A Distributed Security Approach against ARP Cache Poisoning Attack

Mehdi Nobakht
University of New South Wales (UNSW)
Canberra, Australia
mehdi.nobakht@unsw.edu.au

Hadi Mahmoudi
Omid Rahimzadeh
ACM Member
Tehran, IRAN

ABSTRACT

The Address Resolution Protocol (ARP) has a critical function in the Internet protocol suite, however, it was not designed for security as it does not verify that a response to an ARP request really comes from an authorized party. This weak point in the ARP protocol can be used by adversaries to send spoofed ARP messages onto a Local Area Network (LAN) and poison victim's ARP cache table. Once an attacker succeeds in an ARP spoofing attack, they can perform a man-in-the-middle (MITM) attack to relay or modify the data or launch a denial-of-service (DoS) attack. Thus, it is crucial to detect and counter an ARP cache poisoning attack. A few research works have proposed solutions against this attack. In the literature, we reviewed these works to see how effective these security solutions are in detecting and countering ARP cache poisoning attack. We observed that suggested mechanisms are not efficient enough in detecting and mitigating this attack. To this end, this paper proposes a distributed algorithm to instantly detect ARP cache poisoning attack and discover information about host(s) used by the attacker and finally counter this attack using acquired information. We implemented a prototype of the proposed algorithm, called *agent*, to be run on every host within the network. Agents communicate each other to orchestrate a distributed security tool to detect and mitigate ARP cache poisoning attack.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; *Denial-of-service attacks.*

KEYWORDS

Address Resolution Protocol (ARP), Man-In-The-Middle (MITM), ARP poisoning, DHCP spoofing, DNS spoofing, Host spoofing

ACM Reference Format:
Mehdi Nobakht, Hadi Mahmoudi, and Omid Rahimzadeh. 2022. A Distributed Security Approach against ARP Cache Poisoning Attack. In *Proceedings of the 1st Workshop on Cybersecurity and Social Sciences (CySSS '22), May 30, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3494108.3522765

1 INTRODUCTION

The Address Resolution Protocol (ARP) is used in a Local Area Network (LAN) to discover the data link layer address of a machine within the network and associate it to the network layer address. In the Internet Protocol Suite, ARP maps Internet Protocol version 4 (IPv4) address used by network layer (layer 3) to Media Access Control (MAC) address used by data link layer (layer 2). ARP messages are encapsulated by data link layer protocol and ARP itself is a *request-response* protocol. Hosts in a LAN have ARP tables, also known as ARP cache, to maintain such mapping between IP addresses and MAC addresses. If there is no entry for a certain IP address in ARP cache of a host, an *ARP Request* message is broadcasted to the network to discover the host that owns the IP address. this request message is received by all hosts within the LAN. The host with the same IP address then responds to the source host with a unicast message reporting its MAC address. The source host updates its ARP cache that can be used for future communications.

ARP is a stateless protocol; it accepts ARP responses even if no request has been sent out and there is no verification to authenticate the sender host. This weak point in ARP protocol makes it vulnerable to ARP cache poisoning attack, also known as ARP spoofing attack, where an attacker poisons the target ARP cache and associate its own MAC address with the IP address of another host typically a host that provides a critical service such as gateways. In this way, any traffic from the target host toward the host (e.g., gateway) with that IP address to be sent to the attacker instead. Thus, an ARP cache has the disadvantage of potentially being used by adversaries where a new entry of IP and MAC addresses can be entered into the ARP cache without authentication. Once the attacker succeeds in poisoning its target ARP cache, other attacks such as man-in-the-middle (MITM), Denial of Service (DoS) or session hijacking attacks can be launched.

While it is important to secure ARP against ARP cache poisoning attack, it is in fact hard to do so for several reasons. First, any modification in ARP protocol may not be a viable solution for existing network devices which running current version. A few research work [2, 6, 7] propose using cryptography to encrypt ARP messages. However, such amendments to ARP protocol would be incompatible for networks using current ARP protocol. Second, network traffic might substantially be increased. Other research work suggesting to consider an authentication server to ensure which peer sends ARP messages. Obviously, such schemes would increase ARP network traffic significantly. Furthermore, considering such servers in the network would increase the security risk as they could be potential targets to compromise networks as well. Third, some security features rely on additional servers or databases to resolve MAC address. However, these approaches may not be suitable for dynamic networks in which hosts join and leave the network. Moreover, it requires extra administration efforts to update the database.

