ARP Poisoning and DNS Spoofing Tool

2IC80 Lab on Offensive Computer Security

Julien Erbland (2232243)

August 2025

Abstract

This report presents an improved version of a 2IC80 lab project. It documents a Python tool that performs ARP poisoning and DNS spoofing in a local network. The tool supports multiple targets, stealth modes, and uses a MITM position to spoof DNS responses reliably.

GitHub Repository

Contents

1 Introduction **2**

2 Attack Description **2**

2.1 ARP Poisoning 2

2.2 DNS Spoofing 3

3 Technical Setup **4**

4 Attack Analysis **4**

4.1 Scenario 1: ARP Poisoning in a Local Network 4

4.2 Scenario 2: DNS Spoofing via Man-in-the-Middle 5

5 Attack Engineering **6**

5.1 Network Discovery Phase 6

5.2 ARP Spoofing Mechanism 6

5.3 DNS Spoofing Mechanism 8

6 Conclusion and Future Work **9**

1 Introduction

In local area networks (LANs), devices use the Address Resolution Protocol (ARP) to associate IP addresses with hardware (MAC) addresses, enabling packet delivery within the subnet. Similarly, the Domain Name System (DNS) is responsible for translating human-readable domain names into IP addresses, allowing users to access websites and services. Both ARP and DNS are fundamental to daily network operations. They are implemented on most systems and play a key role in traffic routing and name resolution.

This report focuses on the development of a tool that performs ARP poisoning and DNS spoofing in a LAN environment. The objective is to implement the attacks using Python and Scapy, automate them across different scenarios, and analyze their technical behavior in detail.

2 Attack Description

This section explains the two attacks implemented in the project: ARP poisoning and DNS spoofing. They allow an attacker connected to the same local area network (LAN) as the victim to intercept and manipulate traffic.

2.1 ARP Poisoning

The Address Resolution Protocol (ARP) is used within a local network to resolve IP addresses into MAC addresses. When a device wants to send data to an IP address, it sends an ARP request asking "who has this IP?" and expects a response with the correct MAC address.

In ARP poisoning, the attacker sends forged ARP replies to the victim and the gateway. These replies falsely associate the attacker's MAC address with the IP address of the gateway (to the victim) and vice versa (to the gateway). As a result, both the victim and the gateway send their packets to the attacker, who can inspect, modify, or forward them. This creates a man-in-the-middle (MITM) position for the attacker within the network.

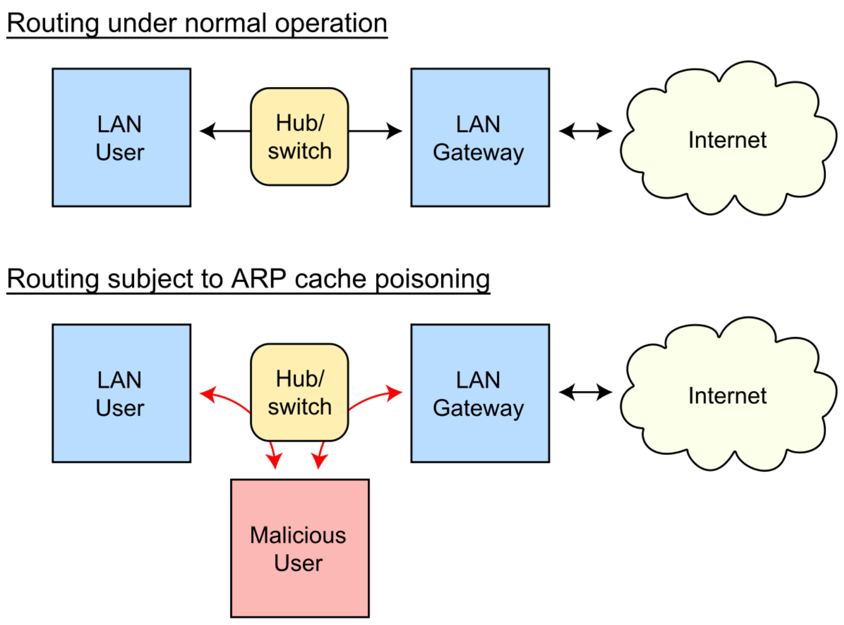


Figure 1: Overview of ARP poisoning: the attacker positions themselves between the victim and the gateway.

