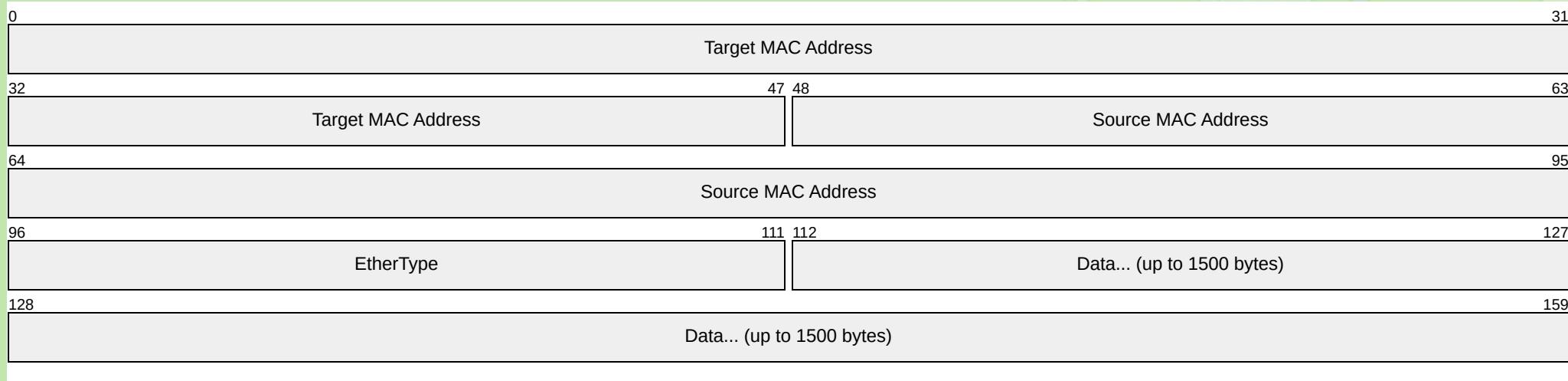
- Point-to-point communication

- Based on MAC addresses
  - e.g. `e2:2b:e9:ee:ff:55`

- On LANs (**L**ocal **A**rea **N**etworks) only

OSI Layers
7. Application (HTTP, FTP, SMTP, DNS)
6. Presentation (SSL/TLS)
5. Session (NetBIOS, RPC)
4. Transport (TCP, UDP)
3. Network (IP, ICMP)
2. Data Link (Ethernet)
1. Physical (Cables, signals, fibers)

[1] 802.3-2018 - IEEE Standard for Ethernet https://ieeexplore.ieee.org/document/8457469

1 - ARP Spoofing
Ethernet II frame format

0	31
Target MAC Address	

32	47	48	63
Target MAC Address		Source MAC Address	

64	95
Source MAC Address	

96	111	112	127
EtherType		Data... (up to 1500 bytes)	

128	159
Data... (up to 1500 bytes)	

Usual EtherTypes:

- IPv4 (*0x0800*)

- ARP (*0x0806*)

- Wake-On-Lan (*0x0842*)
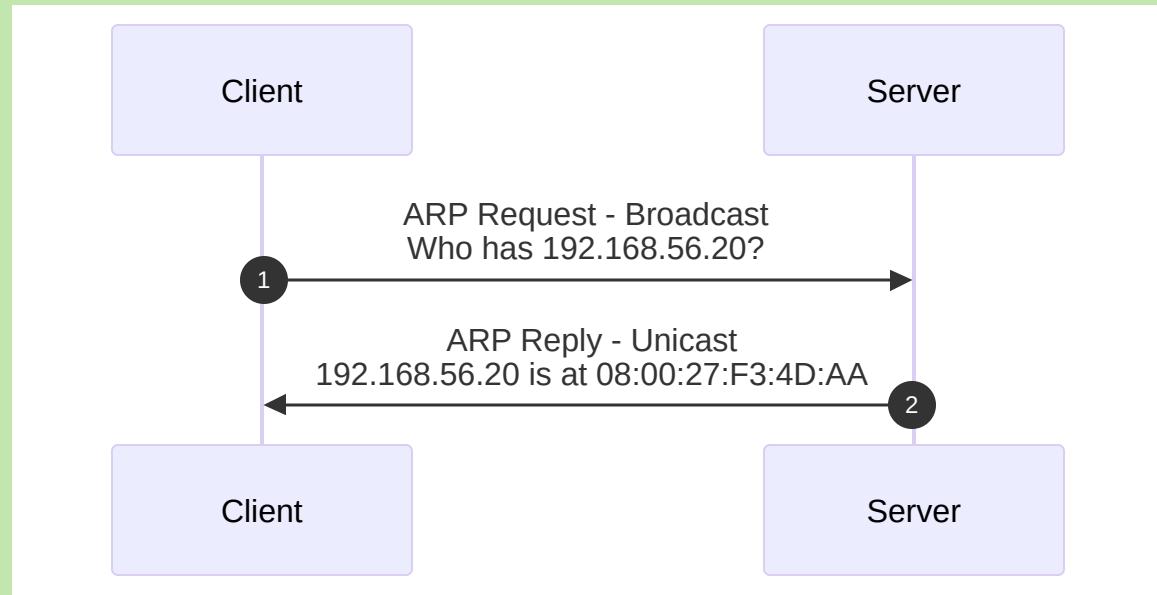
1 - ARP Spoofing
ARP protocol [1]

- **A**ddress **R**esolution **P**rotocol
  - Associates network address (IP) to link-layer address (MAC)
    - e.g. `192.168.20.55 is at e2:2b:e9:ee:ff:55`

- Encapsulated in Ethernet frames
  - Can be carried out over Ethernet or Wifi transports

- Needed for IPv4 to work!
  - IPv6 counterpart is named NDP (**N**eighbor **D**iscovery **P**rotocol)

- Quite simple protocol
  - Sender broadcasts a request on a LAN (*Who Has IP ?*)
  - The *legitimate* machine should respond with its MAC address

[1] RFC 826: An Ethernet Address Resolution Protocol https://www.rfc-editor.org/rfc/rfc826

1 - ARP Spoofing

ARP sequence diagram

Client
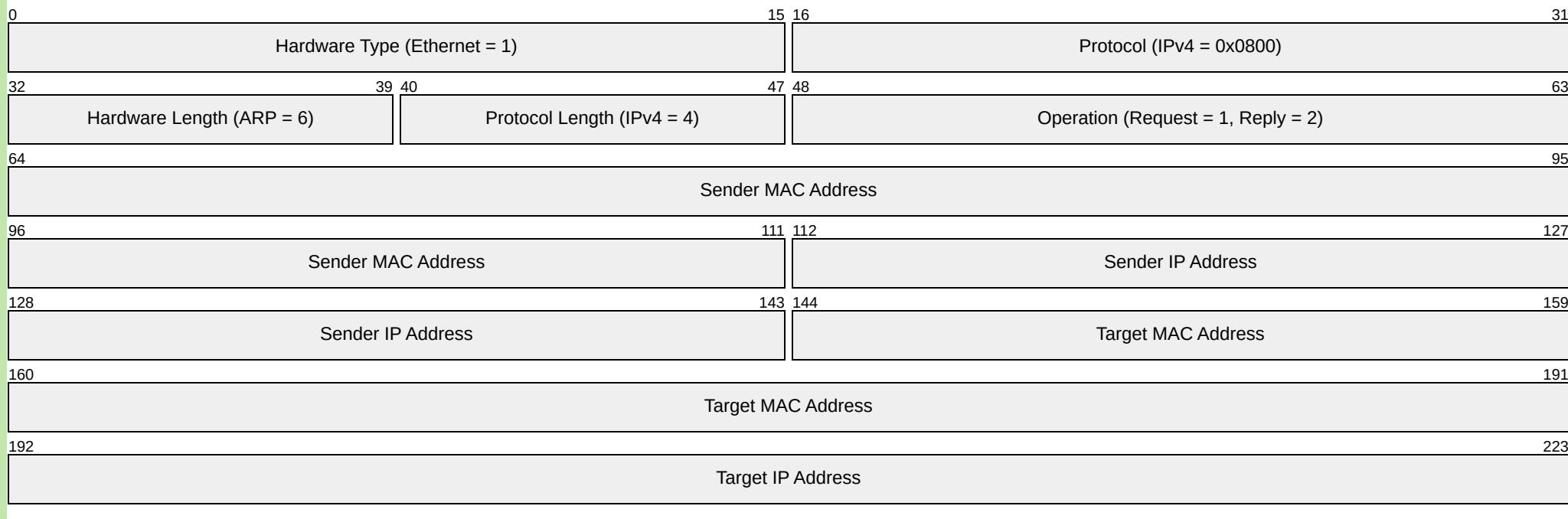
- 08:00:27:1F:1B:F3
- 192.168.56.10

Server

- 08:00:27:F3:4D:AA
- 192.168.56.20

1 - ARP Spoofing
ARP frame format

| 0	15 16	31
Hardware Type (Ethernet = 1)	Protocol (IPv4 = 0x0800)	

32	39 40	47 48	63
Hardware Length (ARP = 6)	Protocol Length (IPv4 = 4)	Operation (Request = 1, Reply = 2)	

64	95
Sender MAC Address	

96	111 112	127
Sender MAC Address	Sender IP Address	

128	143 144	159
Sender IP Address	Target MAC Address	

160	191
Target MAC Address	

192	223
Target IP Address	
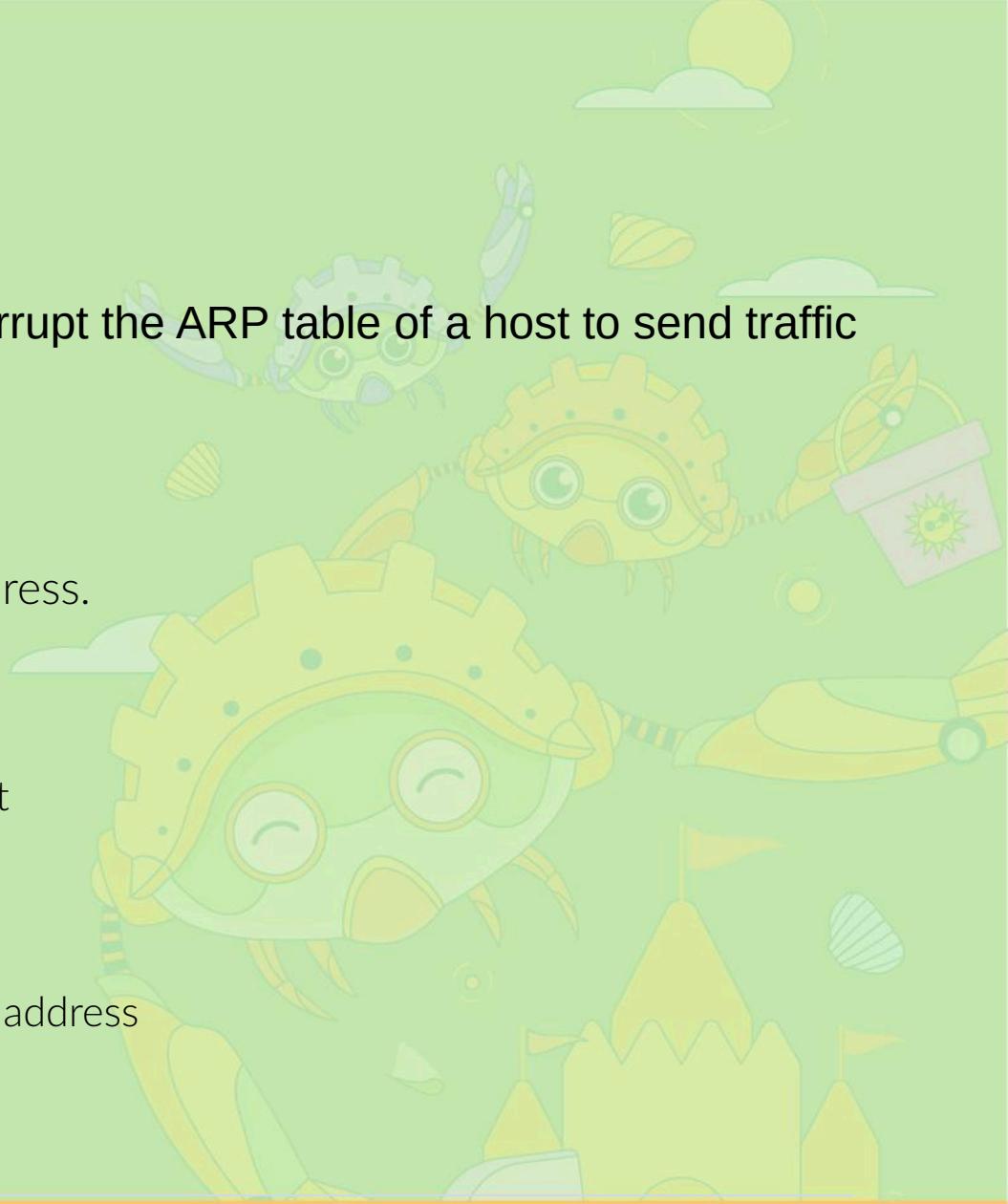
1 - ARP Spoofing

How to ?

