5300 PROJECT FINAL REFLECTION

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.

Project Impact

ARPGuard is structured to close a common learning gap in network security: many learners can describe ARP spoofing conceptually, but struggle to recognize it in real traffic and to explain why specific protocol fields and host-table changes matter. By combining (1) an offline .pcap-driven workflow, (2) a clearly scoped anomaly model based on expected ARP behavior, and (3) a simple web dashboard that makes the attack "visible," the project turns ARP spoofing into an observable, testable phenomenon rather than an abstract vulnerability. The implementation and explanations follow protocol-accurate ARP behavior as defined in RFC 826 and related operational patterns such as conflict detection/gratuitous ARP behavior. The lab also reinforces defensive thinking by requiring learners to connect detections to mitigations and monitoring practices used in real environments. [1], [4], [5].

Expected Outcomes, Evaluation, and Assessment

Expected outcomes include: a working ARPGuard prototype capable of parsing ARP traffic from .pcap files and generating anomaly events; a web dashboard that visualizes host mappings and alerts; a reteaching lab module with step-by-step instructions and screenshots; facilitator notes to enable repeatable delivery; and a quiz with an answer key.

Evaluation will be based on functional correctness (ARP parsing and detection rules behave as expected), reproducibility (a user can install dependencies and run the project with provided captures), and instructional clarity (materials are complete, well-paced, and aligned with learning objectives). Assessment metrics include: whether ARPGuard flags the provided attack .pcap with meaningful alerts, whether benign traffic avoids false positives, the clarity and correctness of quiz answers and rationales, and whether documentation includes proper citations and setup guidance.

Project Evaluation and Outcomes

In the final implementation, ARPGuard's detection engine is intentionally designed to log security-relevant anomalies, not general ARP "activity." As a result, when the engine is executed against benign_arp.pcap, it may correctly produce 0 events, because no IP↔MAC conflicts, MAC-flips, or suspicious reply patterns are present in the capture. This behavior is consistent with the project's evaluation goal of keeping false positives low during baseline traffic analysis.

For the controlled attack capture (arp_spoof_attack.pcap), the expected outcome is that ARPGuard generates at least one alert corresponding to an abnormal IP↔MAC claim (e.g., MAC-flip for a protected IP, duplicate IP announcements, or unsolicited reply bursts), and the dashboard renders the alert(s) in the event timeline and host-mapping view.

To address the graded feedback from the Research & Content Development submission (requesting "correct answers and justification"), the final quiz and answer key are written so that each answer includes a brief rationale explaining why it is correct, explicitly linking the observed evidence (ARP table change, ARP reply fields, repeated claims, etc.) to ARP protocol behavior and the alert logic. This ensures the assessment artifact demonstrates correctness and not only completion. [1], [4]

(Insert your observed counts from your own run logs here if you want them explicitly stated:

- benign_arp.pcap alerts: 0
- arp_spoof_attack.pcap alerts: 2

Personal Impact

The project topic was chosen because ARP spoofing provides a clear, high-impact example of how weaknesses at the link layer can undermine assumptions made at higher layers. The author's objective was to create a deliverable that is both technically functional and instructionally strong: a learner should be able to run the tool, see evidence in a capture, and explain the security consequences in plain language. Building ARPGuard also aligned with the author's broader interest in practical security engineering—where detection logic, logging discipline, and user-facing documentation must all work together to produce a usable security outcome rather than an isolated script.

Personal Learning Goals

The author's primary learning goal is to deepen his understanding of link-layer protocol behavior and how low-level network protocols can be exploited to enable higher-layer attacks such as man-in-the-middle interception. He aims to strengthen his ability to translate protocol specifications and defensive guidance into a working detection tool that produces actionable, explainable alerts rather than raw packet dumps.

A second goal is to improve his ability to design instructional security materials that are clear, reproducible, and aligned with learning objectives, including creating assessments that test both conceptual understanding and practical interpretation of evidence. Finally, he intends to improve software engineering habits for security tooling, including maintainable project structure, reliable environment setup, and user documentation that makes the project easy for others to install and run.