2.2 DNS Spoofing

The Domain Name System (DNS) translates domain names (e.g., `example.com`) into IP addresses. In traditional DNS spoofing attacks, the attacker tries to respond faster than the legitimate DNS server in a race condition. If the attacker's forged response arrives first, the victim connects to the wrong IP address, usually a server controlled by the attacker. This can lead to serious consequences, such as credential theft, session hijacking, or user redirection to malicious websites, as discussed in [1].

In this project, DNS spoofing is used in combination with ARP poisoning. Because the attacker is already in a MITM position, they can intercept DNS queries directly and block the victim's legitimate DNS requests. This removes the need to win a race and ensures that the spoofed DNS response is accepted since the attacker impersonates himself as the gateway.

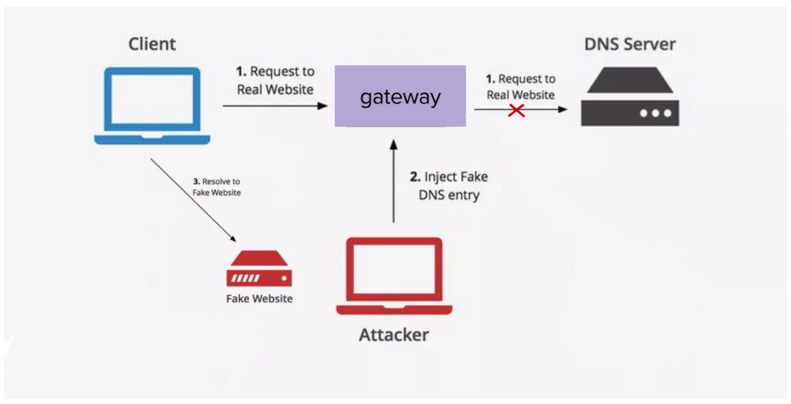

Client
1. Request to Real Website
gateway
1. Request to Real Website
DNS Server
3. Resolve to Fake Website
2. Inject Fake DNS entry
Fake Website
Attacker

Figure 2: DNS spoofing after ARP poisoning: the attacker intercepts the query and returns a forged response.

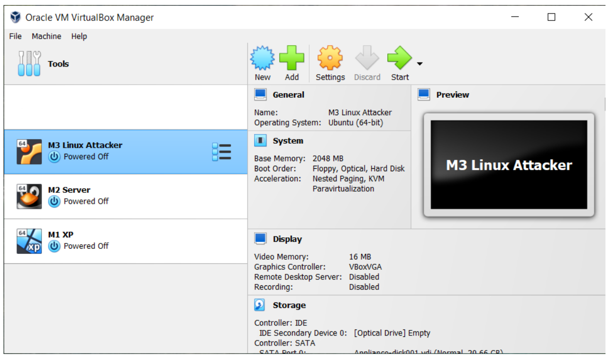
3 Technical Setup

To safely develop and test our attack tool, we created a virtual lab environment using VirtualBox. All virtual machines (VMs) were connected to the same virtual local network to simulate real-world conditions.

We used the following three machines in our setup:

- **Attacker VM:** Runs Ubuntu 22.04. It executes our Python scripts and hosts the spoofing tool.
- **Victim VM:** Simulates a regular user, for example by browsing websites or sending DNS queries.
- **Server VM:** A server the victim can communicates with.

All machines were configured on a shared LAN using the VirtualBox "Internal Network" mode, allowing full packet visibility and control.



M3 Linux Attacker — Powered Off
M2 Server — Powered Off
M1 XP — Powered Off

```
General
Name:              M3 Linux Attacker
Operating System:  Ubuntu (64-bit)

System
Base Memory:   2048 MB
Boot Order:    Floppy, Optical, Hard Disk
Acceleration:  Nested Paging, KVM
               Paravirtualization

Display
Video Memory:           16 MB
Graphics Controller:    VBoxVGA
Remote Desktop Server:  Disabled
Recording:              Disabled

Storage
Controller: IDE
  IDE Secondary Device 0:  [Optical Drive] Empty
Controller: SATA
```

Figure 3: Virtual machines setup

4 Attack Analysis

This section shows how ARP poisoning and DNS spoofing work through practical examples. Each case explains how the attack happens, what changes in the network, and what the attacker can do as a result.