ARP Spoofing (or **ARP cache poisoning**) is a technique to corrupt the ARP table of a host to send traffic to the "wrong" host.

2 ways to carry out an ARP Spoofing attack:
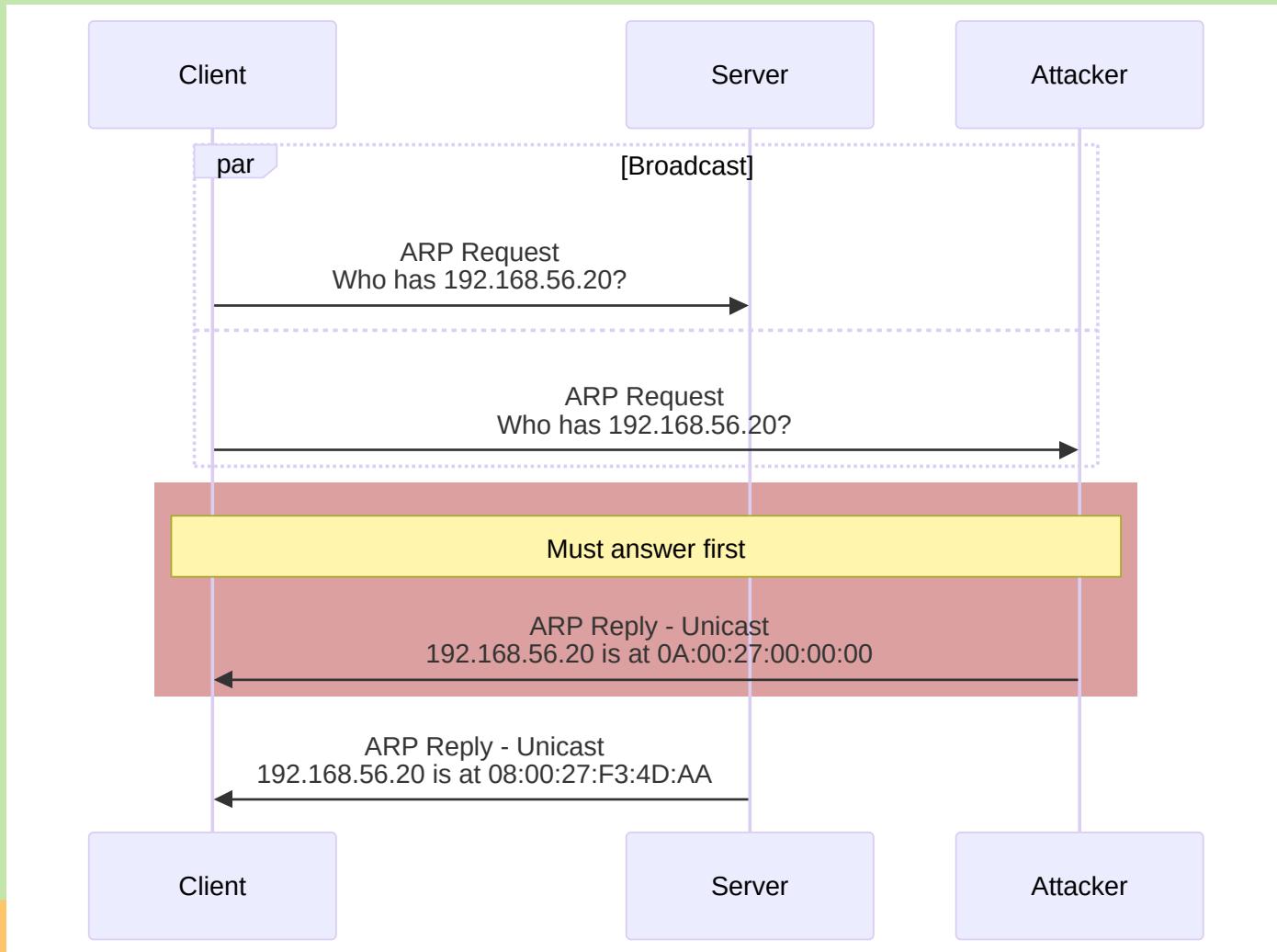
- Respond to a legitimate ARP request with a "wrong" mac address.
    - Called *Man On The Side*
    - Less visible, targeted attack
    - Relies on the fact that we can respond <u>faster</u> than the *real* host
    - *The method we are implementing today*
- Gratuitous ARP [1]
    - Flood the LAN with ARP response with our MAC but target IP address
    - Easy to implement
    - More visible

[1] RFC 5227: IPv4 Address Conflict Detection https://www.rfc-editor.org/rfc/rfc5227

<u>1</u> - ARP Spoofing
ARP Spoofing - Respond to ARP request

Client
- 08:00:27:1F:1B:F3
- 192.168.56.10
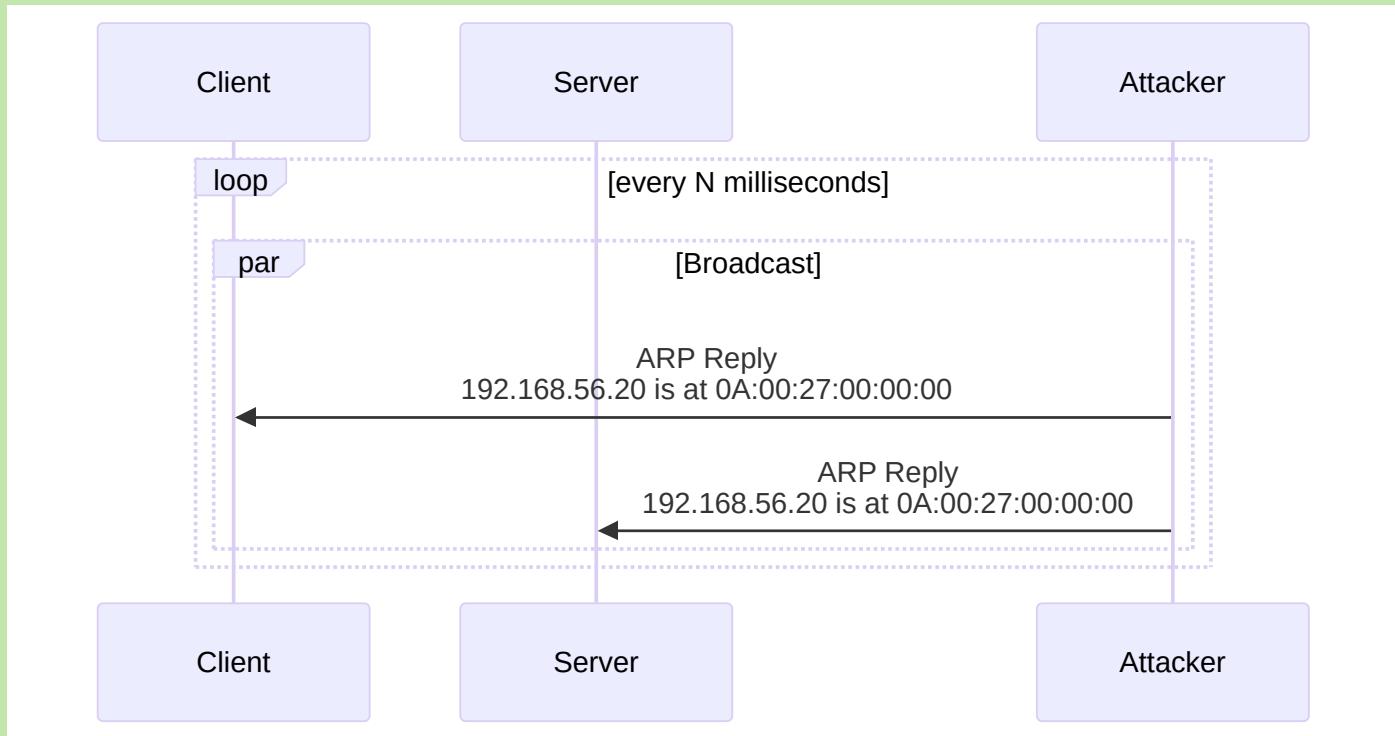
Server
- 08:00:27:F3:4D:AA
- 192.168.56.20

Attacker
- 0A:00:27:00:00:00
- 192.168.56.1

1 - ARP Spoofing

ARP Spoofing - Gratuitous ARP

Client

- 08:00:27:1F:1B:F3
- 192.168.56.10

Server

- 08:00:27:F3:4D:AA
- 192.168.56.20
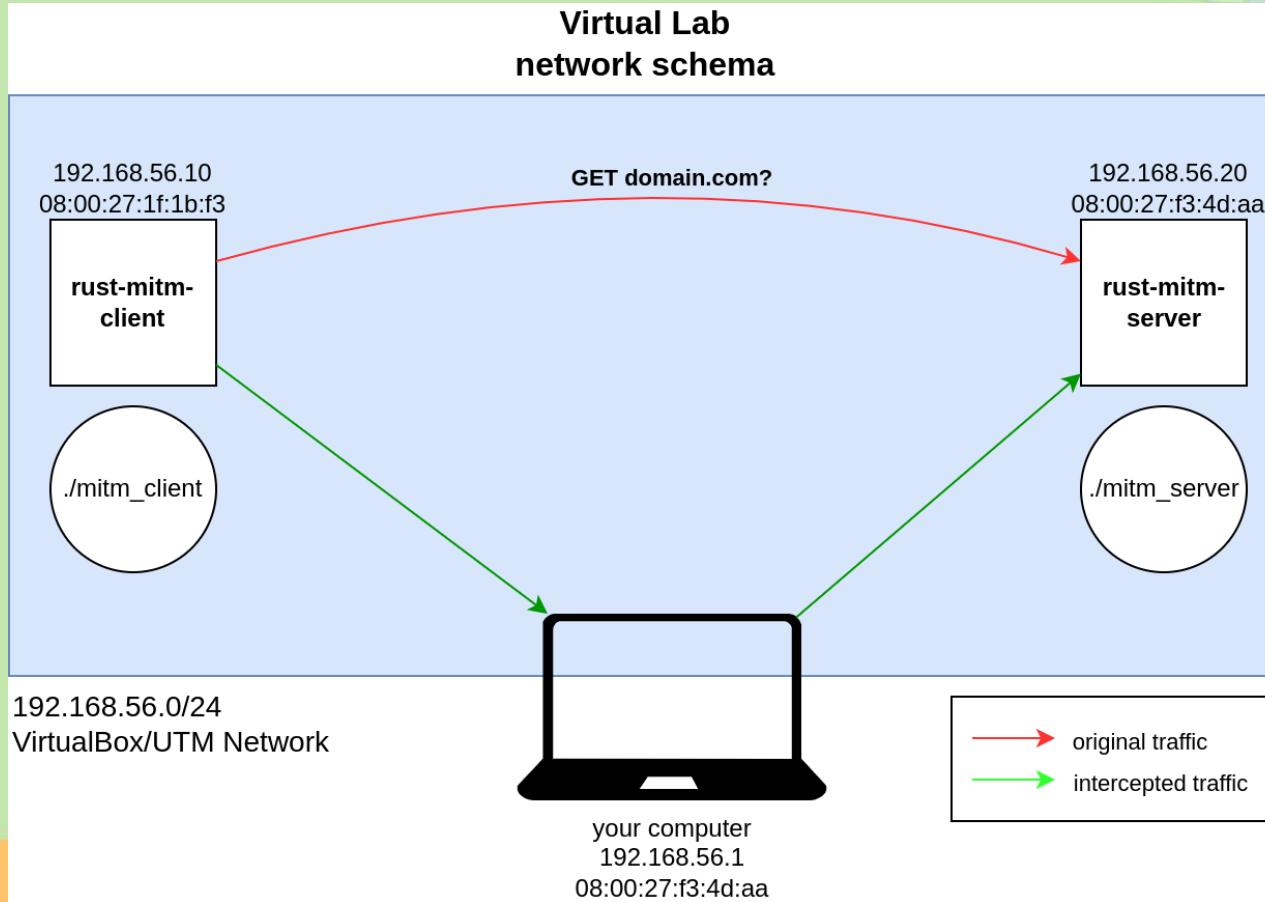
Attacker

- 0A:00:27:00:00:00
- 192.168.56.1

1 - ARP Spoofing
Virtual Lab running on your host

Follow the `setup.md` .