Skills and Knowledge Gained

This project strengthened the author's applied understanding of ARP at the link layer, including which ARP fields are most meaningful for detecting cache poisoning behavior and how host ARP tables reflect on-the-wire claims. Translating RFC-aligned behavior into detection logic required careful thinking about what constitutes an "anomaly" versus normal operational behavior (e.g., legitimate ARP refresh, gratuitous ARP in normal environments), and reinforced the importance of minimizing false positives while still capturing security-relevant deviations. [1], [5]

From a tooling perspective, the author gained experience building a small security pipeline end-to-end: parsing offline packet captures, constructing a baseline mapping model, emitting structured event records, and presenting results in a lightweight dashboard. On the instructional side, he improved his ability to design a lab flow that escalates from basic observation to interpretation to mitigation, and to write assessment answers with short, defensible justifications (as required by grading feedback).

Lessons Learned

A key lesson was that a well-designed security detector does not necessarily produce output on benign traffic—silence can be a correct result when the system is tuned to surface anomalies only. This drove a documentation improvement: the final materials explicitly explain that 0 events on benign_arp.pcap can be expected and is not automatically an error. Another lesson was that grading expectations for assessment artifacts go beyond providing an "answer key"; they require correctness plus justification, which must be written clearly enough that a TA can understand the reasoning without inferring intent. Finally, the project reinforced the practical importance of packaging and accessibility—deliverables must be easy to run, easy to locate, and consistent with naming/format requirements to avoid preventable point deductions.

Future Impact

ARPGuard can be reused as a classroom module in future networking/security offerings because it is built around portable artifacts: curated captures, step-by-step instructions, and a small toolchain that runs on typical student environments. If extended in the future, the project could incorporate additional alert types (e.g., rate-based scanning indicators, per-host trust scoring), support live capture more robustly across OS environments, and expand the reteaching component to include comparisons with enterprise controls such as Dynamic ARP Inspection. The work also provides a foundation for broader "protocol-to-detection" projects where RFC behavior is translated into explainable alert logic, an approach that generalizes beyond ARP to other protocol misuse patterns.

Explanation of Submitted Materials

For the final submission, the author provides the required project materials in two forms to satisfy accessibility and rubric expectations:

1. Canvas-ready artifacts, including the final reflection PDF and .txt copies of the required source code (per course instructions).

2. A public GitHub repository hosting auxiliary materials (full document versions, PCAPs, figures/screenshots, and a ZIP-friendly project structure) so all supporting files remain accessible without bloating the PDF appendices.

Consistent with the author's chosen approach, the Appendices in this document are short outlines only, and the full versions of the corresponding documents are provided as separate files and in GitHub.

Project Content Deliverables

Item 1

- Item Description: Final learner lab handout guiding students through running ARPGuard, inspecting ARP traffic, and answering analysis questions.
- File name/path/link: ARPGuard_Lab_Handout.txt (Canvas upload) and docs/ARPGuard_Lab_Handout.pdf (GitHub).
- Relevant information: Student-facing; references PCAPs and explains expected results, including that benign traffic may produce 0 anomaly events.

Item 2

- Item Description: Facilitator notes with pacing guidance, troubleshooting, and discussion prompts.
- File name/path/link: ARPGuard_Facilitator_Notes.txt (Canvas upload) and docs/ARPGuard_Facilitator_Notes.pdf (GitHub).
- Relevant information: Instructor/TA-facing; includes common failure modes and expected outputs for benign vs. attack traces.

Item 3

- Item Description: Quiz and answer key with short rationales/justifications for each answer.
- File name/path/link: ARPGuard_Quiz_and_AnswerKey.txt (Canvas upload) and docs/ARPGuard_Quiz_and_AnswerKey.pdf (GitHub).
- Relevant information: Each answer includes a brief justification tied to protocol evidence and ARPGuard alert logic (per graded feedback).

Item 4

- Item Description: Project overview and mini user guide summarizing architecture, workflow, and how deliverables connect.
- File name/path/link: ARPGuard_Project_Overview.txt (Canvas upload) and docs/ARPGuard_Project_Overview.pdf (GitHub).
- Relevant information: Quick orientation document for reviewers and future reuse.

Item 5

- Item Description: Curated PCAP traces showing benign ARP behavior and controlled ARP spoofing behavior.
- File name/path/link: pcaps/benign_arp.pcap, pcaps/arp_spoof_attack.pcap (GitHub).
- Relevant information: Used by offline analysis mode and referenced throughout the lab handout; benign trace may yield 0 anomaly alerts by design.