4.1 Scenario 1: ARP Poisoning in a Local Network

In a shared network (e.g., office or campus Wi-Fi), the devices are connected in the subnet `192.168.1.0/24`. Three devices are involved:

- Alice's laptop: IP `192.168.1.20`, MAC `BB:BB:BB:BB:BB:BB`
- The default gateway: IP `192.168.1.1`, MAC `AA:AA:AA:AA:AA:AA`
- Attacker's device: IP `192.168.1.99`, MAC `CC:CC:CC:CC:CC:CC`

Under normal conditions, ARP tables look as follows:

Before attack:

- Alice's ARP table: `192.168.1.1 → AA:AA:AA:AA:AA:AA`
- Gateway's ARP table: `192.168.1.20 → BB:BB:BB:BB:BB:BB`

The attacker then sends fake ARP replies to both Alice and the gateway, tricking them into thinking the wrong MAC address belongs to the other.

After ARP poisoning:

- Alice's ARP table: `192.168.1.1 → CC:CC:CC:CC:CC:CC`
- Gateway's ARP table: `192.168.1.20 → CC:CC:CC:CC:CC:CC`

This places the attacker in the middle of the communication path. Packets sent from Alice to the internet, and responses from the internet back to Alice, now flow through the attacker's machine.

Result: This establishes a Man-in-the-Middle (MITM) position. The attacker can forward traffic transparently, monitor it, or selectively block or alter packets. This forms the foundation for more advanced attacks such as DNS spoofing.

4.2 Scenario 2: DNS Spoofing via Man-in-the-Middle

Once the attacker is in a MITM position through ARP spoofing, they can see Alice's traffic, including DNS requests to resolve website names.

Suppose Alice tries to visit `www.software-updates.com`, which normally points to IP `203.0.113.10` and the attacker creates a false website with IP `192.168.1.200`. The attacker blocks this request and instead sends a fake DNS reply:

Expected DNS Resolution:

- `www.software-updates.com → 203.0.113.10`

Forged DNS Resolution (after spoofing):

- `www.software-updates.com → 192.168.1.200`

Alice's browser is redirected to the fake website hosted by the attacker. It looks like a legitimate update page but serves a malware executable. Because the DNS resolution was spoofed, Alice sees the correct domain in the address bar, and might unknowingly download and run the file.

Result: Even though the victim typed a correct and trusted website address, DNS spoofing silently redirected the request to a fake destination controlled by the attacker. For example, the attacker could deliver a malicious file (like a fake software update) that installs malware when opened. In more advanced setups, this redirection can be combined with SSL stripping or fake certificates to make phishing attacks possible, even on HTTPS websites. This allows the attacker to not only deliver malware, but also steal login credentials or other sensitive data if encryption is bypassed or users ignore warnings.

5 Attack Engineering

This section describes how the tool is built to perform ARP and DNS spoofing attacks. It uses Python with `scapy` for crafting packets and `NetfilterQueue` to intercept and modify traffic in real time. The tool is modular and allows flexible configuration for targeting, spoofing, and clean shutdown.

Code Structure Overview

The project is split into modular Python scripts:

- `main.py` : Launches the tool, manages user input and threading.
- `network_utils.py` : Handles interface selection and network scanning.
- `arp_spoof.py` : Implements ARP spoofing and restoration logic.
- `dns_spoof.py` : Intercepts and forges DNS packets via NetfilterQueue.
- `logs/` : Stores spoofed DNS query logs.

5.1 Network Discovery Phase

Before launching an attack, the user is first prompted to select a network interface from the list of active devices. Once selected, the tool performs a local network scan using ARP requests on the corresponding subnet (e.g., `192.168.1.0/24`). All reachable hosts respond, allowing the tool to build a mapping of active devices (IP and MAC addresses).

After the scan is complete, the user is guided to:

- Select the default gateway (router),
- Choose one or more victims from the discovered devices.

This phase ensures that the attacker has full control over the network mapping and attack targets before starting the ARP or DNS spoofing process.