1 - ARP Spoofing

Exercises Ethernet & ARP

- Using `pnet` [1] crate and given code skeleton

- Complete the scenario to spoof the target by
  - Intercepting Ethernet frames
  - Sending valid ARP replies
    - Targeted client, asked IP addresses and listen interface must be configurable !
    - `wireshark` tool may help you for debugging

- `client` test binary may help you
  - You should pass **tests #0 and #1** to validate this part

- You can also `ssh` into the client VM and run `ip neigh` command to check the association

[1]: pnet: A cross-platform API for low level networking https://github.com/libpnet/libpnet

1 - ARP Spoofing
Countermeasures

- Use encrypted protocols (more on that in the next Exercises)

- Add static ARP entries for known hosts (when feasible)

- On Linux, new cache entries are always validated when received (`STALE` state)
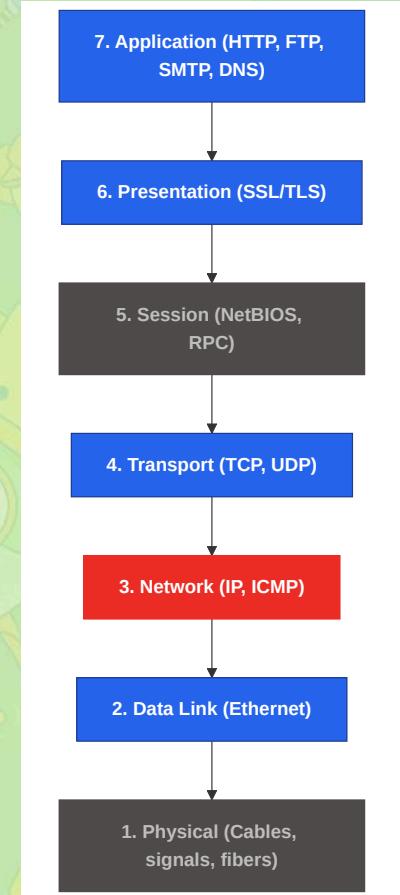
2 - Intercept IPv4 packets

2 - Intercept IPv4 packets

IPv4 protocol [1]
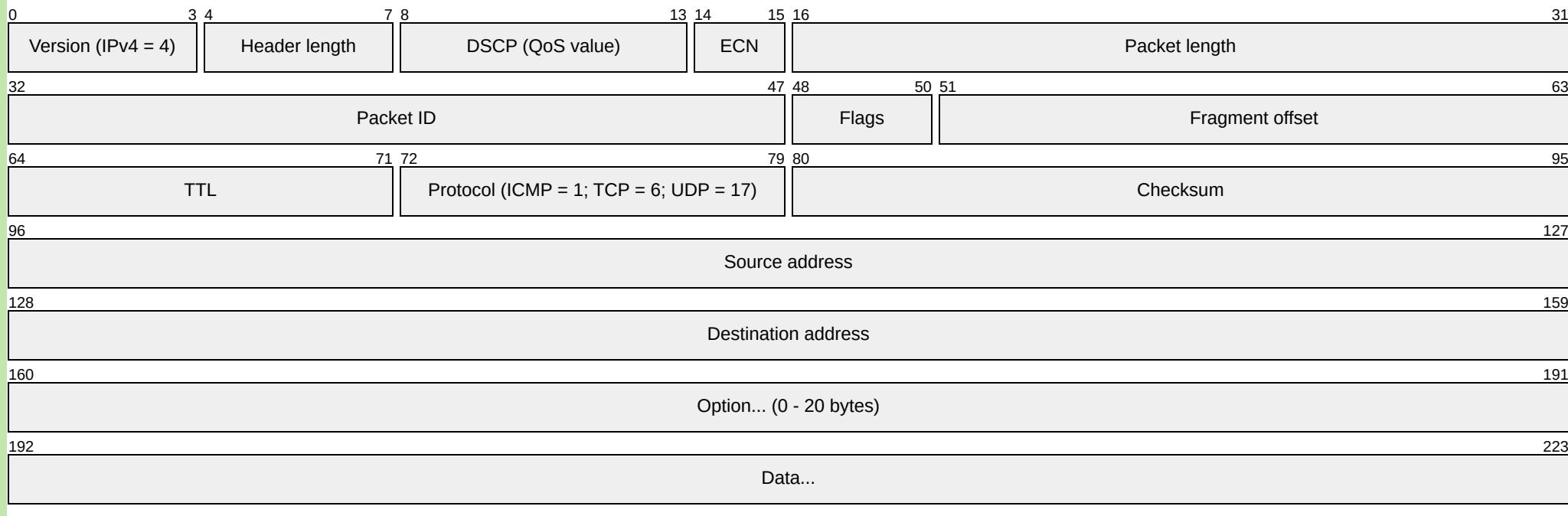
- **I**nternet **P**rotocol, version 4

- Point to point and broadcast communication

- Each host has a unique IP address on a network

- Each address is 4 bytes long (4,294,967,296 addresses)
  - e.g. `192.168.52.7` , `163.172.4.236`

- Many features:
  - Packet fragmentation
  - Quality of Service (QoS): packet priorization
  - Time-to-Live (TTL): prevent packet from being routed indefinitely
  - ...

7. Application (HTTP, FTP, SMTP, DNS)

6. Presentation (SSL/TLS)

5. Session (NetBIOS, RPC)

4. Transport (TCP, UDP)

3. Network (IP, ICMP)

2. Data Link (Ethernet)

1. Physical (Cables, signals, fibers)

[1] RFC791: Internet Protocol https://www.rfc-editor.org/rfc/rfc791

2 - Intercept IPv4 packets
IPv4 packet

0	3	4	7	8	13	14	15	16	31
Version (IPv4 = 4)		Header length		DSCP (QoS value)		ECN		Packet length	

32	47	48	50	51	63
Packet ID		Flags		Fragment offset	

64	71	72	79	80	95
TTL		Protocol (ICMP = 1; TCP = 6; UDP = 17)		Checksum	

96	127
Source address	

128	159
Destination address	

| 160	191
Option... (0 - 20 bytes)	

192	223
Data...	
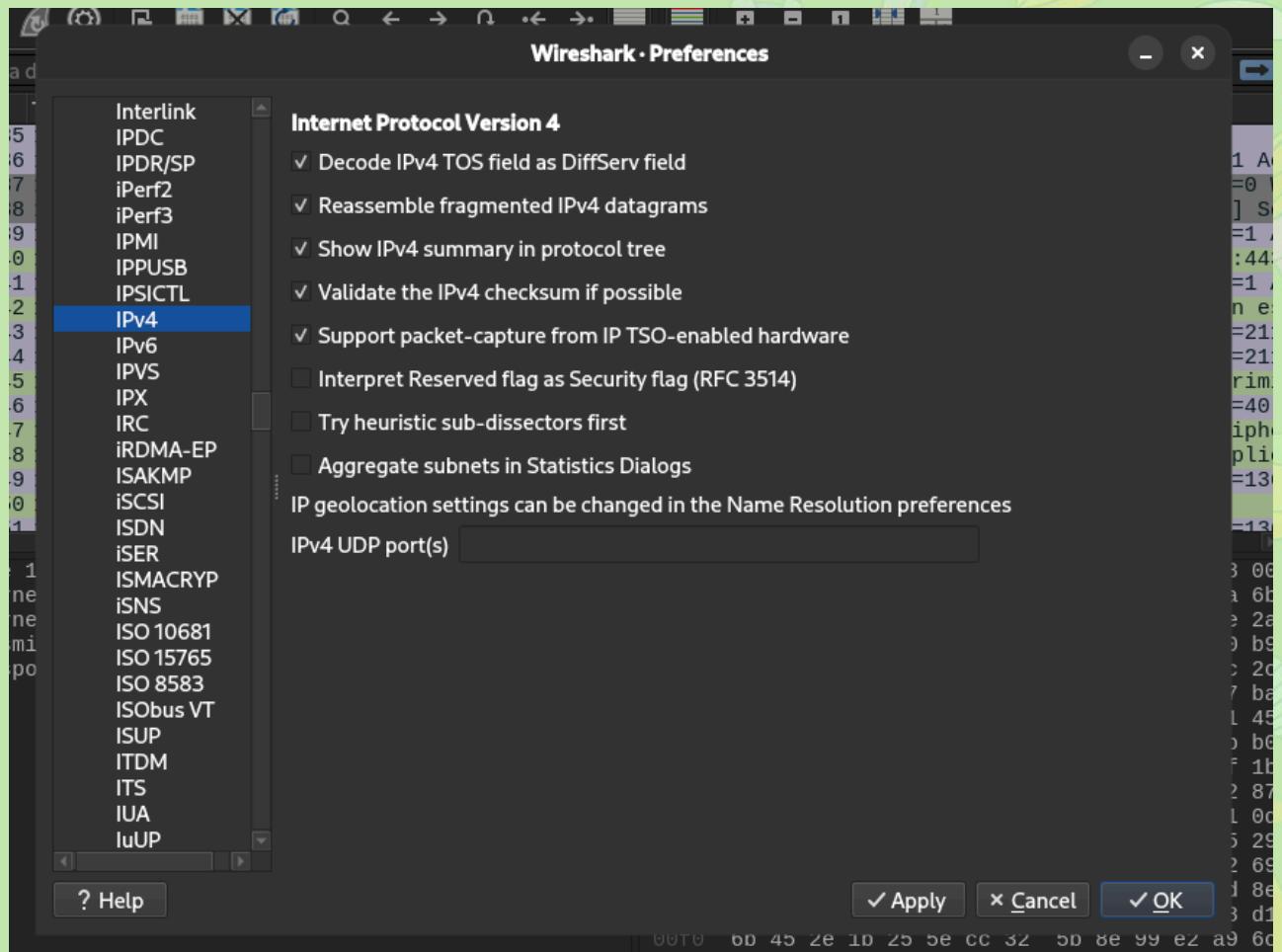
Max packet length: encoded on 2 bytes (up to 65,535 bytes)

2 - Intercept IPv4 packets
Tooling - Wireshark configuration

Activate checksum verification
in wireshark to make sure your
implementation
is correct

