5300 PROJECT RESEARCH & CONTENT DEVELOPMENT

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.

Research Methodology

This project is completed individually, so all research activities were carried out by the author.

He began his research by grounding himself in the formal specification of the Address Resolution Protocol, starting with RFC 826 to understand the exact message fields, expected state transitions, and legitimate uses of ARP in IPv4 networks [1]. From there, he explored how ARP fits at the L2/L3 boundary of the TCP/IP model and how common operating systems implement ARP caching and gratuitous ARP behavior. This protocol-level view ensured that any detection logic he designed would be consistent with standards rather than anecdotes.

Next, he focused on real-world manifestations of ARP spoofing and poisoning. He examined tutorials, lab guides, and tool documentation for utilities such as arpspoof and Ettercap, cross-checking their behaviors in Wireshark traces against the Wireshark User's Guide and display filter examples [4]. By doing this, he assembled concrete patterns to look for: duplicate IP-to-MAC bindings, sudden MAC flips for a given IP, bursts of unsolicited ARP replies, and abnormal packet rates on specific hosts. He verified each pattern by inspecting packet captures and correlating them with the expected MITM attack flow.

He also studied network monitoring and log-management practices to place ARPGuard within a broader defensive context. NIST SP 800-92 helped him think about ARP anomalies as structured security events that should be logged, aggregated, and interpreted, rather than as one-off curiosities [2]. He then mapped these ideas to the CIS Controls v8, particularly the controls related to network monitoring, secure configuration, and detection of network-based attacks, to ensure that the lab's recommended mitigations would be aligned with recognized best practices [3].

Along the way, he investigated several adjacent topics that ultimately did not become part of the project. He looked at IPv6 Neighbor Discovery (ND) spoofing and Secure Neighbor Discovery but concluded that this would dilute the focus of an introductory TCP/IP and ARP-centric teaching tool. He briefly explored BGP hijacking, DNS cache poisoning, and commercial ARP spoof-detection appliances. While technically interesting, these topics either operated at very different layers or required infrastructure beyond the scope and time budget of this course project. He also reviewed advanced machine-learning-based intrusion detection approaches but decided they were unnecessary for the targeted learning objectives and would make the project harder to reproduce in a standard lab environment.

Throughout the research phase, he intentionally prioritized sources that combined protocol accuracy with pedagogical usefulness: standards and official guides for correctness, and hands-on lab documents and Wireshark examples for teachability [1]–[4]. This mix directly informed both the ARPGuard detection rules and the structure of the teaching materials.

Research Findings

From this research, several key findings shaped the ARPGuard design and the surrounding educational content.

Finding 1 – ARP is stateless and unauthenticated, making cache poisoning straightforward:

By studying RFC 826 and Wireshark traces, he confirmed that hosts accept ARP replies without strong validation and may update caches even when no explicit request was made [1], [4]. This meant that simple patterns like unsolicited replies or repeated claims about the same IP can meaningfully alter traffic paths. He translated this into ARPGuard's duplicate-IP and unsolicited-reply detection rules. In the teaching materials, he uses side-by-side screenshots of normal versus poisoned ARP tables to help learners visually connect the research insight to the attack behavior.

Finding 2 – Some "suspicious-looking" ARP traffic is normal in managed networks:

Through practical examples and documentation, he learned that gratuitous ARPs appear during legitimate events, such as host boot, IP conflict resolution, or failover [4]. This made it clear that naive rules ("any gratuitous ARP is evil") would lead to noisy false positives. Instead, he designed ARPGuard to use rate-based and context-aware thresholds: bursts of gratuitous ARPs, combined with inconsistent MAC mappings, trigger alerts, while isolated gratuitous ARPs do not. The lab script explicitly asks learners to compare benign and malicious traces to see this distinction.

Finding 3 – ARP anomalies are best understood within a wider logging and monitoring strategy:

NIST's log-management guidance emphasizes structured event definitions, retention strategies, and correlation across systems [2]. He applied this by defining a small set of ARPGuard event types (e.g., DUP_IP, MAC_FLIP, GRATUITOUS_BURST) and ensuring that each event includes timestamp, source, affected IP/MACs, and a human-readable explanation. In the code, these events are emitted as JSON, and the tutorial shows how to interpret them as part of a hypothetical SOC workflow, bridging the gap between low-level packet anomalies and higher-level incident narratives.

