5300 PROJECT PROPOSAL

Detecting and Visualizing ARP Spoofing in IPv4 Networks

Mubassir Serneabat Sudipto

Fall 2025

Project Description

This project delivers ARPGuard, a lightweight programming + visualization tool that demonstrates Address Resolution Protocol (ARP) spoofing attacks and real-time defense on small IPv4 LANs. The system passively monitors ARP traffic, builds a MAC↔IP ground-truth map, flags anomalies (e.g., duplicate IP addresses bound to multiple MACs, sudden MAC flips), and provides a browser dashboard to visualize events, timelines, and mitigation tips. The demo includes curated .pcap traces and a live capture mode using a virtual lab topology.

Problem Relevance: ARP spoofing enables traffic interception (MITM), session hijacking, and credential theft in flat networks and Wi-Fi segments. Despite TLS adoption, ARP-based redirection remains a common foothold for adversaries in enterprise and campus networks, directly tying into the TCP/IP model (link/network layers) and course topics on network security and defenses.

Category Alignment (**Programming and Visualization)**:

- A Python/Scapy sensor and anomaly engine.
- A web UI (Flask + D3.js) that visualizes ARP state and alerts.
- A short "Reteaching" walkthrough that connects observations to the TCP/IP model and secure network practices.

Target Audience

Undergraduate/graduate students in introductory networking/network security courses; SOC interns and lab TAs who need a practical, visual understanding of L2/L3 attack surfaces.

Audience Skill Level

Basic command-line familiarity, Wireshark fundamentals (filters, following streams), and an understanding of the TCP/IP stack (ARP, IP, TCP/UDP). No advanced programming background is required to use the tool; intermediate Python helps extend it.

LEARNING OBJECTIVES

After interacting with ARPGuard, a participant will be able to:

Objective 1 - Identify ARP spoofing indicators in a packet trace and live capture and justify the detection using at least two observable artifacts (e.g., gratuitous ARP storms, MAC churn), which is measured by a 6-item, ≥ 85% to pass.

Objective 2 - Execute a safe, local lab that reproduces ARP spoofing and capture evidence with Wireshark/Zeek, which is measured by a lab checklist with screen captures.

Objective 3 - Deploy ARPGuard on a test subnet and detect a synthetic attack with ≤ 5s time-to-first alert under default settings.

Objective 4 - Recommend at least three mitigations mapped to the TCP/IP model and network hardening practices (e.g., static ARP for critical hosts, DHCP snooping/DAI, VLAN segmentation).

Objective 5 - Explain how L2 compromise can enable higher-layer attacks (session hijack/credential theft), linking observations to defense-in-depth controls.

RELEVANCE OF TOPIC

ARP operates between the link and network layers (L2/L3 boundary) to resolve IP→MAC bindings. Its lack of authentication enables cache poisoning, enabling on-path manipulation before TCP handshakes or TLS occur. By instrumenting ARP dynamics and correlating IP/MAC changes, ARPGuard trains students to reason how lower-layer weaknesses cascade into transport/application compromises, directly reinforcing this course's TCP/IP, layering, and network security learning goals.

RESEARCH AND SUPPORTING MATERIALS

The project plan is based on well-established networking sources and recent best practices:

[1] D. C. Plummer, "An Ethernet Address Resolution Protocol," RFC 826, IETF, Nov. 1982. (Ground-truth protocol behavior and message formats; informs parser and anomaly rules.)

[2] NIST, "Guide to Computer Security Log Management," SP 800-92, 2006; and NIST "Security Architecture and Engineering" references for network monitoring patterns. (Maps detections to measurable events and logging baselines.)

[3] Center for Internet Security (CIS), CIS Controls v8 (Controls 12 & 13 on network monitoring/secure configuration). (Defense-in-depth controls to translate detections into mitigations.)

[4] Wireshark User's Guide and display filter reference (Practical filtering for ARP anomalies).