- Edit > Preferences >
  Protocols > IPv4 > Check
  Validate the IPv4 checksum
  if possible > Ok
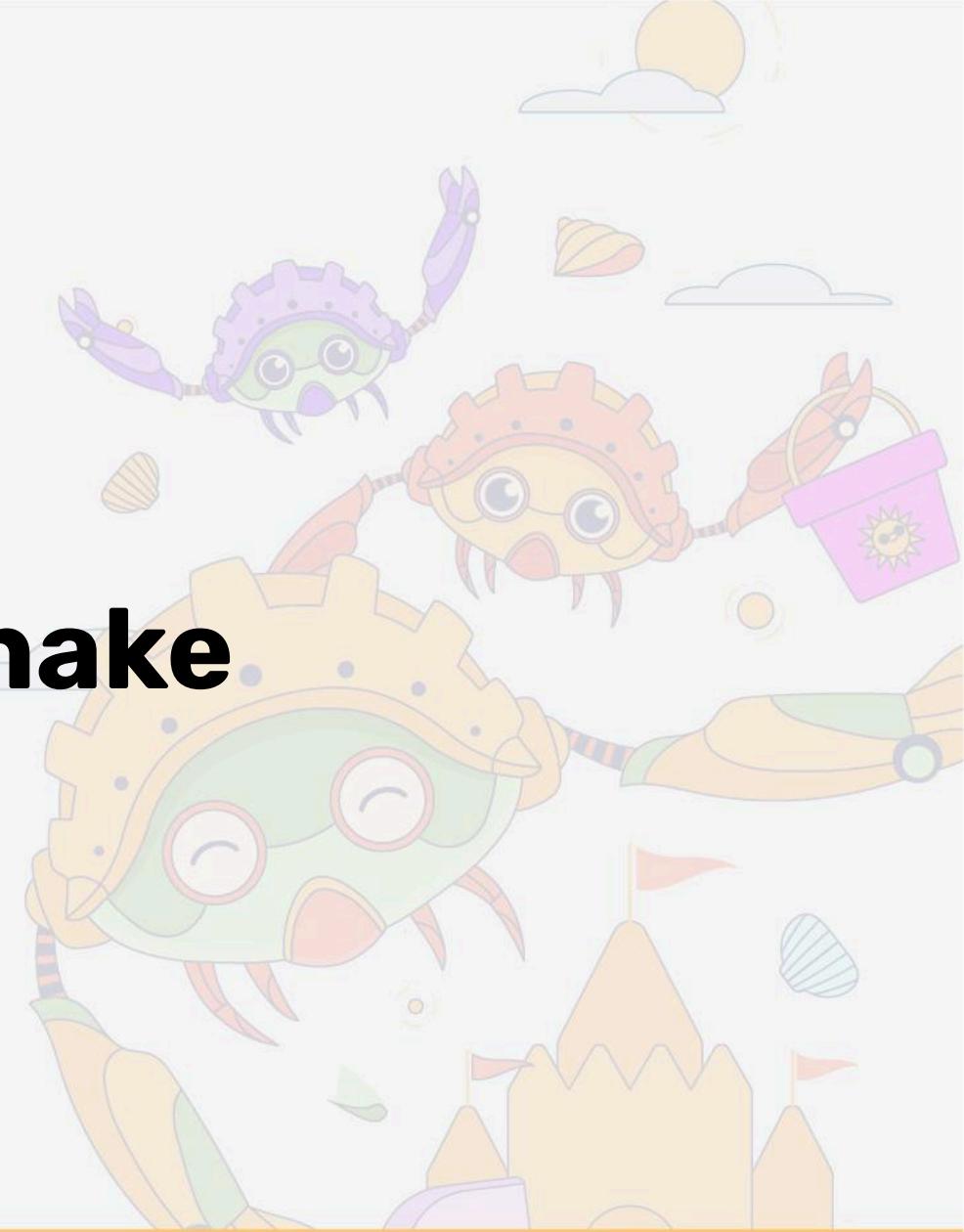
2 - Intercept IPv4 packets
Exercises IPv4 & IPv4 Test1

> After the ARP spoofing developped during Exercise 1 you should now receive IP data from the victim

- Using special `Protocol Number` 0xFD (253 - *Test1*), implement IP protocol interception
- `client` test binary may help you
  - It expects your implementation to respond with the sent data ("echo")
 - You should pass test #2 to validate this part

3 - TCP handshake
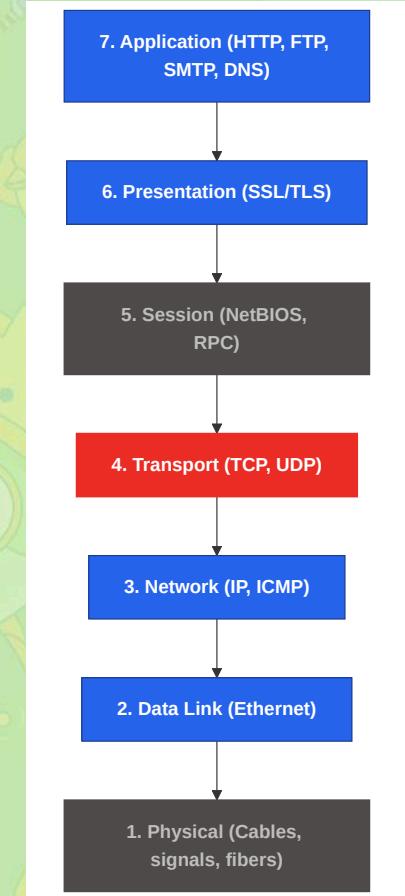
3 - TCP handshake

TCP protocol [1]

- Designed to exchange data over congested or unpredictable networks
    - Reliable, Ordered & Error checked
- Bidirectionnal
- Point-to-Point communication
- Usually paired with IP
- Relies on **port number** to identify different services on a single host with a unique IP address

Focus on **correct** data delivery more than *fast* data delivery.

Layer
7. Application (HTTP, FTP, SMTP, DNS)
6. Presentation (SSL/TLS)
5. Session (NetBIOS, RPC)
4. Transport (TCP, UDP)
3. Network (IP, ICMP)
2. Data Link (Ethernet)
1. Physical (Cables, signals, fibers)

[1] RFC 9293: Transmission Control Protocol (TCP) https://www.rfc-editor.org/rfc/rfc9293

3 - TCP handshake

TCP message types

The TCP protocol defines different flags to identify the semantic of each message
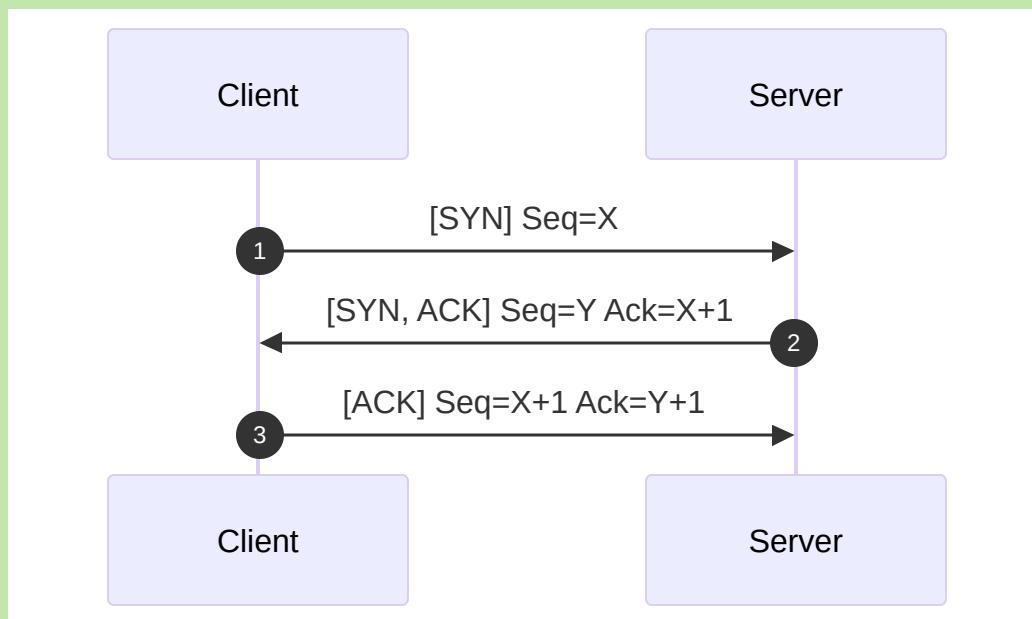
- **SYN**: Sent by each party on the first message of a new communication

- **ACK**: Acknowledge bytes sent by the peer

- **PSH**: Push data on the connection

- **FIN**: Request to close the sender sides of the connection

- **RST**: Reset the connection (closing it immediately)

- ...

TCP uses two sequence numbers (one for each direction) to keep track of the amount of data correctly received by the peer.
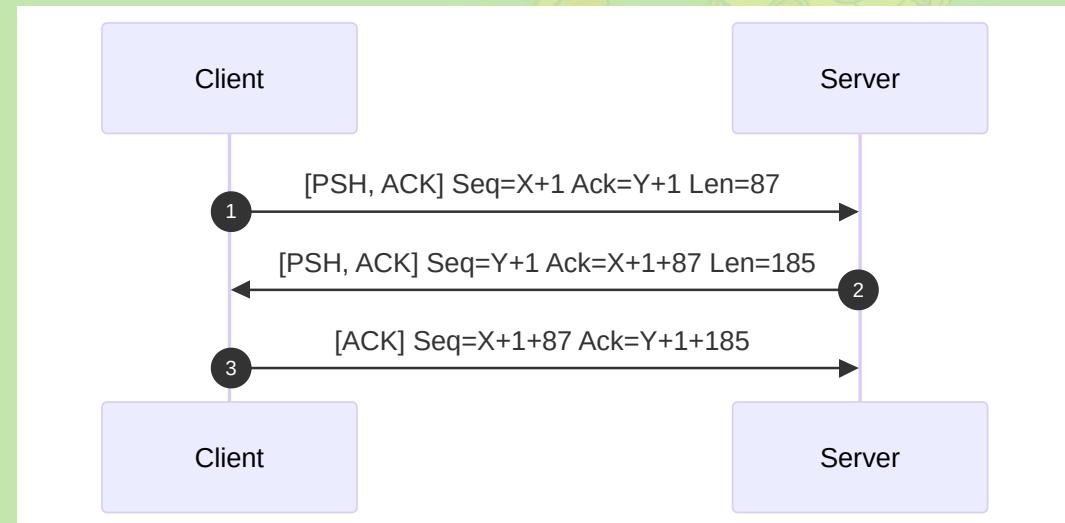
3 - TCP handshake
TCP sequence diagram - 1

New communication (handshake)


Bidirectionnal data exchange (after handshake)
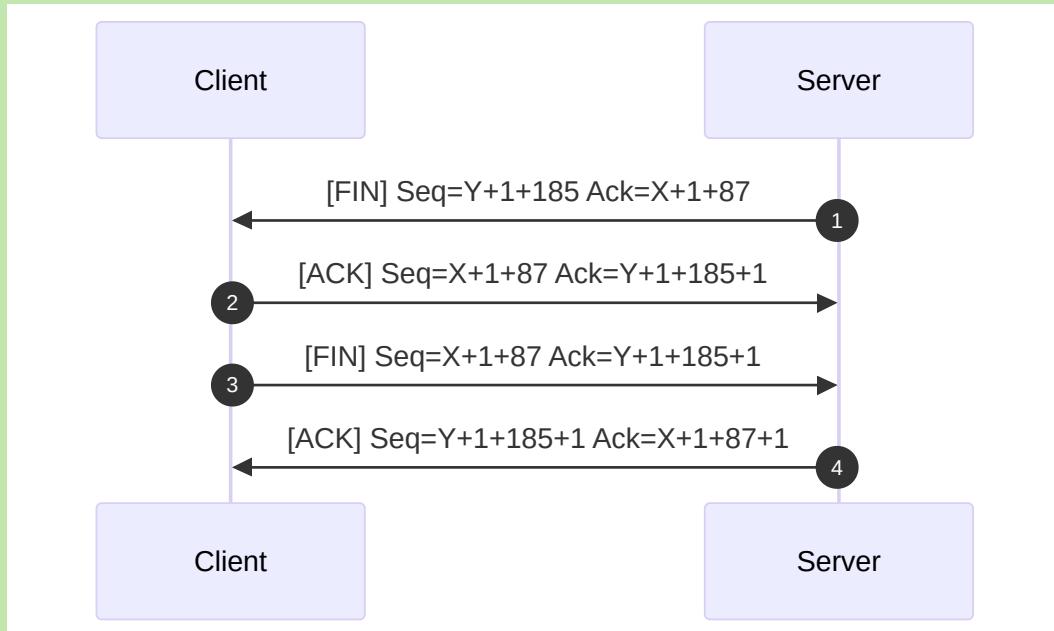
The values of X and Y are randomly generated on each new connection.
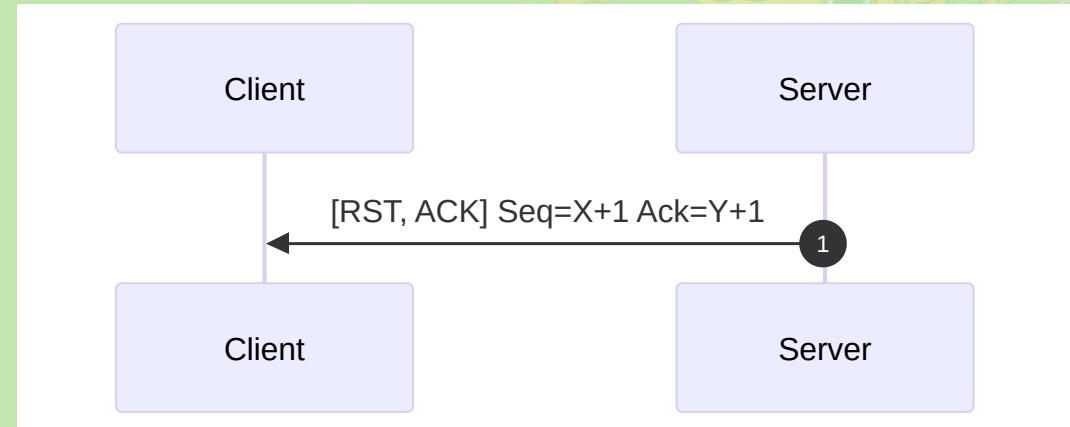
3 - TCP handshake