Finding 4 – Effective mitigations exist at multiple layers and should be taught as part of defense-in-depth:

From CIS Controls v8 and vendor best practices, he identified concrete mitigation strategies: static ARP entries for critical hosts, DHCP snooping and Dynamic ARP Inspection on switches, VLAN segmentation to limit broadcast domains, and the use of encrypted protocols to reduce the impact of MITM [3]. He integrated these into a "Mitigation Cheat Sheet" that learners complete at the end of the lab. For each mitigation, the sheet asks the learner to map the control to a TCP/IP layer and briefly explain how it mitigates the ARP spoofing scenario they just observed.

Finding 5 – Students learn ARP spoofing better when they see both the attack and the normal baseline:

By comparing multiple lab write-ups and teaching resources, he realized that many materials jump straight into launching attacks without first giving learners a stable baseline. He therefore designed the ARPGuard lab to proceed in three phases: (1) observe clean ARP traffic and build intuition, (2) introduce a controlled spoofing attack and run ARPGuard, and (3) discuss mitigations while tying symptoms back to the TCP/IP model. This sequencing is reflected in the lesson plan, facilitator notes, and quiz, making the research insight directly visible in the project content.

Together, these findings not only informed the internal detection logic and code structure but also shaped the narrative arc of the teaching materials; moving from protocol understanding, to attack observation, to mitigation and reflection [1]–[4].

Project Content

This submission includes both teaching-oriented materials and technical assets that align with the project's learning objectives and the TCP/IP/network-security focus of the course. All materials are developed by the author individually; there is no partner on this project, so all content creation effort is his.

On the teaching side, he has prepared a learner-facing lab handout titled *"Detecting ARP Spoofing with ARPGuard"*. It walks students through the prerequisites, a brief recap of ARP and its role at the L2/L3 boundary, and a three-phase activity: capturing benign traffic, running a controlled spoofing attack, and interpreting ARPGuard's alerts. The handout contains step-by-step instructions, Wireshark filter hints, reflection questions, and explicit links back to the learning objectives.

He has also drafted facilitator notes aimed at instructors and TAs. These notes summarize the lab flow, highlight common student misconceptions (for example, confusing IP addresses with MAC addresses, or assuming any ARP traffic is malicious), and provide timing guidance and troubleshooting tips. They also include suggested discussion prompts that connect ARP spoofing to broader topics like man-in-the-middle attacks and defense-in-depth.

To assess understanding, he developed a 10-question quiz that covers both conceptual and applied aspects. Questions ask learners to recognize ARP spoofing patterns in a small snippet of ARP table output, explain why a particular ARPGuard alert fired, and propose mitigations mapped to specific layers of the TCP/IP model. An answer key is provided for instructors, with brief rationales that mirror the explanations in the lab materials.

On the technical side, he is submitting the core ARPGuard source code as .txt files, including the ARP parsing and anomaly-detection engine, the Flask-based web dashboard, and helper utilities for loading .pcap files and generating synthetic traffic. These files include inline comments referencing the protocol rules and best practices derived from the research, tying implementation decisions back to the literature [1]–[4]. A README describes installation steps, dependencies, and example commands to run the lab on a Linux or macOS host.

Finally, he includes sample .pcap traces (benign and attack scenarios) and a concise project overview document that explains how all these pieces fit together into a cohesive instructional experience.

Project Content Deliverables

Item 1

- Item Description: Learner lab handout that guides students through running ARPGuard, observing ARP traffic, and answering reflection questions.
- File name/path/link: ARPGuard_Lab_Handout.pdf
- Relevant information: Primary document for students; referenced in class and in facilitator notes.

Item 2

- Item Description: Facilitator notes with timing guidance, troubleshooting tips, and discussion prompts for instructors/TAs.
- File name/path/link: ARPGuard_Facilitator_Notes.pdf
- Relevant information: Not intended for students; helps align the activity with course outcomes.

Item 3

- Item Description: Quiz and answer key assessing understanding of ARP spoofing, ARPGuard alerts, and mitigations.
- File name/path/link: ARPGuard_Quiz_and_AnswerKey.pdf
- Relevant information: Can be used as a graded assessment or an ungraded formative check-in.