Item 6

- Item Description: Figures/screenshots illustrating ARP tables, Wireshark filters/views, and ARPGuard alert outputs.
- File name/path/link: figures/ (GitHub)
- Relevant information: Used as visual support; included to reduce ambiguity for learners and to improve TA review clarity.

Source Code and READMEs

Item 1
- Item Description: Core ARPGuard detection engine (parsing, baseline map, anomaly event generation).
- File name/path/link: arpguard_core.txt (Canvas upload) and code/arpguard_core.py (GitHub).
- Relevant information: Contains docstrings and in-line comments; logic grounded in ARP protocol behavior. [1]

Item 2
- Item Description: Web dashboard for uploading PCAPs and viewing event timeline and host mappings.
- File name/path/link: arpguard_web_dashboard.txt (Canvas upload) and code/arpguard_web_dashboard.py (GitHub).
- Relevant information: Provides a visual learning aid; connects alerts to explanations and expected evidence.

Item 3
- Item Description: Lab tools and utilities (PCAP runner, helpers, optional synthetic traffic generation).
- File name/path/link: arpguard_lab_tools.txt (Canvas upload) and code/arpguard_lab_tools.py (GitHub).
- Relevant information: Used to reproduce lab steps and standardize how traces are evaluated.

Item 4
- Item Description: Installation and usage documentation (quick start, commands, structure, troubleshooting).
- File name/path/link: README_ARPGuard.txt (Canvas upload) and README.md (GitHub).
- Relevant information: Includes environment assumptions, run commands, and notes on expected outputs.

Item 5
- Item Description: Dependency pinning file for consistent setup.
- File name/path/link: requirements.txt (GitHub; optionally also uploaded to Canvas)
- Relevant information: Supports reproducibility and reduces environment-related grading issues.

Dependencies, Resources & Tools

To run ARPGuard and its associated lab materials, a user needs:
- Operating System: Linux or macOS host, or a Linux VM running on another OS.
- Python Environment: Python 3.11+ with pip available.
- Python Libraries:
  - scapy for packet parsing and offline PCAP processing.
  - flask and jinja2 for the web dashboard.
  - additional dependencies as listed in requirements.txt.
- Network Tools:
  - Wireshark for packet inspection and student-side analysis.
  - Optional: arpspoof (or similar) on a controlled lab network to generate attacks.
- Data Files:
  - Provided PCAP traces (benign and attack scenarios).

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.
[5] S. Cheshire, “IPv4 Address Conflict Detection,” RFC 5227, IETF, July 2008.

ADDITIONAL INFORMATION

Public GitHub Repository (auxiliary files, PCAPs, figures, full document versions):
https://github.com/msudipto/ARPGuard/tree/main

No personal login credentials are included in any submitted materials. The repository is intended to provide durable access to auxiliary artifacts while the Canvas upload provides the required reflection PDF and code-in-.txt format for grading.

Video/Audio: No video or audio files are required and made for this project submission, since the final deliverables are documentation, PCAP-based analysis, and source code.

Appendix A - Lab Handout Outline

ARPGuard Lab Handout

Detecting ARP Spoofing with ARPGuard — Lab Handout

==========

1. Purpose

This lab helps learners observe normal ARP resolution behavior and then recognize an ARP spoofing

pattern

(ARP cache poisoning) in a packet capture. Learners will run ARPGuard to see how simple, explainable

heuristics translate those observations into alerts.

2. Learning Objectives

By the end of this lab, a learner should be able to:

- Describe what ARP does and where it “lives” in the TCP/IP stack (link/local network boundary).

- Identify a suspicious IP→MAC mapping change in ARP traffic.

- Explain why ARP spoofing enables man-in-the-middle and traffic redirection on a LAN.

- Propose at least two mitigations and explain how they reduce ARP spoof risk.

3. Prerequisites

Required:

- Wireshark (or tshark) for opening PCAP files.

- Python 3.11+ with the project dependencies installed.

Provided in this project:

- pcaps/benign_arp.pcap

- pcaps/arp_spoof_attack.pcap

- code/arpguard_core.py (analyzer) and code/arpguard_lab_tools.py (demo)

4. Safety and Ethics

ARP spoofing is an attack technique. This lab is designed to be completed using provided PCAP files.

Learners should not perform ARP spoofing on any network unless the instructor explicitly authorizes a

controlled environment.