TCP sequence diagram - 2

Close connection (gracefully)
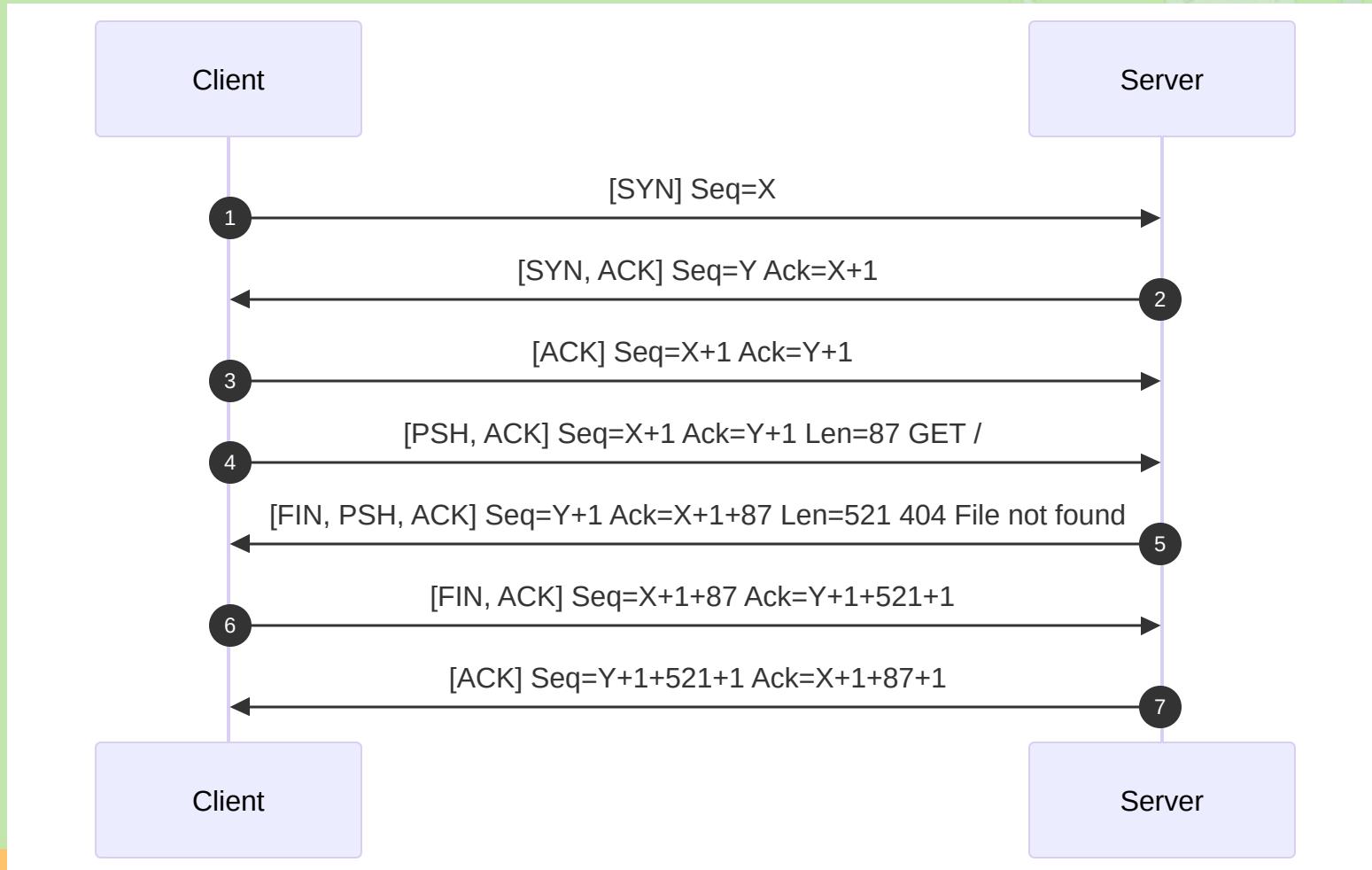
Close connection (force)

Note: It is possible to close the connection in 3 messages instead of 4, if the receiving end of the first FIN merges his FIN and ACK messages.
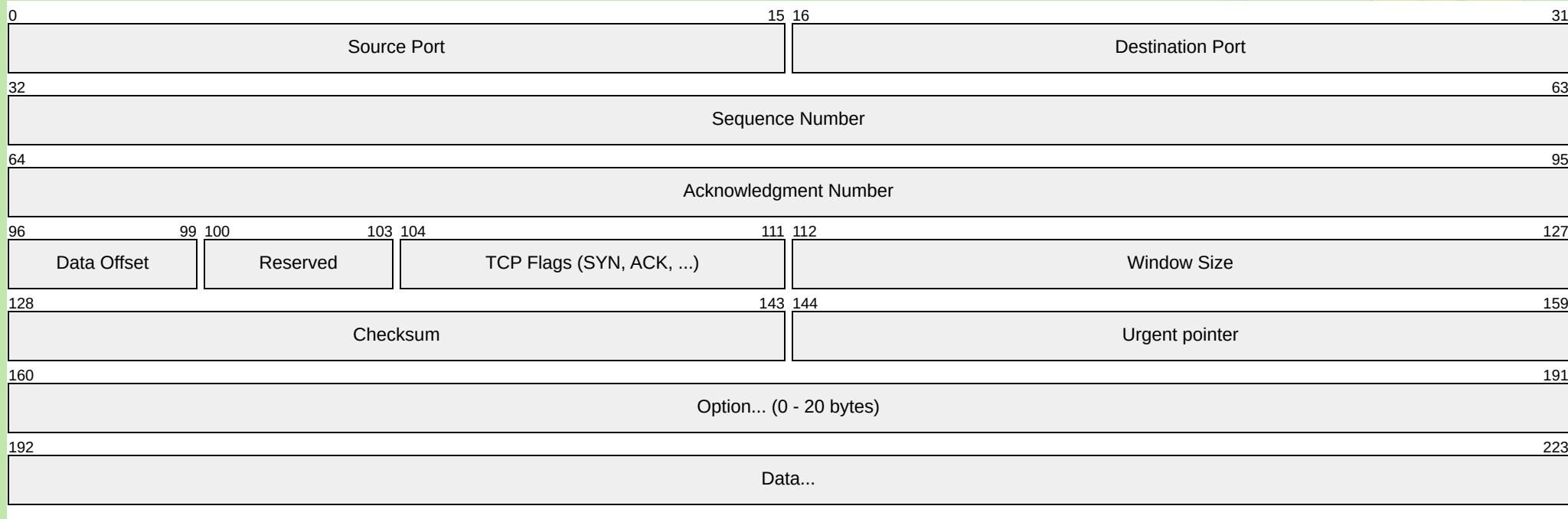
3 - TCP handshake
Example - Full HTTP GET / communication

3 - TCP handshake

TCP packet

0 15	16 31	
Source Port	Destination Port	

32	63
Sequence Number	

64	95
Acknowledgment Number	

96 99	100 103	104 111	112 127
Data Offset	Reserved	TCP Flags (SYN, ACK, ...)	Window Size

128 143	144 159
Checksum	Urgent pointer

160	191
Option... (0 - 20 bytes)	

| 192	223
Data...	
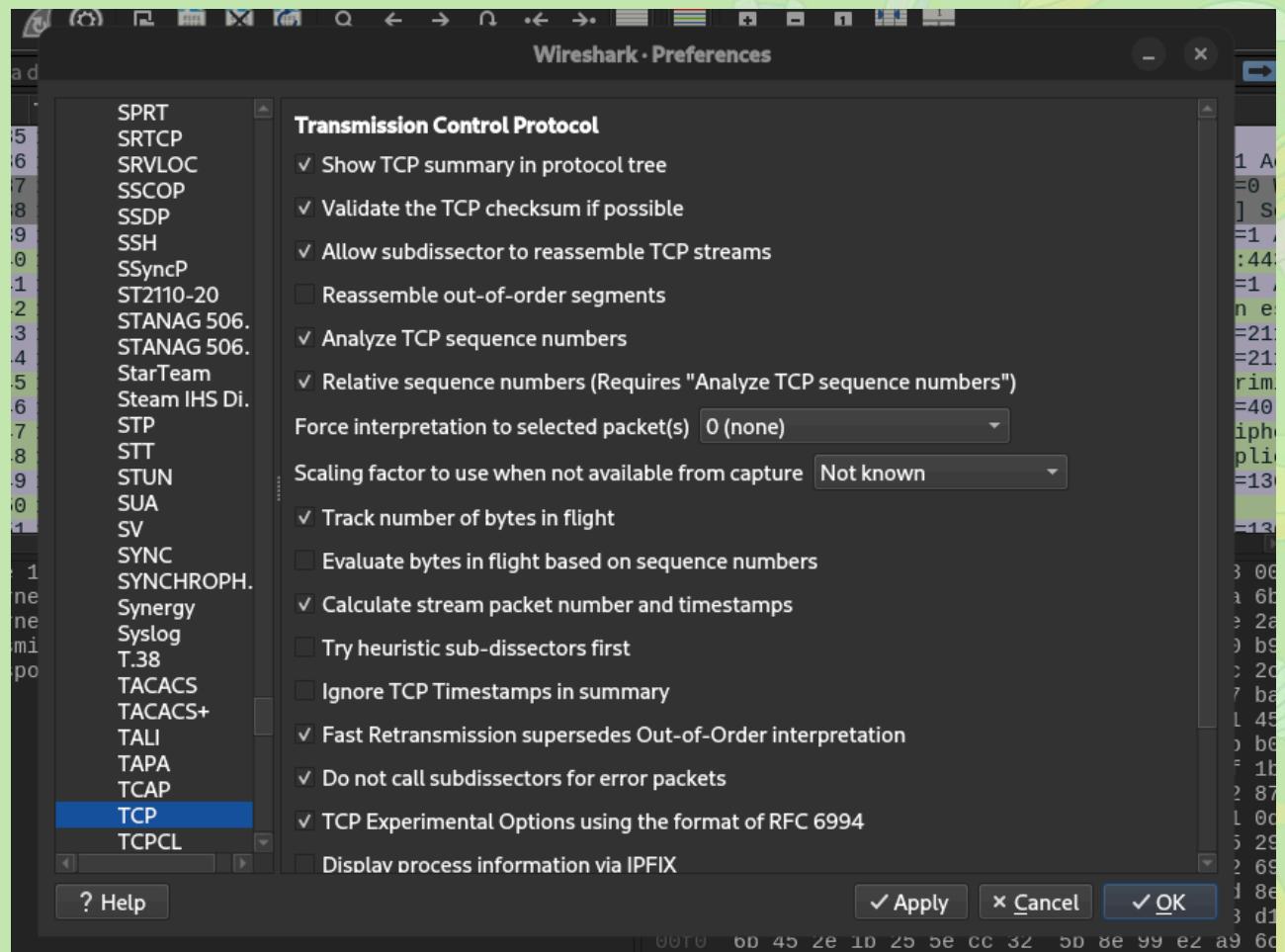
3 - TCP handshake

Tooling - Wireshark configuration

Activate checksum verification
in wireshark to make sure your
implementation
is correct

- Edit > Preferences >
  Protocols > TCP > Check
  Validate the TCP checksum
  if possible > Ok

3 - TCP handshake

Exercises TCP & TCP Echo

> After the IPv4 interception developped in Exercise 2, you should now receive TCP data from the victim.