Item 4

- Item Description: Project overview and mini-user guide summarizing ARPGuard's purpose, learning objectives, and high-level architecture.
- File name/path/link: ARPGuard_Project_Overview.pdf
- Relevant information: Connects the teaching materials and source code to the original project proposal.

Item 5

- Item Description: Curated .pcap traces showing benign ARP behavior and controlled ARP spoofing attacks.
- File name/path/link: pcaps/benign_arp.pcap, pcaps/arp_spoof_attack.pcap
- Relevant information: Used by both the lab handout and ARPGuard's offline analysis mode.

Item 6

- Item Description: Screenshot bundle or figure sheet illustrating ARP tables, Wireshark views, and ARPGuard alerts used in the teaching materials.
- File name/path/link: figures/arpguard_lab_figures.pdf
- Relevant information: Supports visual learners; referenced directly in the lab handout.

All items above are authored solely by the project author.

Source Code and READMEs

Item 1

- Item Description: Core ARPGuard detection engine implemented in Python with Scapy, responsible for parsing ARP traffic, maintaining the IP↔MAC map, and generating anomaly events.
- File name/path/link: code/arpguard_core.txt
- Relevant information: Submitted as .txt per course instructions; includes inline comments and docstrings, plus IEEE-style in-code references to ARP protocol behavior [1], [4].

Item 2

- Item Description: Flask-based web dashboard providing a simple UI for uploading .pcap files, viewing host tables, and inspecting ARPGuard alerts.
- File name/path/link: code/arpguard_web_dashboard.txt
- Relevant information: Also submitted as .txt; references the same engine module and includes comments explaining how events map to UI widgets.

Item 3

- Item Description: Utility script for running ARPGuard against sample captures and for generating synthetic ARP spoof traffic in a controlled lab environment.
- File name/path/link: code/arpguard_lab_tools.txt
- Relevant information: Contains example commands used in the lab handout; comments indicate which scenarios correspond to which .pcap files.

Item 4

- Item Description: README with installation instructions, environment setup steps, dependency list, and quick-start examples.
- File name/path/link: README_ARPGuard.txt
- Relevant information: Explains required Python version, library installation, and how to run the demo on a typical student workstation.

All code and documentation in this section are created by the author and do not include any personal login credentials.

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 live capture.
    - flask and jinja2 for the web dashboard.
    - pcap-related dependencies as required by Scapy.
- 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).
- System Permissions:
    - For live capture, the host may need elevated privileges (e.g., membership in a pcap group or use of sudo).
- Hardware:
    - At least one network-capable VM or physical host for running ARPGuard; an optional second VM for attack simulation.
    - These dependencies are all documented in the project README so that instructors and students can reproduce the environment reliably.

Reflection

The author's development process so far has reinforced how important it is to balance technical realism with pedagogical clarity. Early on, he assumed that simply implementing more detection rules would make ARPGuard better. During testing, however, he realized that overly aggressive rules produced confusing false positives, especially in traces containing legitimate gratuitous ARPs. This pushed him to revisit the protocol details and refine his thresholds and event types so that each alert could be explained clearly to a beginner.

One of the biggest challenges has been designing lab materials that are both step-by-step and conceptually rich. Writing down every terminal command and Wireshark filter is straightforward; explaining *why* each step matters for understanding the L2/L3 boundary in the TCP/IP model requires more thought. He addressed this by iterating on the lab handout and quiz questions, constantly checking that each activity can be tied back to at least one learning objective. He also learned to keep the UI expectations modest for this milestone, focusing on a stable engine and clear documentation before spending time on visual polish. These lessons are shaping his plan for the final phase, where he intends to tighten the integration between the dashboard, the quiz, and the mitigation cheat sheet so that learners see a coherent story rather than a collection of disconnected tools.

Project Modifications

Since the original Project Proposal, the core vision for ARPGuard has stayed the same, but several practical adjustments have been made based on time constraints and early experimentation.

First, he decided to de-prioritize advanced D3.js visualizations for this stage and instead implemented a simpler Flask-based dashboard that relies on HTML tables and straightforward status indicators. This change allows him

to focus on correctness and stability of the detection logic while still giving learners a clear view of host states and alerts. More sophisticated timelines and graphs are now treated as potential enhancements for the final submission.

Second, he has moved Zeek integration and pcap-replay automation into a stretch-goal category. Initial tests showed that integrating Zeek would increase setup complexity for students without necessarily improving the core learning objectives. By keeping Zeek optional, he maintains alignment with the course's emphasis on TCP/IP fundamentals and ARP-level reasoning.

