DAT505 - Assignment 2

Andreas Pedersen

November 5, 2025

Link to GitHub Repository containing all scripts, pcap_files/ and evidence/

1 Setup

In this lab, three virtual machines are connected to an isolated network. The environment and key configuration details are listed below.

1.1 Hardware / VM resources

Role	VM specification
Attacker (Kali)	2 vCPU, 2 GB RAM
Victim (Ubuntu Desktop)	2 vCPU, 2 GB RAM
Gateway / Server (Ubuntu Server)	2 vCPU, 2 GB RAM

IP addresses used in the lab:

- Attacker: `192.168.128.2`
- Victim: `192.168.128.3`
- Gateway/Server: `192.168.128.4`

1.2 Software installed

- Attacker: Kali Linux with `Python3`, `Scapy`, `Wireshark/tcpdump`, and `dnsmasq`.
- Victim: Ubuntu Desktop with a browser and common client tools.
- Gateway/Server: Ubuntu Server with `nginx` and `dnsmasq`.

2 Methodology

This section describes the steps taken to perform the ARP spoofing MitM, capture traffic, and selectively spoof DNS responses.

2.1 Baseline verification

Before any tests:

1. Verified isolated network and that VMs had no default route to external networks: `ip route show` output on each VM showed only the local `192.168.128.0/24` route.
2. Confirmed connectivity between VMs with `ping -c 3 <ip>`.
3. Baseline capture: Using Wireshark (then save as `pcaps`).
4. Baseline browser fetch from victim: `curl http://192.168.128.4/` or `http://www.server.lab`.

2.2 Task 1 - ARP Spoofing tool

`arp_spoof.py` was implemented with the following features:

- CLI arguments: `-v` (victim IP), `-g` (gateway IP), `-i` (interface).
- Optional enable of IP forwarding on the attacker while the spoof runs.
- Graceful cleanup on exit: sends correct gratuitous ARP to restore tables and restores the original IP forwarding value.
- Verbose mode.

Example usage:

```
sudo python3 arp_spoof.py -v 192.168.128.3 -g 192.168.128.4 -i eth0
--enable-forward --verbose
```

2.2.1 ARP poisoning technique

The tool repeatedly sends ARP "is-at" replies:

- to the victim: "gateway IP is at attacker MAC"
- to the gateway: "victim IP is at attacker MAC"

Packets are directed using `Ether(dst=<target-mac>)/ARP(...)`. If real MACs cannot be resolved, the tool falls back to broadcasting a forged ARP, allowing the host to accept the entry still.

2.3 Task 2 - Traffic capture and analysis

Captured packet traffic during ARP spoofing to produce baseline and during-attack evidence. Captures were saved as a PCAP for analysis. A small parser, `traffic_interceptor.py`, was used to extract short, human-readable artifacts from the saved PCAP.
Example usage:

```
sudo python3 traffic_interceptor.py capture.pcap --outdir ./results
```

This produces the CSV files:

- `dns_queries.csv` (timestamp, source IP, query name)
- `http_urls.csv` (timestamp, source IP, URL)
- `top_talkers.csv` (IP, packet count)
- `protocol_counts.csv` (protocol, count)

The PCAP and CSV extracts are stored under `pcap_files/` and `evidence/csv_files/`, respectively. The two annotated Wireshark screenshots are included in `evidence/screenshots/` and shown later in Sec. 3.

2.4 Task 3 - DNS spoofing

The script `dns_spoof.py` was implemented to:

- inspect incoming DNS queries from the victim,
- match domain against a configuration file listing domains to spoof (blacklist or selective list),
- craft and reply with a DNS response that preserves transaction ID and flags, delivering the attacker-controlled IP (e.g., `192.168.128.2`).

Example usage:

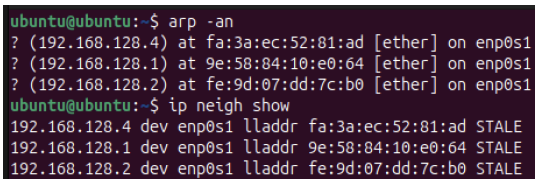
```
sudo python3 dns_spoof.py --iface eth0 --targets targets.txt
--attacker-ip 192.168.128.2 --mode whitelist --forward 192.168.128.4
```

For demonstration, the domain `server.lab` was configured in the text file, and replies were directed to the attacker's web server. The gateway's `dnsmasq` was configured to answer internal queries in the baseline test; during the spoof experiment, the attacker replied directly to victim queries for the target domain.