- Using custom port `4000`, implement TCP protocol interception
- `client` test binary may help you
  - It also expects your implementation to respond with the sent data ("echo")
  - You should pass tests #3 and #4 to validate this part

4 - Intercept HTTP requests

<u>4</u> - Intercept HTTP requests
HTTP Protocol
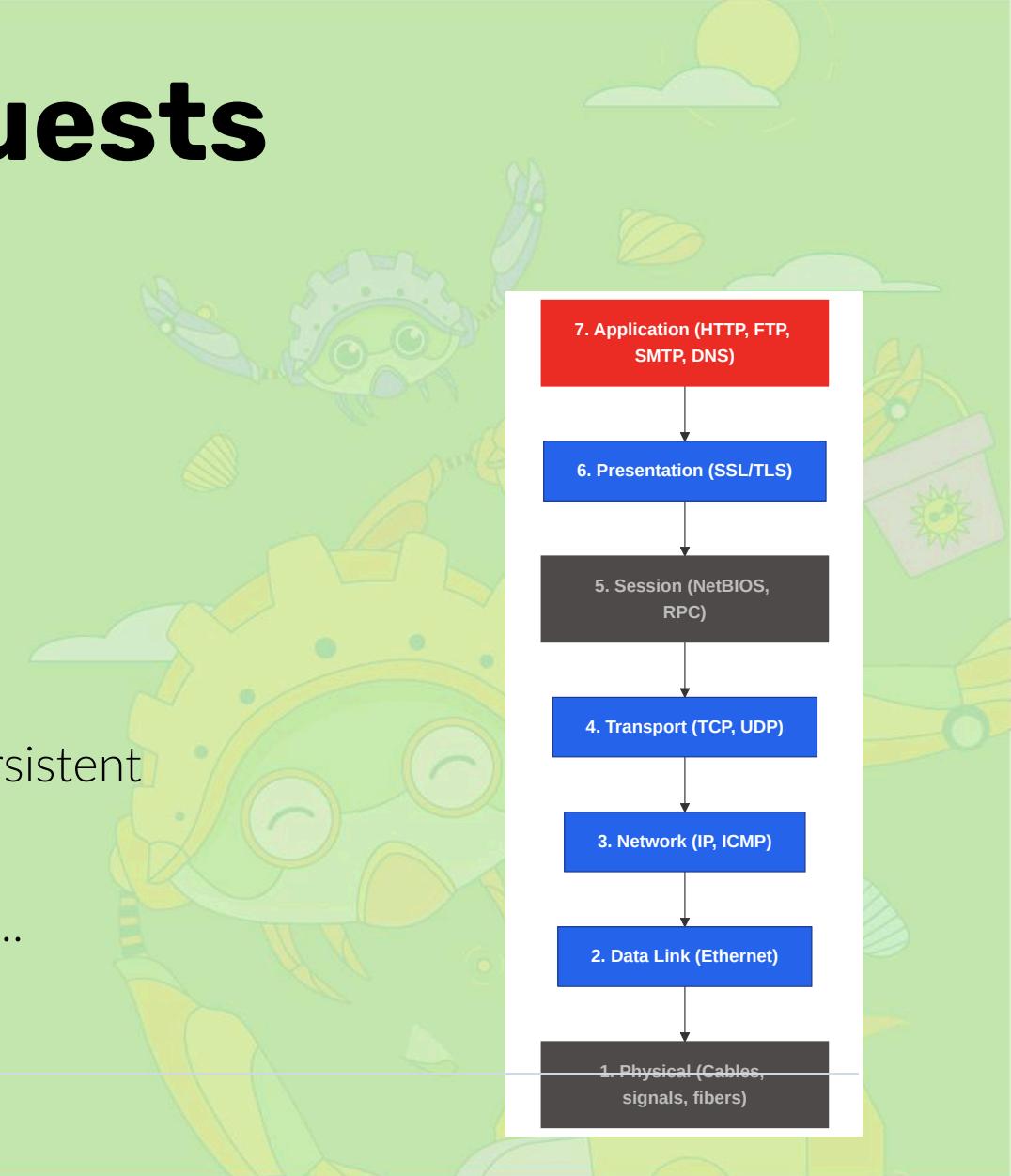
- Request-response point to point protocol

- Client-server model

- Multiple version still in use
  - HTTP/1.0[1]: text frame over TCP (deprecated)
  - HTTP/1.1[2]: Host header required, OPTIONS, persistent connections, ...
  - HTTP/2.0[3]: binary frame, simulatenous request, ...
  - HTTP/3.0[4]: uses UDP and QUIC instead of TCP

[1]: RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0 https://www.rfc-editor.org/rfc/rfc1945
[2]: RFC 9112: HTTP/1.1 https://www.rfc-editor.org/rfc/rfc9112
[3]: RFC 9113: HTTP/2 https://www.rfc-editor.org/rfc/rfc9113
[4]: RFC 9114: HTTP/3 https://www.rfc-editor.org/rfc/rfc9114

4 - Intercept HTTP requests
HTTP Protocol Frame

HTTP Response

```
HTTP/1.1 200 OK
Content-Length: 18274
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
...
</html>
```

HTTP Request

```
GET /index.html HTTP/1.1
Host: website.com
```
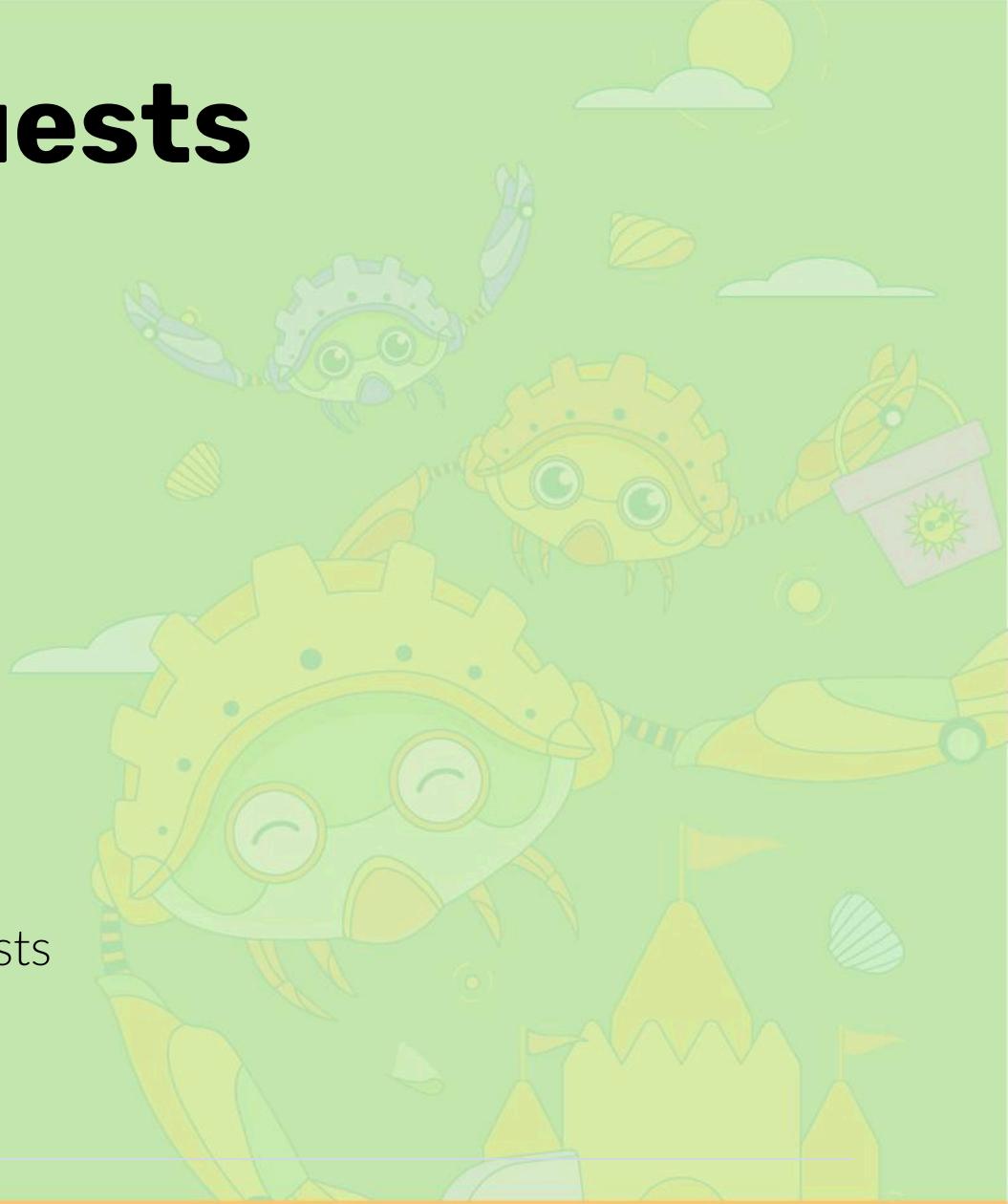
4 - Intercept HTTP requests
HTTP Protocol Sequence

<u>4</u> - Intercept HTTP requests

Recommended libraries

- `ureq` [1]
  - Fully synchronous HTTP client

- `httparse` [2]
  - HTTP parsing library
  - Only supports HTTP/1.1
  - Hints about `Partial` or `Complete` HTTP requests

[1]: `ureq` documentation https://docs.rs/ureq/latest/ureq

[2]: `httparse` documentation https://docs.rs/httparse/latest/httparse/

4 - Intercept HTTP requests

Exercise HTTP

> After the TCP interception from Exercise 3 you should now receive HTTP requests from the victim.

- The victim tries to connect to a website using HTTP. Can you figure out what domain it is using your MITM?
- The victim also seems to connect using credentials, can you intercept and log the username and password?
- Implement HTTP interception by logging credentials & forwarding data from the upstream server to the client
- You can use `client` binary to test your implementation
- You should pass **test #5** to validate this part
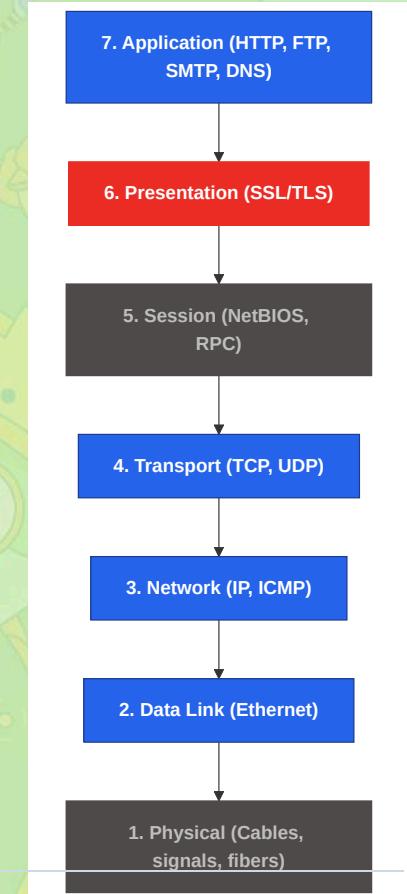
5 - Handle TLS handshake

5 - Handle TLS handshake

TLS Protocol

- Provide security to communications, including confidentiality, integrity, and authenticity (CIA) over various transports

- Based on asymmetric cryptography (via certificates)