There could be multiple approaches to secure ARP cache such as *protocol-based* [2, 6, 7] or *centralized-based* monitoring of ARP traffic [3–5, 8]. However, these approaches are either impractical as they are incompatible with legacy network or make the centralised decision-maker vulnerable to attack and may add considerable overhead.

To advance the state-of-the-art, we propose a distributed mechanism to secure LAN by detecting and mitigating ARP cache poisoning attacks. To this end, our tool, also called *agent*, is installed on every host within the network. Distributed agents communicate with each other as soon as they are installed. An agent consists of three components. The first component is *scanning and checking* and has two responsibilities; it ***(i)*** scans the network to discover critical nodes such as gateways and firewalls to prioritize these nodes over regular nodes within the network and ***(ii)*** ensures correctness and health of the ARP cache of the host running this agent. The second components is *detection* and meant to identify the attack by analysing any change to ARP cache. The last component is *mitigation* which counters the attack by rewriting poisoned entries of ARP cache and isolates attacker node by sending its IP and MAC addresses to neighbouring agents.

Contributions. Our paper makes the following contributions:

- We reviewed research work proposing techniques and mechanisms to secure ARP cache and classify these techniques.
- We propose a distributed security tool to be installed in a LAN nodes to detect ARP cache poisoning attack with a novel distributed algorithm to identify poisoned ARP cache and resolve it.
- We have a preliminary implementation of the proposed architecture in Python.

The remainder of this paper is organised as follows. Section 2 reviews approaches against ARP cache poisoning attack, provides a classification of them and highlights their advantages and limitations. Section 3 provides details of our proposed algorithm. We describe a prototype implementation of the proposed algorithm and evaluation results in Section 4. Finally, the paper concludes in Section 5.

2 RELATED WORK

In the literature, we reviewed scientific research works providing techniques and mechanisms to solve the fundamental ARP protocol problem. A number of works investigated techniques and challenges in the context of ARP cache poisoning attack in computer networks. Table 1 provides an overview of such techniques to detect ARP cache poisoning in wirelines LAN networks.

A first class of prevention mechanisms against ARP cache poisoning is *protocol-based.* These solutions are typically based on cryptography and modification of ARP protocol or considering additional steps in the ARP protocol.

A second class of detection and prevention techniques against ARP cache poisoning is based on monitoring ARP protocol traffic to identify poisoned ARP packets that contain conflicting source information. Depends on the location of the detection and prevention agent, there are two main variants of this class. *Centralized solutions* where a central node (on a server or a proxy) usually implements the entire process of detection and prevention and *distributed solutions* where multiple nodes are being used for detection and prevention purposes.

Limmaneewichid and Lilakiatsakun [6] proposed a *protocol-based* mechanism to add an authentication process to ARP called P-ARP. Authors suggest hiding target IP Address in ARP Request messages to maintain the IP address confidential. P-ARP uses a magic number, nonce, and the authentication data which is the output of HMAC hash function. In this way, in addition to encrypting ARP messages, P-ARP makes ARP protocol secure against ARP cache poisoning attacks. However, cryptographic algorithms often require well-resourced devices (memory storage, computing and power). Thus, implementing such algorithms in a typical node of legacy network is quite difficult and challenging which makes the network unacceptably slow.

Among *centralized-based* research work, Daniyal et al. [3] proposed a 3-phase scheme to provide protection against ARP cache poisoning attack. In the first phase, it employs the method presented in [9] to identify suspicious hosts in the network that might be used for attacking. It basically looks for hosts with IP packet routing enabled. To find these hosts, it broadcasts a trap Internet Control Message Protocol (ICMP) ping packets in the network. Only host(s) whose network card is routed will send the reply to trap ICMP packet which make such hosts to be identified. In the second phase, the test host forwards the information of suspicious hosts to all agents deployed on each host resides in the network. The agent uses the information and make a comparison with the ARP cache to identify any mismatch entries. Finally, in the third phase, the test host collects and analyses mismatch reports by agents. This enables the test host to identify attacker(s) and victim(s). To mitigate the threat, the test host will then send an ARP request to victim host to override the poisoned ARP cache. The proposed scheme can detect and prevent ARP cache poisoning attack, however, it comes with a few limitations. One of drawbacks of this method is that finding suspicious hosts by sending trap ICMP ping packet messages cause an extra load on large networks. This overload on network slows down the network as well as the mitigation process. In addition, when ARP cache poisoning attacks occur frequently in the network, the detection method might cause the network to be vulnerable to DoS attack due an increase in network traffic. Finally, the test host in this scheme is a centralised decision-maker which makes it vulnerable to a single point of failure (SPOF). If the test host is being compromised, the attacker can infiltrate the entire detection and prevention system.

