
Cyber Project - ARP Poisoning Detection & Mitigation

Ori Katz, Or Shani

Under guidance of Fraier Sorana,

and supervision of Mitrany Roy

Technion – Israel Institute of Technology

Table of Contents

1. Abstract.....	6
1.1. Motivation.....	6
1.2. Objectives.....	6
1.3. Conclusions	6
2. Introduction.....	8
2.1. The 5-Layer Network Model	8
2.2. Address Resolution Protocol.....	9
2.2.1. ARP Packet Structure:	9
2.2.2. Protocol Process\Flow	10
2.3. Man in the Middle Attack	11
2.4. ARP Poisoning Attack.....	11
2.4.1. General.....	11
2.4.2. How It Works.....	11
3. Scenario and Network Architecture	12
3.1. Scenario Assumptions.....	12
3.1.1. Network Framework.....	12
3.1.2. Attack Framework.....	12
3.2. Configurations.....	12
3.2.1. General.....	12
3.2.2. Subnet Environment	14
3.2.1. Topology.....	14
3.2.2. Subnet Implementation	15
3.3. Attack Tools and Profile	15
3.3.1. Network Mapping	16
3.3.2. ARP Cache Poisoning.....	16
3.3.3. Implementing MITM	16
3.4. Attack Methodology	17
3.4.1. Before Attack Activation	17
3.4.1. After Attack Activation.....	20
4. Considerations for Solution Approach.....	22
4.1. Deciding What Environment the Solution is For	22
4.2. Micro-Segmentation	22
4.2.1. What is Micro-Segmentation?	22
4.3. Deciding What the Solution Should Provide.....	23
4.3.1. Existing ARP Poisoning Mitigation Techniques	23
4.3.2. Drawing Conclusions	23
4.4. Primary Concerted Traits	24
5. Practical Implementation Principles.....	25
5.1. General.....	25
5.2. Outline.....	25
5.3. Controlling the ARP Cache	25
5.4. Multiple Detection Techniques.....	25

5.4.1. Passive Detection.....	25
5.4.2. Proactive Detection.....	26
5.5. Multiple Prevention Techniques.....	26
5.6. Selecting the Right Tools.....	26
6. Solution Description.....	27
6.1. General.....	27
6.2. Capabilities Overview.....	28
7. Solution Implementation.....	29
7.1. Tools Used.....	29
7.1.1. General.....	29
7.1.2. Language and Libraries	29
7.1.3. Linux System Libraries.....	29
7.2. Objects and Data Structures	30
7.2.1. ARP table entries.....	30
7.2.2. Blacklist\Whitelist	31
7.3. Features and Capabilities.....	32
7.3.1. ARP Table Lock	32
7.3.2. ARP Table Unlock	32
7.3.3. ARP Table Dedicated Lock.....	33
7.3.4. ARP Table Dedicated Unlock.....	33
7.3.5. Populate ARP Table From Static Source.....	34
7.3.6. Connection Control Mechanism (Network Filtering).....	34
7.3.7. Monitor Network Traffic (Detect Typical Attack Behavior – Passive).....	35
7.3.8. Scan and Search Mode (Detect Typical Attack Behavior – Active)	36
7.3.9. Guard Mode.....	37
7.3.10. [Advanced] Fix ARP Cache Dynamically	37
7.3.11. [Advanced] Retaliate by ARP-Spoofing the Attacker	38
7.3.12. [Advanced] Help Fix the ARP Cache of Neighboring Hosts	38
8. Results.....	39
8.1. Solution Effectiveness	39
8.1.1. Standard Tools	39
8.1.2. Advanced Tools.....	40
9. Conclusions.....	41
9.1. Findings	41
9.1.1. Objectives Attainment	41
9.1.2. Places for Improvement.....	41
9.2. Ideas for Further Development	42
10. Acknowledgments.....	43
11. Bibliography.....	44