- Different versions
  - SSL1.0-3.0 (1994 to 1996): deprecated
  - TLS1.0 / TLS1.1[1] (2006): deprecated since March 2021
 - TLS1.2[2] (2008)
  - TLS1.3[3] (2018)

| 7. Application (HTTP, FTP, SMTP, DNS) |
| 6. Presentation (SSL/TLS) |
| 5. Session (NetBIOS, RPC) |
| 4. Transport (TCP, UDP) |
| 3. Network (IP, ICMP) |
| 2. Data Link (Ethernet) |
| 1. Physical (Cables, signals, fibers) |
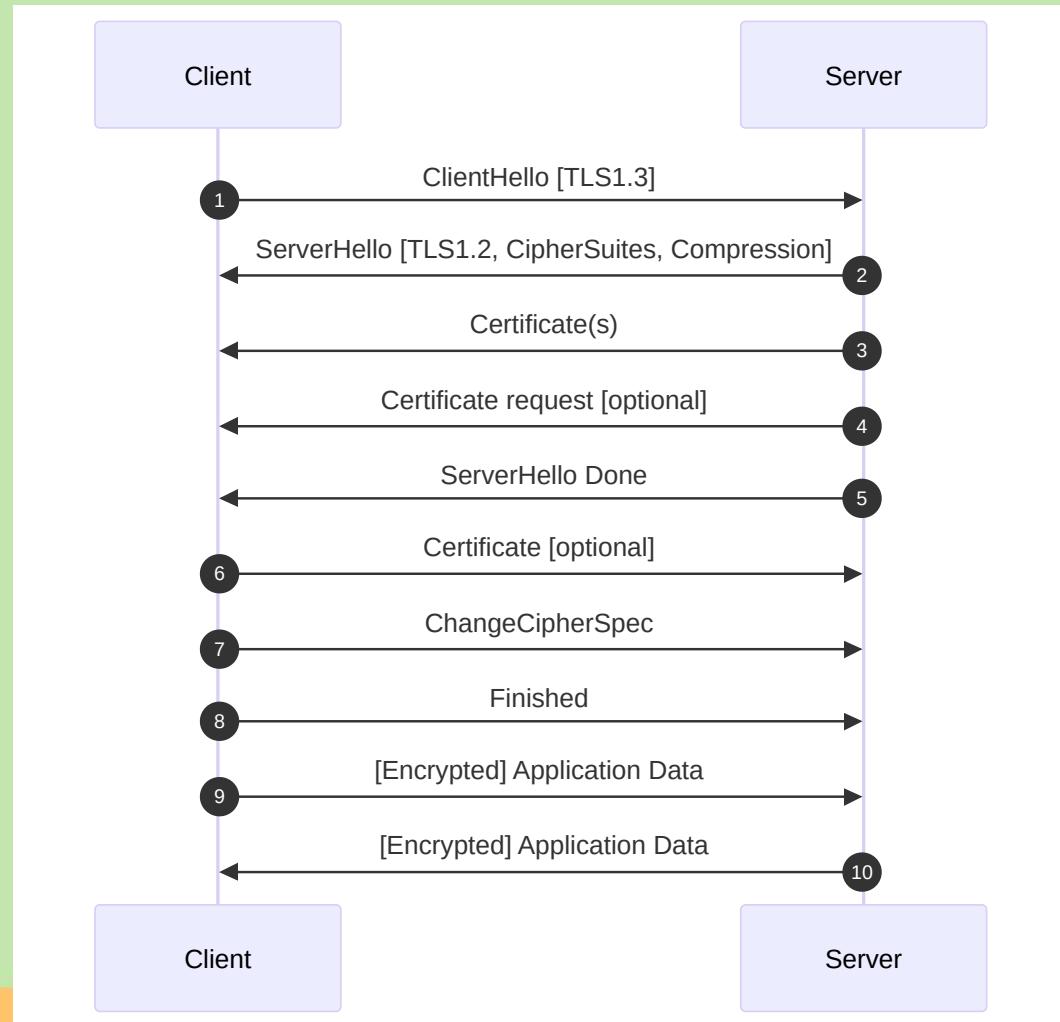
[1]: RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1 https://www.rfc-editor.org/rfc/rfc4346
[2]: RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 https://www.rfc-editor.org/rfc/rfc5246
[3]: RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 https://www.rfc-editor.org/rfc/rfc8446

5 - Handle TLS handshake
TLS Protocol

- Client initiates the handshake with its TLS version and ciphersuites
- The server responds with various information like its certificate, TLS version, chosen ciphersuite / compression...
- This is then validated by the client using a `ChangeCipherSpec` message.

5 - Handle TLS handshake
TLS Protocol

- TLS sessions can be negociated over various protocols
    - TCP (our use-case)
    - UDP

- Protocols using TLS
    - HTTPS (over TCP or UDP via QUIC)
    - DOT (*DNS Over TLS*)
    - SMTPS
    - ...

5 - Handle TLS handshake
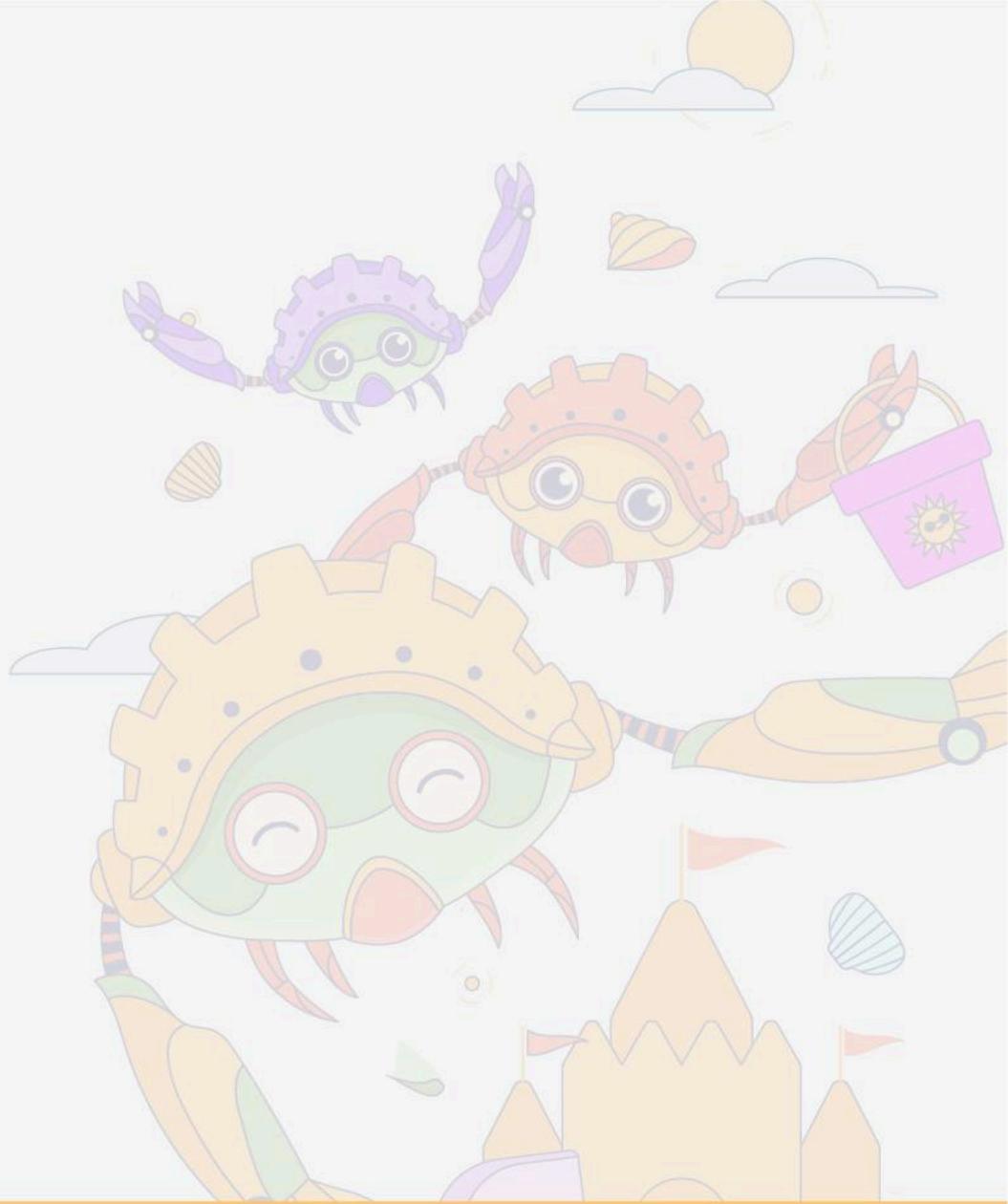
Exercise TLS

> Using the HTTP interception from Exercise 4 we will now try to intercept TLS from our victim.

- The TLS configuration of the victim is bad and does not check provided certificates.
- You can MITM the victim by providing a self-signed certificate:
    - Either on-the-fly (generating it for the asked domain)
    - Or one-time (using `openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes -subj "/CN=mitm.com"`)
- TLS layer should then call on your previous HTTP layer :)
- `rustls` [1] library is a good choice to manage the TLS session
- You can use `client` binary to test your implementation
    - You should pass test #6 to validate this part

[1]: rustls: A modern TLS library https://github.com/rustls/rustls

<u>Bonus</u>

Bonus

Capture The Flag

You can now inject data into the *real* HTTP server response before forwarding it to the client, either from TLS or HTTP connections.

- This can be done by adding e.g a `<script>alert("pwned")</script>` .

- You can use `client` binary to test your implementation

- You can also open firefox on the client VM and navigate to http://FOUND_DOMAIN.com/index.html

- You should pass test #7 to validate this part

Bonus

Advanced tests
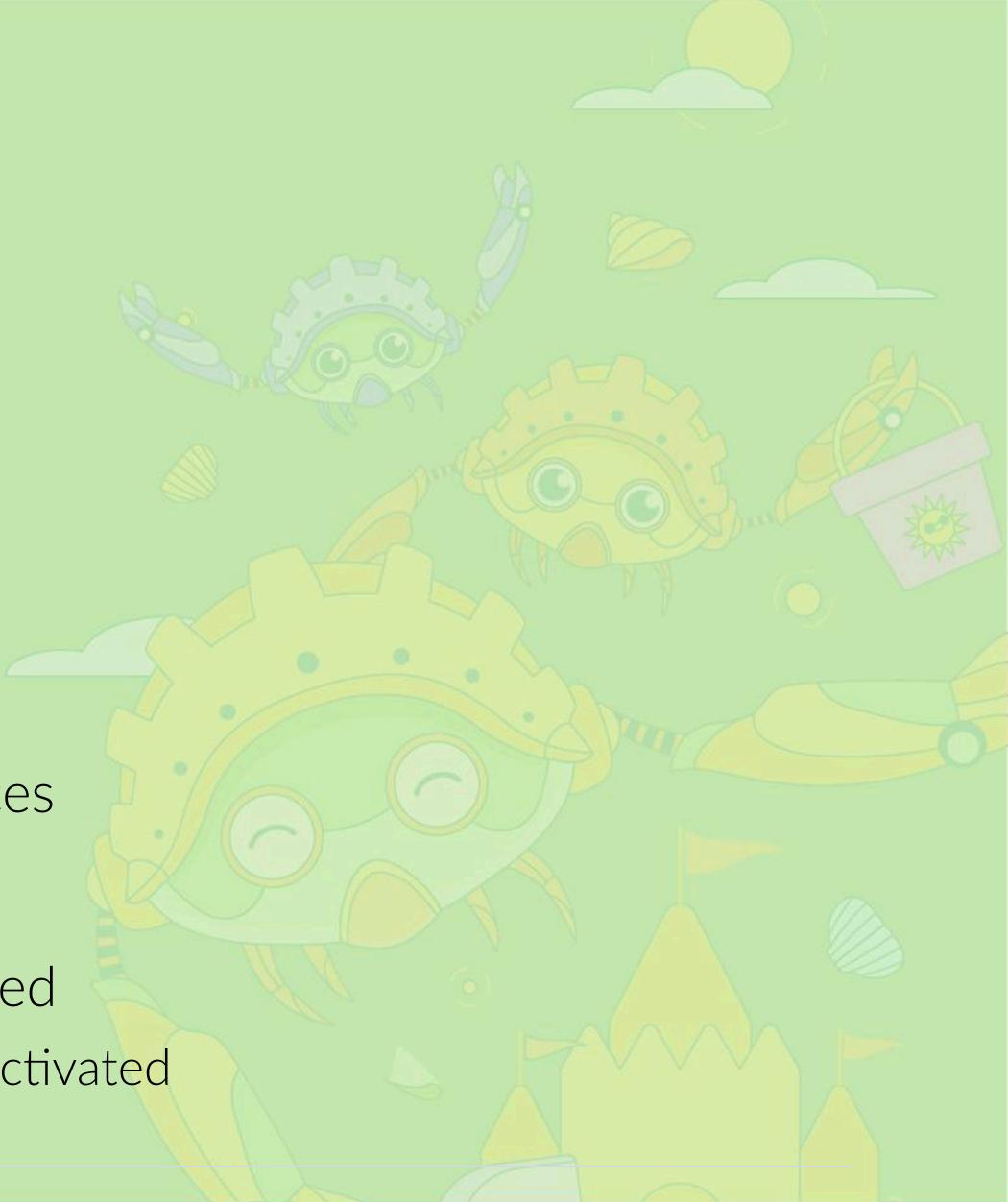
A few "advanced" tests have been implemented to experiment more protocol features.