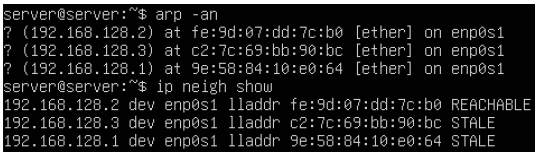
3 Results

3.1 ARP spoofing

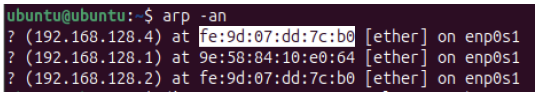
- **ARP table before attack:** captured via `arp -an` on victim and gateway, see Fig 1a and 1b.
- **ARP table during attack:** Figure 1c and 1d shows arp table after starting `arp_spoof.py`. The ARP entry for the gateway IP on the victim displays the attacker's MAC, and similarly, the victim's IP on the gateway shows the attacker's MAC (evidence).
- **PCAP evidence:** `pcap_files/task1_arp_spoof.pcap` contains packets showing the victim's traffic arriving at the attacker.



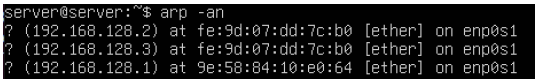
(a) Victim before poisoning.



(b) Gateway before poisoning.



(c) Victim after poisoning.



(d) Gateway after poisoning.

Figure 1: ARP tables before and after poisoning on both the victim and the gateway.

3.2 Traffic analysis summary

Using `traffic_interceptor.py`, a set of short CSV outputs was produced from the PCAP:

- `evidence/csv_files/dns_queries.csv`: all DNS queries observed with timestamps.
- `evidence/csv_files/http_urls.csv`: extracted HTTP Host:URI pairs.
- `evidence/csv_files/top_talkers.csv`: IPs ordered by packet count.
- `evidence/csv_files/protocol_counts.csv`: Counts of protocols.

Two annotated Wireshark screenshots:

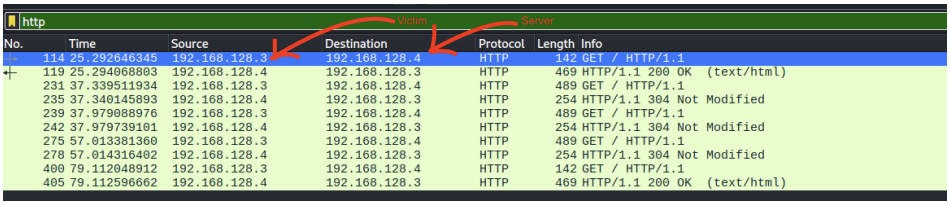


Figure 2: Captured on attacker: Victim → Server: HTTP GET

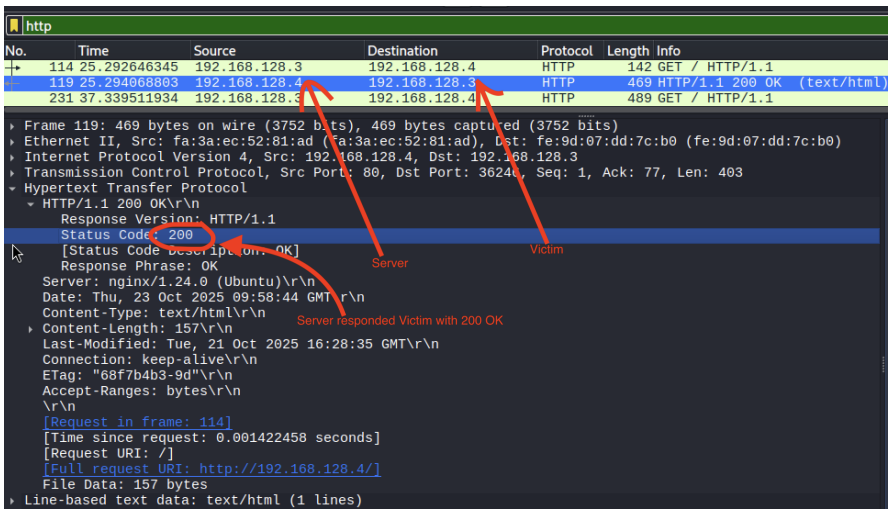


Figure 3: Captured on attacker: Server → Victim: HTTP/1.1 200 OK

3.3 DNS spoofing

- PCAP showing DNS response crafted by the attacker: `pcap_files/task3_dns_spoof.pcap`.
- Should have had a screenshot of the victim showing the redirected attacker page, but due to unknown issues (might be caching), the browser would not display it; however, the `dig` command and PCAP did in fact show a **successful DNS spoof** as seen in Fig 4 and 5 respectively.

```
ubuntu@ubuntu:~$ dig @192.168.128.4 server.lab

; <<>> DiG 9.18.39-0ubuntu0.24.04.1-Ubuntu <<>> @192.168.128.4 server.lab
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11460
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;server.lab.                    IN      A

;; ANSWER SECTION:
server.lab.             0       IN      A       192.168.128.4

;; Query time: 1 msec
;; SERVER: 192.168.128.4#53(192.168.128.4) (UDP)
;; WHEN: Thu Oct 23 16:56:49 UTC 2025
;; MSG SIZE  rcvd: 55

ubuntu@ubuntu:~$ dig @192.168.128.4 server.lab

; <<>> DiG 9.18.39-0ubuntu0.24.04.1-Ubuntu <<>> @192.168.128.4 server.lab
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26212
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;server.lab.                    IN      A

;; ANSWER SECTION:
server.lab.             300     IN      A       192.168.128.2

;; Query time: 41 msec
;; SERVER: 192.168.128.4#53(192.168.128.4) (UDP)
;; WHEN: Thu Oct 23 16:57:34 UTC 2025
;; MSG SIZE  rcvd: 54
```