5.2 ARP Spoofing Mechanism

Once the attacker selects the network interface, the gateway, and one or more victims, the ARP spoofing process is launched in a dedicated thread via the `start_arp_spoofing()` function. This function performs continuous poisoning of the ARP tables of both the victims and the gateway by crafting and sending forged ARP replies.

Spoofing Interval Configuration You can change how often spoofing happens using the `interval` setting. The user can pick fast (1 second), normal (2 seconds), slow (10 seconds), or set a custom time. This interval controls how often ARP replies are sent again. Shorter times make the attack more stable, it keeps working even if devices refresh their ARP cache, but they create more network traffic, which is easier to notice. Longer times make the attack harder to detect but increase the chance that devices recover and stop the attack from working.

Packet Construction Each ARP reply is built by the `build_arp_packet()` function, which uses Scapy's `Ether()` and `ARP()` layers to construct a spoofed Ethernet frame:

```
ethernet = Ether(src=src_mac, dst=dst_mac)
arp_layer = ARP(op=2, psrc=src_ip, hwsrc=src_mac, pdst=dst_ip, hwdst=
    dst_mac)
return ethernet / arp_layer
```

Listing 1: Constructing a Spoofed ARP Reply Packet

Here, `op=2` specifies an ARP reply. For each victim-gateway pair, two packets are sent every round:

- One to the victim: claiming the gateway IP is at the attacker's (or broadcast) MAC.
- One to the gateway: claiming the victim IP is at the attacker's (or broadcast) MAC.

These packets are sent using Scapy's `sendp()` function, as seen in the inner `send_round()` method.

Spoofing Modes The tool supports two ARP spoofing strategies:

- **Normal MITM Mode:** The attacker uses their real MAC address in ARP replies. This allows full bidirectional interception and forwarding of traffic, placing the attacker transparently between the victim and gateway.
- **Anonymous SMITM Mode:** As described in this paper [2], this mode uses the broadcast MAC address (`ff:ff:ff:ff:ff:ff`) instead of the attacker's real one. This makes the poisoned hosts unable to send directed replies to the attacker, breaking the channel and preventing active interference.

  However, any packets sent by the victim to the broadcast MAC are visible to all nodes on the LAN, including the attacker. This allows passive sniffing of traffic while minimizing traceability. Critically, even if a victim detects the poisoning and inspects their ARP cache, the attacker's MAC never appears, making attribution difficult.

Enabling Forwarding and Avoiding Disruption To allow packets to flow through the attacker in MITM mode, system forwarding is enabled and ICMP redirects are disabled using `sysctl`:

```
sysctl -w net.ipv4.ip_forward=1
sysctl -w net.ipv4.conf.all.rp_filter=0
sysctl -w net.ipv4.conf.all.send_redirects=0
```

These steps prevent Linux from interfering with the spoofed flows or detecting inconsistencies.

Attack Termination and ARP Table Restoration Upon interruption or completion, the spoofing thread exits and triggers the `restore_all_arp()` function, which repairs ARP tables of both victims and gateway. This is done by sending multiple correct ARP replies (5 times by default) to both parties:

```
sendp(correct_to_victim, iface=iface)
sendp(correct_to_gateway, iface=iface)
```

Listing 2: Restoring ARP Tables After Attack

This cleanup phase helps restore network stability and minimizes the risk of user suspicion.

5.3 DNS Spoofing Mechanism

When DNS spoofing is enabled, the attacker leverages their MITM position (established via ARP poisoning) to intercept DNS traffic traversing their machine. This interception is achieved using `iptables` rules (see `setup_nfqueue()`) that redirect all DNS packets, both requests and replies (UDP port 53), into a NetfilterQueue:

```
iptables -I FORWARD -p udp --dport 53 -j NFQUEUE --queue-num 1
iptables -I FORWARD -p udp --sport 53 -j NFQUEUE --queue-num 1
```

NetfilterQueue allows userspace processing of packets within Python. Each packet entering the queue is inspected in the `handle_packet()` function. DNS queries are identified using:

```
if pkt.haslayer(DNSQR) and pkt[DNS].qr == 0:
```

Listing 3: Checking for a DNS Query Packet

This checks for a DNS query (`QR = 0`), which means the packet is a request. If the requested domain matches an entry in the user-defined `domain_map`, the tool crafts a spoofed DNS reply in `spoof_dns_packet()`:

- The spoofed response copies the transaction ID and question section from the original query.
- A forged `DNSRR` answer is created to map the queried domain to a fake IP address. The response flags are set accordingly to indicate a valid answer, based on standard DNS header structure as described in [3].
- IP and UDP headers are reversed (`IP.src` $\leftrightarrow$ `IP.dst`, `UDP.sport` $\leftrightarrow$ `UDP.dport`) to correctly target the response.