5. Background (Short)

ARP maps IPv4 addresses to MAC addresses on a local network segment by broadcasting “who-has”

requests

and receiving “is-at” replies [1]. Many hosts update their ARP caches based on observed ARP replies, which

is

one reason spoofing works: an attacker can inject a forged mapping (e.g., “Gateway IP is at Attacker MAC”),

causing the victim to send traffic to the attacker [1].

6. Phase 1 — Observe Benign ARP Behavior (benign_arp.pcap)

6.1 Open the PCAP

- Open pcaps/benign_arp.pcap in Wireshark.

- Apply the display filter: arp

6.2 Identify the Request/Reply Pair

Learners should locate:

- An ARP who-has request (broadcast) asking for a target IP (often the gateway).

- A corresponding ARP is-at reply that provides the target's MAC address.

6.3 Record the Mapping

Learners should record:

- Sender Protocol Address (SPA / psrc)

- Sender Hardware Address (SHA / hwsrc)

- Target Protocol Address (TPA / pdst)

- Target Hardware Address (THA / hwdst)

Expected observation:

- The gateway IP remains associated with a single, consistent MAC address across the capture.

7. Phase 2 — Observe a Spoofing Pattern (arp_spoof_attack.pcap)

7.1 Open the PCAP and Filter

- Open pcaps/arp_spoof_attack.pcap in Wireshark.

- Apply the display filter: arp

7.2 Look for an IP→MAC Conflict

Learners should identify:

- A legitimate gateway reply that maps the gateway IP to the gateway MAC.

- One or more later replies mapping the same gateway IP to a different MAC.

Expected observation:

- The capture contains at least one mapping change for the same IP address (classic poisoning symptom).

7.3 Interpret the Risk

Learners should discuss:

- If a host updates its ARP cache to the attacker's MAC for the gateway IP, where will the host send traffic?

- How could an attacker forward traffic to remain stealthy (MITM) versus dropping traffic (DoS)?

8. Phase 3 — Run ARPGuard and Interpret Alerts

8.1 Install Dependencies (if needed)

From the project root:

pip install -r requirements.txt

8.2 Run the Demo (recommended)

python code/arpguard_lab_tools.py demo --out-dir pcaps

8.3 Analyze a PCAP Directly

python code/arpguard_core.py pcaps/arp_spoof_attack.pcap --json sanity/attack_results.json

8.4 What Learners Should See

For benign_arp.pcap:

- ARPGuard should report zero or very few alerts.

For arp_spoof_attack.pcap:

- ARPGuard should report at least one IP_MAC_CONFLICT alert indicating the gateway IP mapped to multiple MACs.

9. Reflection Questions (Submit Short Answers)

---

1) What is the key behavioral assumption that makes ARP spoofing possible?

2) In the attack capture, what evidence indicates that the gateway mapping was poisoned?

3) Which ARPGuard alert is the strongest indicator of spoofing and why?

4) Name two mitigations (technical or operational) and briefly explain how each helps.

5) What limitations does ARPGuard have (false positives / false negatives) based on its simple heuristics?

10. References

---

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

Appendix B – Facilitator Notes Outline

ARPGuard Facilitator Notes

ARPGuard — Facilitator Notes (Instructor / TA)

==========

1. Recommended Timing (50–75 minutes)

- 5–10 min: ARP recap (broadcast request vs. unicast reply; ARP cache concept)

- 15–20 min: Phase 1 (benign PCAP) guided observation

- 15–20 min: Phase 2 (attack PCAP) guided observation and discussion

- 10–15 min: Phase 3 ARPGuard run + alert interpretation

- 5–10 min: Wrap-up + mitigations + reflection prompt

2. Setup Checklist

- Ensure learners can open PCAPs in Wireshark.

- Ensure Python 3.11+ is available if learners will run ARPGuard locally.

- If time is limited, the lab can be completed purely via Wireshark observation and discussion using the PCAPs.

3. Common Misconceptions and How to Address Them

A) “ARP is the same as DNS.”

- Correction: ARP resolves IPv4→MAC on a local segment; DNS resolves names→IP at higher layers.

B) “Any ARP packet is malicious.”

- Correction: ARP is normal for local address resolution. The signal is inconsistency (e.g., IP→MAC changes).

C) “ARP spoofing only affects one machine.”

- Correction: Poisoning can target multiple hosts (victims) and/or the gateway, enabling broader interception.