They can be activated using `--advanced` option of `client`

- **[IP]** Packet with options

- **[TCP]** Echo with large payloads (validating MTU handling)

- **[TCP]** Maximum Segment Size handling

- **[HTTP]** Content-Encoding: gzip upstream responses

Real-life scenario
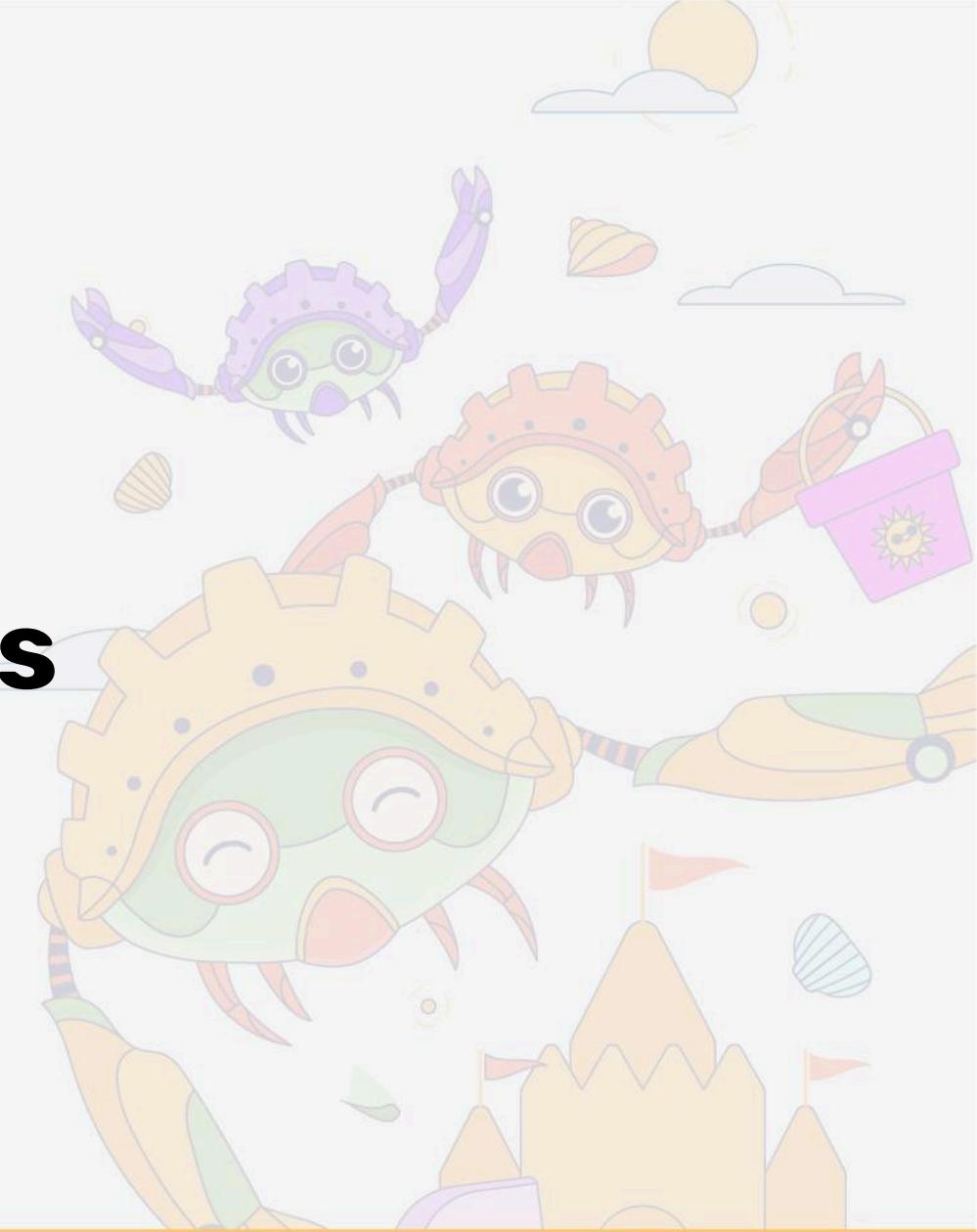
- Implement IPv6 features

- Implement missing TCP features

  - MTU management

  - Window size

  - Retransmission

- Multithreading treatments to improve performances

- Attack a gateway + dhcp spoofing

- In 2025, around 90%[1] of internet trafic is encrypted

  - Real-life clients (should?[2]) not have insecure TLS activated

[1] Traffic initiated by Google Chrome https://transparencyreport.google.com/https/overview
[2] CVE-2024-2466 in libcurl https://nvd.nist.gov/vuln/detail/cve-2024-2466

Appendices

Appendices

Physical lab
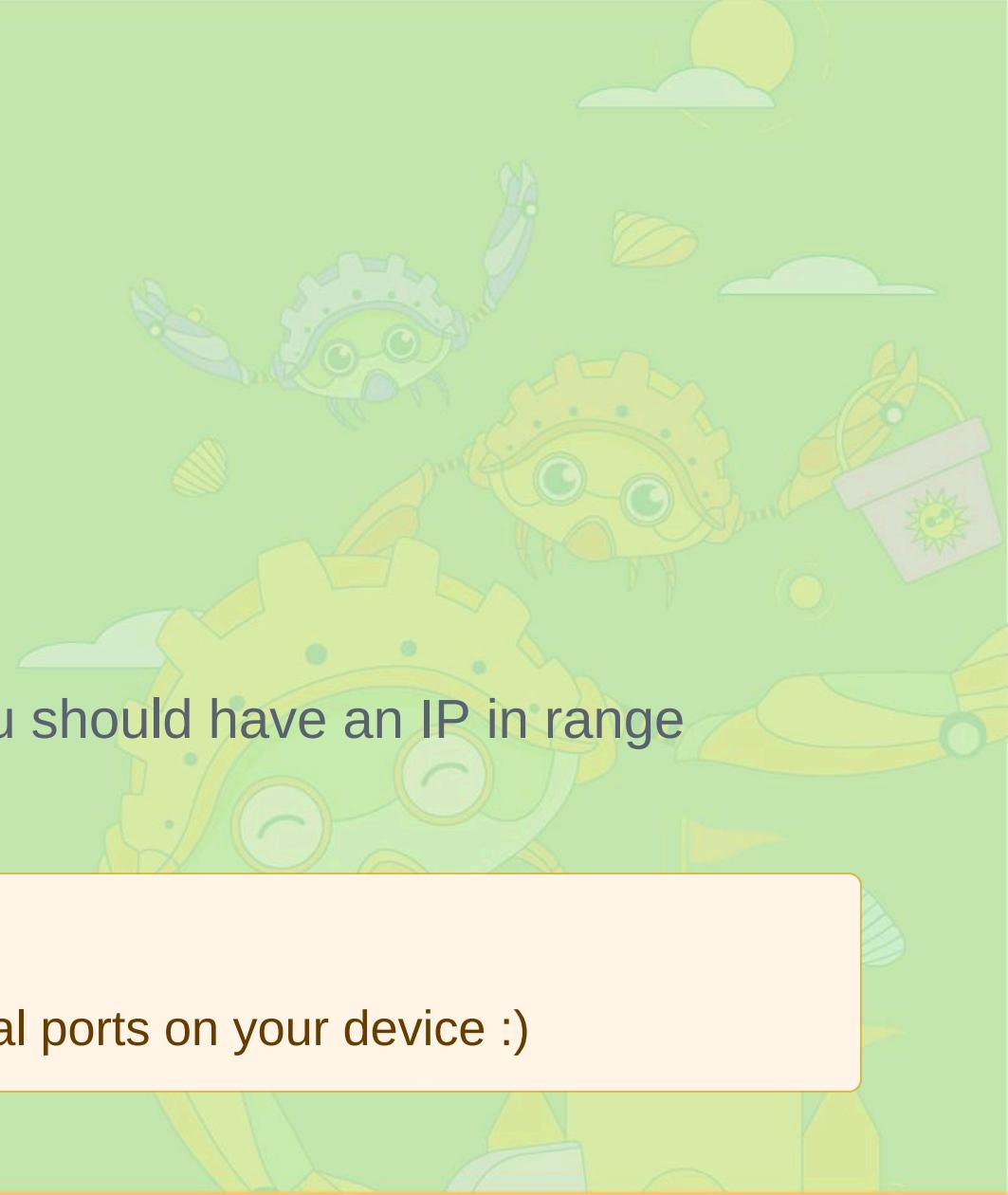
- SSID: `RUSTLAB-MITM`

- WPA2

- Password: `KhiXSvrrbCj634eXmstvcWVisw`

Note: A DHCP is running on the access point, you should have an IP in range

`172.16.100.[50-99]`

> ⚠ Exposed services
>
> Lab is setup at layer 2, be careful to not expose local ports on your device :)

Appendices

Known IPv4 protocol numbers

| Number	Protocol	Description
1	**ICMP**	Internet Control Message Protocol, used for error messages and diagnostics (e.g., ping)
6	**TCP**	Transmission Control Protocol, connection-oriented protocol for reliable communication
17	**UDP**	User Datagram Protocol, connectionless protocol for fast but unreliable communication
41	**IPv6**	IPv6 encapsulation within IPv4
58	**ICMPv6**	ICMP for IPv6, error messages and diagnostics for IPv6
89	**OSPF**	Open Shortest Path First, an internal routing protocol (IGP)
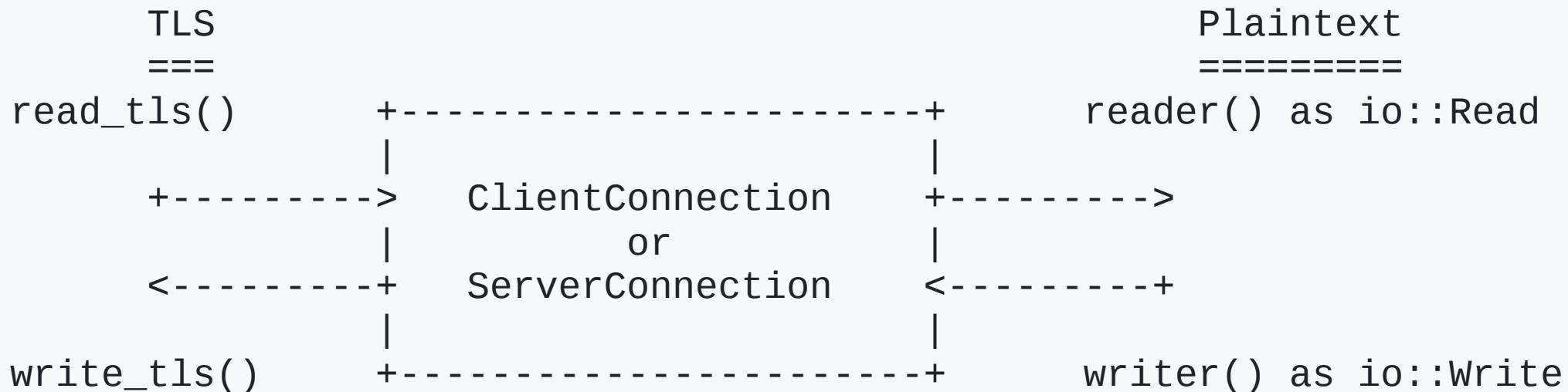
Appendices

Known TCP port numbers

Port	Service	Description
22	**SSH**	Secure Shell, for secure remote login and file transfer
25	**SMTP**	Simple Mail Transfer Protocol, for sending email
53	**DNS**	Domain Name System, resolving hostnames to IP addresses
80	**HTTP**	Hypertext Transfer Protocol, web traffic
443	**HTTPS**	HTTP over TLS/SSL, secure web traffic
465	**SMTPS**	SMTP over SSL, secure email sending

Appendices

rustls `ServerConnection` workflow

```
      TLS                              Plaintext
      ===                              =========
read_tls()       +------------------------+       reader() as io::Read
                 |                        |
      +--------->   ClientConnection     +-------->
                 |         or            |
      <---------+   ServerConnection     <---------+
                 |                        |
write_tls()      +------------------------+       writer() as io::Write
```

Appendices

Force MAC address for a given host

> This command can be run on the `client` to "fake" the ARP spoofing part

```
CLIENT_IFACE_NAME=enp1s0
MITM_MAC_ADDRESS=0A:00:27:00:00:00
SERVER_IP_ADDR=192.168.56.20

# Linux
sudo ip neigh add $SERVER_IP_ADDR lladdr $MITM_MAC_ADDRESS dev $CLIENT_IFACE_NAME permanent
```

```
set MITM_MAC_ADDR = 08-00-27-f3-4d-aa
set MITM_IP_ADDR = 192.168.56.1
set SERVER_IP_ADDR = 192.168.56.20

:: Windows
arp -s %SERVER_IP_ADDR% %MITM_MAC_ADDR% %MITM_IP_ADDR%
```