On *distributed* techniques, Ghazi et al. [1] proposed a simple tool that is deployed on all end nodes reside in the network to identify ARP cache poisoning attacks. First, it captures the ARP table with an initial assumption that it is safe with no attack. It then monitors the ARP table and issues an alert when any illegitimate change detected. For mitigation, it tries to restore the ARP cache and block the attacker host using the tool deployed on the admin system. Again here, the proposed solution helps to protect the network, however, it has three main drawbacks. Firstly, it assumes that ARP cache is initially safe. The second drawback is that it only considers

Table 1: Classification of techniques and mechanisms to prevent or detect ARP cache poisoning

Class of technique	Methodology	Advantages	Limitations
Protocol-based	1. ARP Correction [2, 6] 2. ARP Replacement [7]	Optimised fundamental solution with no need to any additional measures.	1. Incompatible with legacy network 2. Requires additional computation resources
Centralized-based	1. Server-based [4, 5, 8] 2. Agent-based [3]	More complete solutions can be provided to identify and prevent attacks.	1. Single point of failure 2. Vulnerable to DoS due to increased network traffic because of permanent connection to authentication servers
Distributed-based	Agent-based [1]	1. Eliminates additional inter-network communication with the server. 2. Each agent can make decision, thus, the response speed is high.	1. Poor design in communication between agents may cause overload network traffic, thus, slow the process down. 2. Vulnerable to DoS in some cases.

changes to MAC addresses and does not check IP addresses. However, when an attacker forges MAC address and changes IP address, the tool fails to detect this type of ARP poisoning attacks. Finally, it uses a static table including IP and MAC addresses of legitimate routers. However, in large networks such as enterprise and campus networks, the IP and MAC addresses of routers are being changed frequently for security reasons. If this tool is employed in such large networks, any change to IP and MAC addresses of routers must be updated in the tool accordingly. This will add a tedious job for network administrators to manually update new IP and MAC addresses of routers on every agent in the network.

As discussed, due to the limitations of protocol-based solution (demand for high computational power and incompatibility with existing network) and centralized-based solutions (a single point of failure and substantial overhead), this paper proposes a novel distributed security solution to ARP cache poisoning attacks. In contrast to existing distributed-based solutions, this paper provides a practical solution to ARP cache poisoning attack which can detect and prevent any type of such attacks, react faster and last but not least can be deployed on any scale of network with minimum burden for network administrators. We propose a multi agent distributed security mechanism in which each agent on an end-host is autonomously responsible for detection and prevention of ARP cache poisoning attack with minimum overhead on computation power and communication traffic.

3 THE PROPOSED SOLUTION

In this section, we first outline a high-level overview of the our proposed security solution by highlighting the steps that it takes to monitor the network and counter potential ARP cache poisoning attacks. The detailed design will then be provided.

3.1 Overview

The proposed security solution against ARP cache poisoning attack consists of three interrelated steps as discussed below.

3.1.1 Step one: Collecting and Checking. When there is necessary to monitor network traffic, it is often desirable to set priority for critical nodes over regular hosts. Critical nodes are often the main target of attacks as they provide critical services to the entire network. When critical nodes are compromised, it can have severe consequences on the entire network. Furthermore, monitoring and scanning the entire network often take a long time. The lack of considering priority for critical services especially in large scale networks can add substantial delay in detection and prevention of any type of attack and in particular ARP cache poisoning attack. This in turn increases the risk and harm of attack. Having ensured that critical nodes are safe and secure, the agent will then check less critical or ordinary hosts for ARP cache poisoning attack.

To this end, we set priority for critical nodes over regular hosts resides in the network. Each agent deployed on end hosts initially discovers gateways and critical nodes in the network. The agent maintains the details of critical nodes in a list called *Critical-nodes (CNs)*. The agent then checks the correctness, accuracy and health of its ARP table. Once it ensures the ARP table is safe, it stores a capture of ARP table in a file called *Safe-ARP*. It then collects real-time ARP caches on a regular basis at short intervals and saves the result in another file called *New-ARP*. Agents generate hash codes of both *Safe-ARP* and *New-ARP* files and regularly compare them. Any change in them will be analyzed in the next step.