These references support protocol-accurate parsing, alert logic, and hardening guidance. As required, two or more networking sources will be formally cited in IEEE style in the final write-up and in-app help panel.

Required resources

Software: Python 3.11+, Scapy, Flask, Jinja2, D3.js (or Chart.js), Zeek (Optional), Wireshark, Docker (Optional for packaging).

Environment: MacOS or Linux VM with bridged adapter; sample .pcap traces; optional two-VM lab (attacker/victim) using Kali Linux/Ubuntu.

Data: Curated ARP spoofing captures; benign baseline captures.

Docs & media: IEEE-style references; screenshots for the reteaching guide.

File Sharing: Shared repo and task board per Canvas partner guidance; expanded scope to include Zeek integration and automated tests.

References

[1] D. C. Plummer, "An Ethernet Address Resolution Protocol," RFC 826, IETF, Nov. 1982.

[2] K. Kent and M. Souppaya, "Guide to Computer Security Log Management," NIST SP 800-92, Sept. 2006.

[3] Center for Internet Security, "CIS Controls v8," 2021.

[4] Wireshark Foundation, "Wireshark User's Guide," latest ed.

Project Timeline

Week	Tasks	Deliverables & Exit Criteria
1	Requirements, lab topology design; build packet parser & baseline MAC↔IP map; collect benign traces	Parser unit tests (≥ 80% coverage); 2 benign .pcap sets; draft UI mock
2	Implement anomaly rules (dup-IP, MAC-flip, flood/rate); real-time capture; alert queue	Detect synthetic attack in lab with ≤ 5 s alert latency; trace + screenshots
3	Flask UI (events timeline, host graph, "why flagged?" explainer); export reports	Running dashboard; JSON/CSV export; user guide v1
4	Reteaching module (step-by-step lab); hardening guide; polish; usability check	Final demo video (≤ 5 min); PDF lab handout; release zip

Progress measurement: Unit tests, Reproducible Lab Checklist, and a Scoring Rubric: Detection Latency, True-Positive on Provided Attack Pcaps, Clarity of Reteaching Assets, and Citation Quality.

Task Dependencies & Potential Issues

Dependencies: UI depends on a stable event schema; anomaly scoring depends on the baseline learner built in Week

Risks: Limited ARP traffic in some VMs (Workaround: Generate with arpspoof/Scapy); permission constraints for live capture (Use sudo/pcap group).

Mitigations: Fall back to curated PCAPs; provide Docker image with capabilities; include "offline mode."

Expected Outcomes, Evaluation, and Assessment

Deliverables:

- ARPGuard Tool (Source + Dockerfile).
- Dashboard UI.
- Curated PCAPs.
- Lab walkthrough PDF.
- Short demo video.

Evaluation:

- **Functional:** Detects three attack patterns on supplied PCAPs (100% TP, ≤ 5s first alert).
- **Pedagogical:** Student completes quiz ≥ 85% and produces a mitigation list mapped to TCP/IP layers.
- **Documentation:** IEEE-style references; clear "why flagged" explanations in UI.

Success indicators:

- TP/FP metrics on test traces.
- User survey (SUS ≥ 70).
- Reproducible lab run in < 20 minutes.

Personal Learning Goals

I aim to deepen my applied understanding of how layer-2 weaknesses propagate to higher-layer risks, strengthen hands-on skills in packet-level analysis and security visualization, and practice turning detections into actionable guidance aligned with common frameworks (NIST/CIS). I will also improve software engineering practices (Unit Tests, Minimal Docker Packaging) and communication of complex network behaviors through clear visuals and reteaching artifacts.

Additional Information

If I am approved to work on this project and allocated a partner, I will expand the scope to include Zeek scripting for ARP logs, pcap-replay automation, and a richer mitigation playbook. Tasks will be divided so each member owns a security and software component, per course rules (Details would be listed here and in the repo README).