D) “If the attacker spoofs the gateway, traffic always breaks.”

- Correction: A MITM attacker can forward traffic to remain stealthy; drops yield obvious DoS.

4. Troubleshooting Notes

- If learners see no ARP packets, confirm Wireshark display filter uses: arp

- If ARPGuard fails to run:

- Verify dependencies installed: pip install -r requirements.txt

- Verify scapy import works: python -c "from scapy.all import rdpcap; print('ok')"

5. Teaching Emphasis (Why the Alerts Make Sense)

ARPGuard uses explainable heuristics, so the facilitator can tie alerts back to observable evidence:

- IP_MAC_CONFLICT: the same IP address is associated with multiple MAC addresses over time.

This should be visible directly in Wireshark by comparing the sender hardware addresses in ARP replies.

- UNSOLICITED_REPLY: ARP replies that are not clearly triggered by a recent who-has request.

Emphasize that this can be benign (gratuitous ARP), so ARPGuard treats it as a softer indicator.

6. Discussion Prompts (Mitigations and Defense-in-Depth)

- What changes if the network enforces DHCP snooping + Dynamic ARP Inspection (DAI)?

- Why do static ARP entries reduce spoofing risk, and what operational cost do they introduce?

- How does segmentation (VLANs) reduce the blast radius of ARP poisoning?

- What monitoring signals might complement ARPGuard (switch logs, IDS, host ARP cache audits)?

7. Answer Key Alignment

The quiz answer key includes both the correct answer and a short justification.

If a learner’s answer is directionally correct but lacks justification, partial credit can be assigned, but the rubric

goal is that learners can explain “why,” not only “what.”

8. References

---

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

ARPGuard Quiz and Answer Key

ARPGuard Quiz and Answer Key (with Justification)

==========

Instructions:

- Unless otherwise stated, answers should be 1–3 sentences.

- Where applicable, learners should justify the answer by referencing observable ARP behavior in a capture.

Q1) (Conceptual) What problem does ARP solve on an IPv4 Ethernet LAN?

Answer:

ARP resolves an IPv4 address to a link-layer (MAC) address on the local network segment.

Justification:

An Ethernet frame needs a destination MAC; ARP provides the mapping needed to send IP packets to a next hop [1].

Q2) (Conceptual) In the TCP/IP model, ARP is most closely associated with which boundary?

A) Application layer

B) Transport layer

C) Network-to-Link boundary (local delivery)

D) Physical layer

Correct Answer: C

Justification:

ARP is used for local delivery on a broadcast domain to map IP (network-layer identifier) to MAC (link-layer identifier); it is not an end-to-end transport mechanism.

Q3) (Wireshark) Which Wireshark display filter shows only ARP traffic?

Answer:

arp

Justification:

Wireshark's protocol filter "arp" matches ARP packets and excludes other Ethernet/IP traffic.

Q4) (Detection) What is the strongest single indicator of ARP spoofing in a capture?

Answer:

An IP→MAC conflict: the same IPv4 address is observed mapping to different MAC addresses over time.

Justification:

ARP poisoning relies on convincing a victim to associate a trusted IP (often the gateway) with an attacker's MAC.

A mapping change for the same IP is a direct symptom of that attack pattern.

Q5) (Applied) A host first learns "192.168.1.1 is-at 00:aa:bb:cc:dd:01" and later sees "192.168.1.1 is-at de:ad:be:ef:00:66". What risk does this create for the host?

Answer:

The host may update its ARP cache so that packets intended for the gateway IP are sent to the attacker's MAC,

enabling traffic interception (MITM) or disruption (DoS).

Justification:

On a LAN, the host uses the ARP cache to choose the destination MAC for traffic to the gateway; poisoning redirects those frames to the attacker.

Q6) (ARPGuard) Which ARPGuard alert corresponds to the situation in Q5?

Answer:

IP_MAC_CONFLICT

Justification:

ARPGuard raises IP_MAC_CONFLICT when it observes the same IP address associated with multiple MAC addresses over time.

Q7) (Reasoning) Why can ARP spoofing work even if the attacker cannot break cryptography (e.g., TLS)?

Answer:

ARP spoofing can still redirect traffic at the link layer; without additional controls, the attacker can position itself as a man-in-the-middle or cause denial-of-service regardless of application-layer encryption.