3.1.2 Step two: Detection. The agent, at this stage, reviews and analyzes any changes in the *New-ARP* compared with *Safe-ARP* to detect possible attacks including IP and MAC address forgery and ARP cache poisoning attack.

3.1.3 Step three: Mitigation. Once a malicious host is detected, the agent isolates the attacker host by blocking any traffic from and to the attacker. It also requests other agents within the network to isolates the attacker.

3.2 Detailed Design

In the first step, the installed agent discovers the critical network nodes using Internet Router Discovery Protocol (IRDP). The agent uses the conventional port number of critical services such as DHCP (67), DNS (53), SQL Server (1433), Web Detects server (80) and multicasts ICMP messages to the network to discover gateways and critical nodes. If different port numbers are used, the new assigned port number can be entered manually to the agent. After discovering critical services, the agent stores them in a list called Critical-nodes (CNs). The agent then multicasts an *ARP request* message to the network and once it ensures that the returning result is correct and healthy, it stores the result in a file called *Safe-ARP*. From now on, the agent continuously monitors real-time ARP cache and saves it in a file called *New-ARP* every few seconds. The agent then compares hash files of both *Safe-ARP* and *New-ARP* for any changes.

if a change occurs in the ARP table of an identified critical node, the *agent* analyzes the difference to detect related attack types depends on different conditions that are explained in detail below. Our proposed distributed tool is able to detect various kind of ARP cache poisoning attacks. To this end, it considers two priority levels and three conditions to cover all possible situations. Algorithm 1 shows the details of the proposed approach.

3.2.1 Priority One. The agent takes every entry within *CNs* list and finds the corresponding entry in *New-ARP* list with the same IP address to make a comparison between MAC addresses of these two entries. If a change is detected, the host running this agent is flagged as victim node and the agent looks for the trace of the attacker (IP and MAC address of the compromised host by the attacker) in the host's *New-ARP* list. Table 2 shows a typical example of situation in priority one. Note that if one of the nodes in the *CNs* list is detected as a compromised node, the host on which the agent is installed and has detected this attack is compromised. Depends on attack types, the following steps identify the attacker through 3 conditions.

Condition One. In this condition, attacker scans the victim host to discover information about the target host. *New-ARP* table contains two different IP addresses that have the same MAC address. This indicates an ARP cache poisoning attack is taking place.

As shown in Table 2, because the IP address 192.168.1.1 can be recognized as the router, the attacker's IP is probably 192.168.1.12. The host running this agent is flagged as victim. Our proposed solution identifies the victim system, the type of attack and the attacker profile (line 7 of the algorithm). The agent then sends discovered information to other agents for mitigation.

Table 2: An example of <IP, MAC> binding stored in *New-ARP* and *CNs* tables illustrating Condition One of Priority One situation.

(a)

New-IP	New-MAC
192.168.1.1	d3-c2-76-34-34-14
192.168.1.10	1t-f0-76-23-13-35
192.168.1.12	d3-c2-76-34-34-14

(b)

CN-IP	CN-MAC
192.168.1.1	d3-c2-76-34-34-2a
192.168.1.2	3b-4r-89-52-1g-6j
192.168.1.3	e4-f0-76-23-23-3l

Condition Two. This condition occurs when an attacker leaves no trace of himself in *New-ARP* list and only attempts to forge the MAC address of a critical node on a victim node. Table 3 shows a typical example of this condition. To identify the attacker profile in this condition, our proposed tool propagates the forged MAC address to all other agents in the network. If the agent receives a response with an IP address associated with that MAC address (first case), then that IP address is attacker's IP (line 11 of the algorithm). When no response is received (second case) (line 15 of the algorithm), this implies that the attacker did not scan the network at all but was present in the network and is still connected to the network. In this case, the agent activates a procedure called `net-scan` which network scanning is used to identify the attacker IP.

Table 3: An example of <IP, MAC> binding stored in *New-ARP* and *CNs* tables Condition Two of Priority One situation.

(a)

New-IP	New-MAC
192.168.1.1	d3-c2-76-34-34-14
192.168.1.10	1t-f0-76-23-13-35
192.168.1.12	t4-g5-98-13-y7-u7

(b)

CN-IP	CN-MAC
192.168.1.1	d3-c2-76-34-34-2a
192.168.1.2	3b-4r-89-52-1g-6j
192.168.1.3	e4-f0-76-23-23-3l

Condition Three. In this condition, the status of *New-ARP* is the same as in Condition Two. The attacker performs ARP spoofing but temporarily disconnects from the network to prevent detection. In response to this condition, the agent randomly selects one of the neighbouring agents and requests a net scan (line 35 of the algorithm). This procedure is repeating on a regular basis until the attacker returns to the network and the agent would detect it immediately.