If the domain is not in the list, the packet is forwarded untouched via:

```
nf_packet.accept()
```

Listing 4: Allowing Unmodified Packets to Continue

The spoofing logic ensures that only selected DNS queries are manipulated, preserving all unrelated network activity. Additionally, ICMP packets of type 3/code 3 (port unreachable) are dropped to prevent DNS clients from detecting failed responses:

```
if pkt.haslayer(ICMP) and pkt[ICMP].type == 3 and pkt[ICMP].code == 3:
    nf_packet.drop()
```

Listing 5: Dropping ICMP Destination Unreachable Messages

To facilitate control and reliability, the tool includes additional engineering elements:

- **Custom Domain Injection:** Users can input multiple domains and corresponding fake IPs, enabling flexible targeting.
- **Live Logging:** Spoofed and ignored DNS queries are timestamped and stored for later inspection.
- **Graceful Teardown:** On interruption, NetfilterQueue is unbound and all `iptables` rules are removed using `teardown_nfqueue()` to restore normal network behavior and ARP tables are restored too.

6 Conclusion and Future Work

This project is a modular and interactive tool for performing man-in-the-middle attacks using ARP and DNS spoofing in a local network. It includes features like network discovery, customizable poisoning intervals, different spoofing modes, and DNS redirection using `NetfilterQueue` to inspect and modify packets. The user can choose specific devices to target, redirect domain name requests, and keep logs of spoofed DNS traffic, all while making sure the attack ends cleanly by restoring the ARP tables to their original state.

ARP spoofing is flexible, especially with the *Anonymous SMITM* mode, which hides the attacker's MAC from ARP tables. On the DNS side, the spoofing is reliable through custom domain-to-IP mapping, using precise packet crafting and a stable queue management system to control traffic in real time.

Future improvements could focus on several key areas:

- **Enhanced control in Anonymous SMITM mode:** Currently, this mode only allows passive sniffing. One possible extension would be to monitor incoming broadcast packets more closely and selectively block or allow them. Adding logic to discard certain packets could allow partial control while being anonymous.
- **IPv6 support:** Right now, the tool only works on IPv4 networks using ARP spoofing. In order to make it work with IPv6, we would need to add support for a similar protocol called Neighbor Discovery Protocol (NDP). If this is done, the tool could be used in more modern networks where IPv6 is used instead of IPv4.
- **Detection avoidance and anti-analysis:** Incorporating random delays in poisoning intervals or mimicking legitimate ARP traffic behavior could reduce the likelihood of detection by IDS or security tools.
- **GUI or Web Interface:** While the current tool is CLI-driven, adding a graphical or web-based frontend would improve usability and help visualize network topologies and attack status.

Overall, this tool provides a solid and extensible foundation for exploring MITM attack techniques and understanding LAN-level security weaknesses.

References

[1] T. R. Kukutla, “A Deep Dive into DNS Spoofing and Security Measures,” *Medium*, Dec. 2023. [Online]. Available: `https://www.researchgate.net/publication/376271471_A_Deep_Dive_into_DNS_Spoofing_and_Security_Measures`

[2] N. R. Samineni, F. A. Barbhuiya and S. Nandi, “Stealth and semi-stealth MITM attacks, detection and defense in IPv4 networks,” in *Proc. 2nd IEEE Int. Conf. on Parallel, Distributed and Grid Computing (PDGC)*, Solan, India, 2012, pp. 364–367, doi: 10.1109/PDGC.2012.6449847.

[3] L. Li, Y. Liu, K. Lu, C. Li and Z. Zhang, ”Research on inconsistent consecutive DNS responses from DNS resolvers,” in *Proc. 2nd Asia-Pacific Conf. on Communications Technology and Computer Science (ACCTCS)*, Shenyang, China, 2022, pp. 86–91. doi: 10.1109/ACCTCS53867.2022.00025