Third, he expanded the assessment pieces beyond what was described in the proposal. In addition to the reteaching walkthrough, he developed a structured quiz and a mitigation cheat sheet that explicitly map mitigations to TCP/IP layers and CIS controls. This helps make the project's educational impact more concrete and easier to evaluate.

Overall, these modifications keep the project achievable within the course timeline while strengthening its value as a teaching tool.

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.

ADDITIONAL INFORMATION

This project is completed by a single author with no partner. All original text, code, and instructional materials are developed by the author, and all external resources are cited in IEEE style. The submitted content is designed so that an instructor or TA can run the ARPGuard lab in a standard departmental environment without needing any proprietary or personal credentials.

Appendix A - ARPGuard Lab Handout (Full Text Version)

Lab: Detecting ARP Spoofing with ARPGuard

==========

1. Lab Overview

This lab uses ARPGuard and Wireshark to observe Address Resolution Protocol (ARP)

behavior in an IPv4 network. The student first examines benign ARP traffic to

build intuition about what "normal" looks like and then analyzes a controlled

ARP spoofing attack. ARPGuard is used to detect anomalies, and the student

connects these observations to realistic network-security mitigations.

By the end of the lab, the student should be able to recognize ARP spoofing in packet traces, explain why it works, and propose defenses mapped to TCP/IP layers and defense-in-depth concepts.

2. Learning Objectives

After completing this lab, the student will be able to:

LO1. 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).

LO2. Execute a safe, local lab that reproduces ARP spoofing and capture evidence with Wireshark.

LO3. Detect a synthetic ARP spoofing attack using ARPGuard with a small (≤ 5 seconds) time-to-first-alert under default settings.

LO4. 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).

LO5. Explain how L2 compromise can enable higher-layer attacks (session hijack, credential theft), linking observations to defense-in-depth controls.

3. Prerequisites

The student should:

- Be comfortable with basic command-line usage.
- Have prior exposure to the TCP/IP model (ARP, IP, TCP/UDP).
- Have basic Wireshark experience (opening captures, applying filters).

The lab does NOT require advanced programming skills, but familiarity with

Python will make ARPGuard's code easier to follow.

4. Estimated Time

Total estimated duration: 60–90 minutes

- Part 1 – Baseline ARP Behavior: ~20 minutes
- Part 2 – ARP Spoofing and ARPGuard Detection: ~30–40 minutes
- Part 3 – Mitigation and Reflection: ~10–20 minutes

5. Environment and Files

Operating system

- Linux or macOS host, or a Linux VM.

Software / tools

- Python 3.11+
- Required Python packages (per README_ARPGuard.txt), including:
  - scapy
  - flask
  - jinja2
- Wireshark

Lab files

- pcaps/benign_arp.pcap (benign ARP traffic)
- pcaps/arp_spoof_attack.pcap (controlled ARP spoofing scenario)
- ARPGuard source files (e.g., code/arpguard_core.py,
  code/arpguard_web_dashboard.py)

Mode of operation

- Offline analysis using the provided PCAP files (required).
- Optional live capture variant if the environment permits and policy allows.

6. Part 1 – Baseline ARP Behavior

6.1 Concept Check

~~~~~~~~~~~~~~~~~~

Answer briefly in your own words:

1. What problem does ARP solve in an IPv4 network?
2. At which boundary of the TCP/IP model does ARP operate?
3. Why does a host need an IP-to-MAC mapping before sending an IPv4 packet?
~~~~~~~~~~~~~~~~~~

6.2 Inspecting Benign ARP Traffic in Wireshark

~~~~~~

1. Open Wireshark and load pcaps/benign_arp.pcap.

2. Apply the display filter:

```
arp
```

3. Locate at least one ARP request/reply pair and record:

- Which IP address is asking, and for which IP address?
- Which MAC address responds?

6.3 Building a Baseline ARP Table

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Based on the benign capture, construct a small ARP table:

IP Address	MAC Address	Notes

6.4 Baseline Reflection

~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~

Answer the following:

1. Does any IP address appear with two different MAC addresses in this capture?

2. Do you see any gratuitous ARP messages? If so, how frequent are they?

3. In 2–3 sentences, describe what “normal” ARP traffic looks like in this capture.

7. Part 2 – ARP Spoofing and ARPGuard Detection

7.1 Running ARPGuard in Offline Mode

~~~~~~

1. From the ARPGuard project root, run the core engine on the attack PCAP.
   For example (command may vary slightly depending on your environment):

```
python code/arpguard_core.py \
  --pcap pcaps/arp_spoof_attack.pcap \
  --output results_attack.json
```

2. Start the ARPGuard web dashboard:

```
python code/arpguard_web_dashboard.py
```

   By default, the dashboard should listen on http://127.0.0.1:5000 (or as configured in the script).