Figure 4: Testing DNS resolution (using `dig`) shows before and after spoof, and that the IP changes to the attacker.

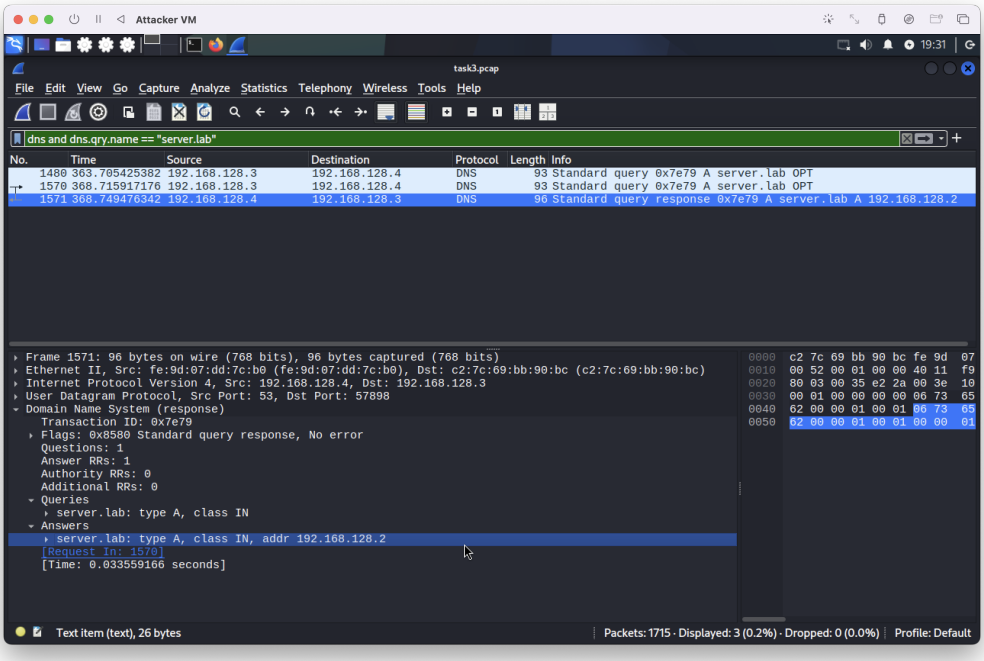


Figure 5: PCAP showing spoofed response.

4 Mitigation Ideas

In this section, we look at mitigation ideas to prevent or reduce the impact of ARP and DNS spoofing.

4.1 Local/network configuration

- **Static ARP entries** on critical hosts for gateway/router MAC: prevents learning of forged ARP entries (administratively heavy at scale).
- **Dynamic ARP Inspection (DAI)** on managed switches: switches validate ARP packets against a trusted database (DHCP snooping table).
- **Segmentation / VLANs**: separate sensitive hosts from general client VLANs to reduce attack surface.

4.2 Protocol and host-level defenses

- **Use DNS-over-HTTPS / DNS-over-TLS** combined with authentication to prevent on-path manipulation of DNS responses.
- **Encrypted protocols and HSTS:** ensure HTTPS with HSTS is used; modern browsers will reject HTTP fallback, mitigating SSLStrip style downgrades.
- **Mutual/Authenticated DHCP/DNS services:** use secure DNS setups and avoid trusting unauthenticated on-path DNS.

4.3 Operational recommendations

- Maintain up-to-date host-based intrusion detection and ARP anomaly monitoring.
- Monitor ARP table changes and alert on rapid/large ARP changes.
- Use centralized logging for DNS and network flows and apply analytics to detect suspicious spikes or anomalies.

5 Ethics

All experiments in this lab were conducted on an isolated virtual network under explicit authorization for coursework. The techniques presented are powerful and can cause service disruption and privacy violations if used outside controlled environments. It is ethically and legally imperative to:

- Obtain explicit authorization before testing networks or systems not owned by you.
- Avoid exposing test traffic to the wider Internet.
- Remove or redact any personally-identifiable information (PII) before publishing evidence.
- Use the knowledge gained to strengthen defenses and for academic/defensive purposes only.