Justification:

ARP poisoning changes who receives Ethernet frames; TLS protects payload confidentiality/integrity, but it does

not prevent redirection or traffic disruption at L2.

Q8) (Mitigation) Name two mitigations and briefly explain how each reduces ARP spoofing risk.

Answer:

(Example 1) DHCP snooping + Dynamic ARP Inspection (DAI): the switch validates ARP replies against trusted

DHCP bindings and drops forged ARP packets.

(Example 2) Static ARP entries for critical hosts (e.g., gateway): the host does not accept unsolicited mapping

changes, reducing cache poisoning opportunities.

Justification:

Both mitigations reduce the attacker's ability to inject or propagate forged IP→MAC mappings.

Q9) (Limitations) Give one realistic reason ARPGuard might produce a false positive.

Answer:

Gratuitous ARP or legitimate IP/MAC changes (e.g., failover, NIC replacement, VM migration) can cause a real

mapping change that looks like an IP→MAC conflict.

Justification:

ARPGuard uses heuristic signals; legitimate network events can resemble poisoning unless contextualized.

Q10) (Interpretation) If ARPGuard reports 0 alerts on a benign capture, what conclusion is most defensible?

Answer:

The capture does not contain ARP mapping inconsistencies that match ARPGuard's heuristics; it does not prove

that the network is attack-free.

Justification:

Absence of detected anomalies is not proof of absence; ARPGuard is limited by what is present in the capture

and by the detector's rule set.

References

---

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

Appendix D – Project Overview Outline

ARPGuard Project Overview

ARPGuard Project Overview

==========

1. Project Summary

ARPGuard is a compact instructional artifact that demonstrates how Address Resolution Protocol (ARP) behavior
can be observed and how ARP spoofing (ARP cache poisoning) can be detected using simple, explainable heuristics. The author developed ARPGuard to support a hands-on lab sequence that aligns with TCP/IP model
thinking and common network-defense concepts.

The project intentionally combines:

- Learner-facing instruction (lab handout),
- Instructor-facing guidance (facilitator notes),
- A brief assessment instrument (quiz + answer key with justification),
- A small, reproducible technical artifact (PCAPs, Python analyzer, and optional web dashboard).

2. Intended Audience and Scope

The project is designed for a cybersecurity networking course where learners have basic familiarity with:

- IPv4 addressing and subnetting,
- MAC addressing and Ethernet framing,
- Packet capture and inspection concepts (e.g., Wireshark filters).

It does not assume prior experience with ARP poisoning tooling or advanced switch security features.

3. Deliverables (High-Level)

- Lab handout: “Detecting ARP Spoofing with ARPGuard”

- Facilitator notes: timing, common misconceptions, troubleshooting

- Quiz + Answer Key: includes correct answers and justification

- Source code:

- arpguard_core.py (PCAP analyzer),

- arpguard_lab_tools.py (PCAP generation + demo),

- arpguard_web_dashboard.py (optional UI)

- Supporting artifacts:

- benign_arp.pcap

- arp_spoof_attack.pcap

- figure assets (topology + analysis snapshots)

4. Technical Approach (Detection Heuristics)

ARPGuard focuses on transparent indicators that learners can validate directly in a capture:

- IP→MAC conflict: the same IPv4 address is observed mapping to multiple MAC addresses over time.

This is a primary symptom of ARP cache poisoning in a local broadcast domain [1].

- Unsolicited ARP replies: ARP “is-at” replies observed without a recent “who-has” request.

This can indicate poisoning behavior, though legitimate gratuitous ARP exists; ARPGuard treats it as a

softer

signal (lower severity).

- MAC fanout: a single MAC address claims a large number of IPv4 addresses (threshold-based), which can

be

suspicious depending on context.

5. Safety and Ethics

ARP spoofing is a real attack technique. The project materials assume a controlled lab environment and

either

(a) synthetic PCAPs, or (b) instructor-authorized isolated networks. Learners should not attempt spoofing on production or campus networks. The lab can be completed using the provided PCAPs without running any live

spoofing tools.

6. References

---

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

[2] Wireshark Foundation, “Address Resolution Protocol (ARP),” Wireshark User’s Guide, accessed 2025-12-14.

[3] Cisco Systems, “Dynamic ARP Inspection,” security configuration documentation, accessed 2025-12-14.

[4] IEEE, “IEEE 802.1X Port-Based Network Access Control,” standard overview, accessed 2025-12-14.