3.2.2 Priority Two. Having examined all critical nodes in the network, the agent will analyze ARP table of other ordinary hosts in the network to detect any MAC forgery or MITM attack. Once the agent detects two entries in the *New-ARP* list with the same MAC address but different associated IP addresses, this indicates that one of the IP addresses is the attacker's IP address (line 21 of the algorithm). Table 4 shows a typical example of this case. In order to find out attacker's IP address, the agent sends an `ARP Request` message and compares the target MAC. Once the attacker and the victim host are discovered, this information is published among all internal agents to perform corresponding mitigation actions.

Finally, in the third step, mitigation process will commence once the agent detects an attack. Through this process, the network admin set <IP, MAC> addresses of routers and attackers statically at variable time intervals (first 10 minutes, second time 30 minutes, third time 2 days). The admin's host then publishes the detected attacker's <IP, MAC> addresses to all neighbouring agents. This will prevent the attacker from launching an attack on the network within the initial set time. If the time of static <IP, MAC> addresses of the attacker and router expires and the attacker attacks again, the time interval will increase.

4 PRELIMINARY EVALUATION

We have implemented a prototype of the scheme as a software tool called *agent* in Python and include approximately 500 lines of code. The tool will soon be published as an open source on OWASP site.

Algorithm 1: ARP Spoofing Attack Detection (run by all *agents* and repeated at short intervals)

```
Input  : List of Critical Nodes of Network ($CNs$), New_ARP_cache_log of Agent ($New\_ARP$), Neighbors
Output : Attacker IP
begin
    $Attackers \leftarrow \emptyset$
    foreach $(IP, MAC)_{CNs} \in CNs$ do
        foreach $(IP, MAC)_{New\_ARP} \in New\_ARP$ do
            if $IP_{CNs} = IP_{New\_ARP} \wedge MAC_{CNs} \neq MAC_{New\_ARP}$ then
                if $(\exists((IP, MAC)_{New\_ARP}) \in New\_ARP \mid IP_{New\_ARP} \neq IP_{CNs})$ then
                    $Attacker.push((IP, MAC)_{New\_ARP})$                         // Condition One
                else
                    publish $MAC_{New\_ARP}$ to all agents in network
                    if receive IP from other agent then
                        $Attackers.push((IP, MAC)_{New\_ARP})$                    // condition two (first case)
                    else
                        $IP = net\_scan(MAC_{New\_ARP}, (IP, MAC)_{CNs}, Neighbors)$
                        if $IP$ then
                            $Attackers.push((IP, MAC)_{New\_ARP})$                // condition two (second case)
            else if $((IP_{CNs} \neq IP_{New\_ARP}) \wedge ((\exists((IP, MAC)_{New\_ARP}) \in New\_ARP))$ then
                Get_MAC = $send\_arp\_request(IP_{New\_ARP})$ in network
                if $(Get\_MAC = MAC_{New\_ARP})$ then                             // $Get\_MAC \Rightarrow MACAddress$
                    $Attacker.push((IP, MAC)_{CNs})$                              // Priority Two
                else
                    $Attacker.push((IP, MAC)_{New\_ARP})$                         // Priority Two
    if the attacker list is not empty then
        publish attacker list to all $Neighbors$
        isolate attackers and $CNs$ for limited time                              // Prevention step
Procedure $net\_scan(MAC_{New\_ARP}, (IP, MAC)_{CNs}, Neighbors)$
    M = send ARP request $(IP \in IP_{CNs})$ in network
    if $M = MAC_{CNs}$ then
        send ARP request for net_scan to all subnet nodes
        $ni\_list$ = Recieved ARP reply including $(IP, MAC)$ from all nodes in network
        if $(\exists(IP, MAC)_{ni\_list} \in ni\_list \mid MAC_{ni\_list} = MAC_{New\_ARP} \wedge IP_{ni\_list} \neq IP_{CNs})$ then
            return $IP_{ni\_list}$

    else
        choose one random neighbor from $Neighbors$ as ni
        run Procedure $net\_scan(MAC_{New\_ARP})$ by ni and exit.                 // condition three (third case)
```

Table 4: An example of <IP, MAC> binding stored in *New-ARP* table illustrating Priority Two situation.