Table of Figures

Figure 1-1: Visual representation of what ARP-poisoning feels like	7
Figure 2-1: Structure of the ARP packet	9
Figure 2-2: Man in the Middle Traffic Redirection.....	11
Figure 3-1: Workings of a Hub	13
Figure 3-2: Workings of a Switch	13
Figure 3-3: Detailed LAN Topology and Setup	15
Figure 3-4: Diagram of Man-in-the-Middle on the LAN.....	16
Figure 3-5: Before Attack Activation - Bob's ARP cache is in order.	17
Figure 3-6: Before Attack Activation - Assert Bob-Kali connection.	18
Figure 3-7: Before Attack Activation - Bob-Alice ping trace is normal	19
Figure 3-8: After Attack Activation - Bob's ARP cache is poisoned.....	20
Figure 3-9: After Attack Activation - Bob-Alice ping is captured by Kali's Wireshark.....	21
Figure 4-1: Visualization of a comparison between networks with and without microsegmentation	23
Figure 6-1: The "ARPAche" protection program logo	27
Figure 6-2: ARPAche had become somewhat of a "Swiss army knife"	28
Figure 7-1: Typical Linux ARP table example	30
Figure 7-2: The effect of locking the cache, using the API 'ARPAche_lock'	32
Figure 7-3: The effect of unlocking the cache, using the API 'ARPAche_unlock'	32
Figure 7-4: The effect of locking\unlocking the cache, using the APIs 'ARPAche_lock_match' and 'ARPAche_unlock_match'	33
Figure 7-5: The effect of repopulating the cache, using the API 'ARPAche_get_static_cache'	34
Figure 7-6: The effects of managing the blacklist, using the API 'ARPAche_blacklist' commands	35
Figure 7-7: The results received after running the passive detection	36
Figure 7-8: The results received after running the active detection	36
Figure 7-9: The alert notification displayed when an attack is detected	37

Figure 7-10: The results received after running the dynamic cache repopulation 37

Figure 7-11: The attacker's ARP cache before and after ARPache's retaliation 38

Figure 7-12: Fixing Alice's cache. As can be seen, Bob's (192.168.10.102) MAC address is corrected even though ARPache is not running on Alice..... 38

Figure 8-1: Behavior of a full defense policy, named "ARPache_warrior" 39

Figure 8-2: Fixing Alice's cache from Bob's ARPache, allows for some legitimate conversation 40

1. Abstract

1.1. Motivation

ARP poisoning (or ARP spoofing) is a type of cyberattack that involves sending malicious ARP packets to one of the default gateways on a Local Area Network (LAN) in order to change the IP to MAC address pairings (ARP cache) of victim nodes. The ARP Protocol translates IP addresses into MAC addresses and was not designed with security in mind. Therefore, for an attacker inside - or with access to - the LAN, ARP poisoning attacks are extremely easy to carry out.

Although a simple attack itself, successfully completing it allows the attacker to implement more sophisticated and harmful attacks, such as “Man in the Middle”, and consecutively to read\alter private communications and to impersonate the victims. ARP poisoning attacks are a known and common problem for network security, plaguing LANs all over the world. Although many solutions exist today, ARP poisoning attacks are still a concern and new ways of detection and protection are still developed.

1.2. Objectives

In this project, we attempt to study the subject in depth as well as examine existing solutions, with the main objective of implementing our own mechanism for detecting and protecting a client from such an attack. The prescribed requirements also specified certain capabilities the solution should include: provide protection at end-station level, traffic listening mode, locked mode, enable populating the ARP cache with a known static list, and manage blacklisting hosts.

1.3. Conclusions

Our solution was implemented in a virtual environment we modeled as a network with a virtual subnet and traffic passing through a switch. The attack was implemented using “Bettercap” on a “Kali” machine. In our work we used several tools, e.g: Linux shell scripts, Linux system calls, Linux net libraries (as arping, nftable, etc.), python, scapy, wireshark and more.

Through our research we were exposed to many networking concepts and principles as well as different protection methods and strategies. This research led us to specify more requirements and capabilities to include in the solution, pushing it beyond the familiar basics. The process by which we came by the selected solution, is specified in this paper.

In the end, after examining its capabilities and testing its performance, we found that our implemented solution surpassed preliminary expectations, as on top of being able to accurately detect ARP-poisoning attacks and defend against them, it can even fight back.

The implemented features work as desired, including some novel ones, and can be used as part or even a basis of a much larger defense suite. The solution can be further developed to incorporate micro-segmentation security measures, and use each host and other network systems to create a variation of a zero-trust network.



Figure 1-1: Visual representation of what ARP-poisoning feels like