3. Open the dashboard in your browser and either:
~~~~~~

- Upload pcaps/arp_spoof_attack.pcap, or

- Load the precomputed results_attack.json file.

4. Wait for the host table and alerts to populate.

7.2 Examining the Host Table

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Use the dashboard to answer:

1. Which IP address appears to be the victim (for example, the host whose mapping is being hijacked)?

2. Which MAC address(es) are associated with that IP?

3. Does the same IP appear with multiple MAC addresses during the scenario?

7.3 Examining ARPGuard Alerts

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Fill in the Alert Summary Table below with the alerts you observe:

Alert Type	Timestamp	IP Address	MAC Address(es)	Explanation (your own words)

Use the following guide:

- DUP_IP: same IP appears with multiple distinct MAC addresses.
- MAC_FLIP: MAC address for a given IP changes abruptly.
- GRATUITOUS_BURST: suspicious burst of unsolicited ARP replies.

7.4 Explaining Spoofing Artifacts

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

In a short paragraph, identify at least two specific artifacts that indicate ARP spoofing in this scenario (for example, duplicate IP↔MAC bindings, gratuitous ARP bursts). For each artifact, explain why it is suspicious.

7.5 Optional Live Capture Variant

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If your environment permits and your instructor approves:

1. Run ARPGuard in live mode on a controlled interface, for example:

   sudo python code/arpguard_core.py --iface <INTERFACE_NAME>

2. Use an approved tool (such as arpspoof) within a lab network to generate ARP spoofing traffic.
3. Measure the time between starting the attack and ARPGuard's first alert.
4. Record the observed time-to-first-alert and compare it with the offline PCAP experiment.

8. Part 3 – Mitigation and Reflection

8.1 Mitigation Mapping Table

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Complete the table below by listing at least three mitigations and mapping each one to a TCP/IP layer and a brief explanation of how it helps:

Mitigation Technique	TCP/IP Layer	How It Helps Against ARP Spoofing
Static ARP entries for critical hosts		
DHCP snooping + Dynamic ARP Inspection		
VLAN segmentation		
Use of HTTPS instead of HTTP		

8.2 Reflection Questions

~~~~~~~~~~~~~~~~~~~~~~~~~

Answer in 2–4 sentences each:

1. How can an ARP spoofing attack lead to session hijacking or credential theft, even if the attacker never logs directly into the victim's account?

2. If a network already uses TLS for many services, why is it still valuable to detect and prevent ARP spoofing?

3. Suggest one way ARPGuard alerts could be integrated into a broader security monitoring or SIEM system.

8.3 Student Deliverables Checklist

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

By the end of this lab, you should have:

- A completed baseline ARP table for the benign capture.
- A completed alert summary table for the attack scenario.
- A completed mitigation mapping table.
- Written answers to the reflection questions.

9. 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 edition.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Appendix B - ARPGuard Quiz And Answer Key (Full Text Version)

Part I – Student Quiz

Q1. (Multiple Choice)

Which of the following best describes the role of ARP in an IPv4 network?

A. It encrypts data at the network layer.

B. It translates domain names into IP addresses.

C. It maps IP addresses to MAC addresses on a local network.

D. It manages TCP port allocations on remote hosts.

Q2. (Multiple Choice)

Which observation is the strongest indicator of ARP spoofing in a packet capture?

A. A single ARP request broadcast to the local network.

B. An IP address appearing with two different MAC addresses within a short time.

C. Occasional gratuitous ARP messages at host boot time.

D. DNS requests to an unfamiliar domain.

Q3. (Short Answer)

In one or two sentences, explain why ARP spoofing works in many IPv4 networks.

Refer to how hosts treat ARP replies.