New-IP	New-MAC
192.168.1.11	i9-4r-90-54-6y-3r
192.168.1.10	1t-f0-76-23-13-35
192.168.1.16	i9-4r-90-54-6y-3r

Experimental Setup. We demonstrate the applicability of proposed scheme and the utility of the implemented design in a real-time environment in our lab. The testbed includes a LAN network with 15 hosts and 3 switches. Different operating systems are used to run hosts including Linux Ubuntu and Windows 10. We installed and executed the implemented tool in all hosts in the network.

We have also developed a Python script to launch ARP cache poisoning attack in various scenarios similar to those described in Section 3 through Table 2, 3 and 4.

Preliminary Evaluation. Extensive experiments are conducted by executing the attack script on random hosts in the network to create three different scenarios as discussed in Section 3. In all these cases, an *agent* on a host detects the attacker host instantly and isolates it. The agent then notifies other hosts in the network to take appropriate action to counter the attack.

5 CONCLUSIONS

We have presented a distributed security mechanism to detect ARP cache poisoning attack and counter the attack. It features a novel distributed algorithm for detection. The algorithm considers every possible situations that such attack can occur. Distributed algorithms are being run simultaneously on every node of the network. The algorithm set priority for hosts providing critical services over ordinary hosts. The node that first detects the attack instantly isolates the compromised host by removing its IP and MAC addresses from the node's ARP table. It then communicates with other nodes within the network requesting them to isolate the host used by the attacker. The proposed mechanism is capable of detecting more advanced type of this attack in which the attacker leaves very little trace of itself.

The prototype of proposed mechanism has been implemented in Python as a soon-to-be-released tool. The feasibility and efficiency of the prototype demonstrated through conducting extensive experiments in a LAN network with 15 hosts where we performed various types of ARP cache poisoning attacks. The results of the evaluation demonstrate instant detection with milliseconds precision with minimum overhead on network traffic. In future, we are planning to evaluate our scheme in a large scale LAN of an order of 100 hosts and make a comparison with similar mechanisms on the basis of the number of overhead packets in the network during the detection process.

REFERENCES

[1] Ghazi Al Sukkar, Ramzi Saifan, Sufian Khwaldeh, Mahmoud Maqableh, and Iyad Jafar. 2016. Address Resolution Protocol (ARP): Spoofing Attack and Proposed Defense. *Communications and Network* 08 (01 2016), 118–130. https://doi.org/10.4236/cn.2016.83012

[2] D. Bruschi, A. Ornaghi, and E. Rosti. 2003. S-ARP: a secure address resolution protocol. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.* 66–74. https://doi.org/10.1109/CSAC.2003.1254311

[3] A. Chronopoulos, Abdul Khan, Mudassar Aslam, and Daniyal Sakhawat. 2019. Agent-based ARP cache poisoning detection in switched LAN environments. 8 (01 2019). https://doi.org/10.1049/iet-net.2018.5084

[4] S. Hijazi and M. S. Obaidat. 2019. A New Detection and Prevention System for ARP Attacks Using Static Entry. *IEEE Systems Journal* 13, 3 (2019), 2732–2738. https://doi.org/10.1109/JSYST.2018.2880229

[5] S. Kumar and S. Tapaswi. 2012. A centralized detection and prevention technique against ARP poisoning. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec).* 259–264. https://doi.org/10.1109/CyberSec.2012.6246087

[6] P. Limmaneewichid and W. Lilakiatsakun. 2011. P-ARP: A novel enhanced authentication scheme for securing ARP. In *Proceedings of 2011 International Conference on Telecommunication Technology and Applications*, Vol. 5. IACSIT Press, Singapore.

[7] Myeongjin Oh, Young-Gab Kim, Seungpyo Hong, and Sung Deok Cha. 2012. ASA: Agent-based secure ARP cache management. *IET Commun.* 6, 7 (2012), 685–693. https://doi.org/10.1049/iet-com.2011.0566

[8] D. Srinath, S. Panimalar, A. Simla, and J. Deepa. 2015. Detection and Prevention of ARP spoofing using Centralized Server. *International Journal of Computer Applications* 113 (2015), 26–30.

[9] Zouheir Trabelsi. 2005. Switched Network Sniffers Detection Technique Based on IP Packet Routing. *Information Systems Security* 14 (09 2005), 51–60. https://doi.org/10.1201/1086.1065898X/45528.14.4.20050901/90089.7