Q4. (Short Answer)

ARPGuard generates a DUP_IP alert for IP address 192.168.1.20. What does this alert mean, and why might it be suspicious?

Q5. (Multiple Choice)

Which Wireshark display filter shows only ARP traffic?

A. ip.addr == 192.168.1.1

B. tcp.port == 80

C. arp

D. eth.addr == ff:ff:ff:ff:ff:ff

Q6. (Short Answer)

Describe one scenario in which a gratuitous ARP might be legitimate, and one scenario in which a burst of gratuitous ARPs might be malicious.

Q7. (Short Answer)

Name two concrete signs in an ARP table or packet capture that suggest a host might be performing a man-in-the-middle attack using ARP spoofing.

Q8. (Short Answer)

Briefly explain how ARP spoofing could lead to credential theft, even if the attacker never logs directly into the victim's account.

Q9. (Short Answer)

For each mitigation below, indicate the TCP/IP layer it primarily operates at and how it helps mitigate ARP spoofing:

a. Static ARP entries for critical hosts

b. DHCP snooping and Dynamic ARP Inspection (DAI) on switches

c. Using HTTPS instead of HTTP for web logins

Q10. (Short Answer / Reflection)

ARPGuard focuses on a fairly small set of detection rules. If you had unlimited time and resources, what is one feature or improvement you would add to ARPGuard, and why would that feature be useful from a security-operations perspective?

Part II – Answer Key

A1.

Correct answer: C

- ARP maps IPv4 addresses to MAC addresses on a local (L2) network segment.

A2.

Correct answer: B

- Seeing the same IP bound to two different MAC addresses in a short time window is a strong indicator of ARP spoofing or misconfiguration.

A3.

Sample correct answer:

- ARP spoofing works because hosts typically trust ARP replies and update their ARP caches without strong authentication or verification. Attackers can send forged ARP replies to convince a victim that the attacker's MAC belongs to a legitimate IP address.

A4.

Sample correct answer:

- A DUP_IP alert means ARPGuard observed the same IP address (e.g., 192.168.1.20) associated with more than one MAC address. This is suspicious because normally one host should own that IP; seeing different MACs suggests an attacker may be impersonating the real host or that there is an IP conflict.

A5.

Correct answer: C

- The arp display filter shows only ARP packets in Wireshark.

A6.

Sample correct answer:

- Legitimate: A host might send a gratuitous ARP when it boots or after a failover event to announce its IP-to-MAC mapping and check for conflicts.

- Malicious: An attacker might send a burst of gratuitous ARPs claiming to own another host's IP to poison ARP tables across the subnet.

A7.

Sample correct indicators (any two):

- The same IP address bound to two different MAC addresses in a short period.
- Repeated unsolicited ARP replies from a single MAC claiming to own an IP that previously belonged to another MAC.
- A sudden change in the MAC associated with the default gateway's IP.

A8.

Sample correct answer:

- By performing ARP spoofing, an attacker can position themselves between a victim and a legitimate server, forwarding or modifying traffic. As the victim sends credentials (e.g., via HTTP or other unencrypted protocols), the attacker can read or capture those credentials while still passing traffic along so the victim does not immediately notice.

A9.

Sample correct mapping:

a. Static ARP entries for critical hosts
   - Layer: Link/network boundary (L2/L3 interaction).
   - How it helps: Critical hosts pin specific IP↔MAC mappings, making it harder for an attacker to overwrite those entries with spoofed ARP

replies.

b. DHCP snooping and Dynamic ARP Inspection (DAI)

- Layer: Primarily at Layer 2 (data link) on switches.
- How it helps: DHCP snooping tracks valid IP/MAC bindings; DAI uses this information to drop forged ARP packets that do not match the learned bindings.

c. Using HTTPS instead of HTTP

- Layer: Application (with security at the transport layer via TLS).
- How it helps: Even if an attacker performs ARP spoofing and intercepts traffic, HTTPS encrypts the data, making it much harder for them to read credentials or sensitive content.

A10.

Sample acceptable answers:

- Adding integration with a SIEM or log-management system so ARPGuard alerts can be correlated with other events.
- Implementing richer visual timelines to show when ARP spoofing starts and stops.
- Adding adaptive or machine-learning-based anomaly detection to tune thresholds to a specific network baseline.

Any thoughtful feature that clearly improves detection, usability, or operational value can receive credit.