Preventing ARP Cache Poisoning Attacks: A Proof of Concept using OpenWrt

Andre P. Ortega, Xavier E. Marcos, Luis D. Chiang and Cristina L. Abad
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
Email: {aortega,xmarcos,lchiang,cabad}@fiec.espol.edu.ec

***Abstract*—The Address Resolution Protocol (ARP) is used by computers to map network addresses (IP) to physical addresses (MAC). The protocol has proved to work well under regular circumstances, but it was not designed to cope with malicious hosts. By performing ARP cache poisoning or ARP spoofing attacks, an intruder can impersonate another host (man-in-the-middle attack) and gain access to sensitive information. Several schemes to mitigate, detect and prevent these attacks have been proposed, but each has its limitations. In this paper we propose a solution to the problem that can be implemented in SOHOs using low-end networking equipment running the OpenWrt firmware. The solution proposed is effective and inexpensive and presents several advantages over other existing prevention methods.**

I. Introduction

The Address Resolution Protocol (ARP) [1] is used by computers to map network addresses (IP) to physical addresses (MAC). When a host wants to learn the MAC (Media Access Control) address of another host, it broadcasts an ARP request on the network asking for the MAC address of the host with IP (Internet Protocol) address X. The host with the given IP answers back in a unicast ARP reply indicating its MAC address. The host that issued the request caches the $\langle IP, MAC \rangle$ pairing in a local ARP cache so that it does not have to issue the same request in the near future.

ARP has proved to work well under regular circumstances, but it was not designed to cope with malicious hosts. With ARP cache poisoning or ARP spoofing attacks, an intruder can impersonate another host and gain access to sensitive information. Furthermore, these attacks can be performed by novices or *script kiddies*, using widely available and easy to use tools specially designed for the purpose [2]–[4].

As we will explain later (see Section II-B), the security problems that the use of ARP introduce in a local area network (LAN) may create vulnerabilities to the distributed systems that run on these networks, even if these systems are designed to use secure channels (e.g., Secure Sockets Layer (SSL)) for communications.

Due to the importance of this problem, several schemes to mitigate, detect and prevent ARP attacks have been proposed, but each has its limitations. In this paper, we present a scheme that is able to successfully prevent ARP cache poisoning attacks through an economic, efficient, and easy to implement scheme. For this reason, we believe it is a viable solution to be used in Small Office, Home Office (SOHO) LANs.

The design presented in this paper uses a specialized server, but it can be easily modified so that all the functionality is implemented at a low-end switch/router using OpenWrt. The Server function is to receive all ARP requests that are transmitted in the local area network, and respond with a correct ARP answer (based on a mapping table of $\langle IP, MAC \rangle$ tuples maintained by the server). The switch is configured to block ARP responses, except those coming from the server. The switch also changes the destination of ARP requests, just sending them to the server, which does not permit that its ARP cache to be updated with these petitions. In this way, the malicious replies or requests are blocked, making ARP attacks unsuccessful.

The rest of this paper is organized as follows. Section II illustrates the problem considered in this paper and provides a comparison of the characteristics and features of existing schemes to prevent/detect this problem. Section III details the design of our solution. Section IV details the design of our solution outlines the implementation of our scheme. Section V describes the tests that were performed to prove the effectiveness of the solution. Finally, in Section VI we conclude.

II. Problem Definition

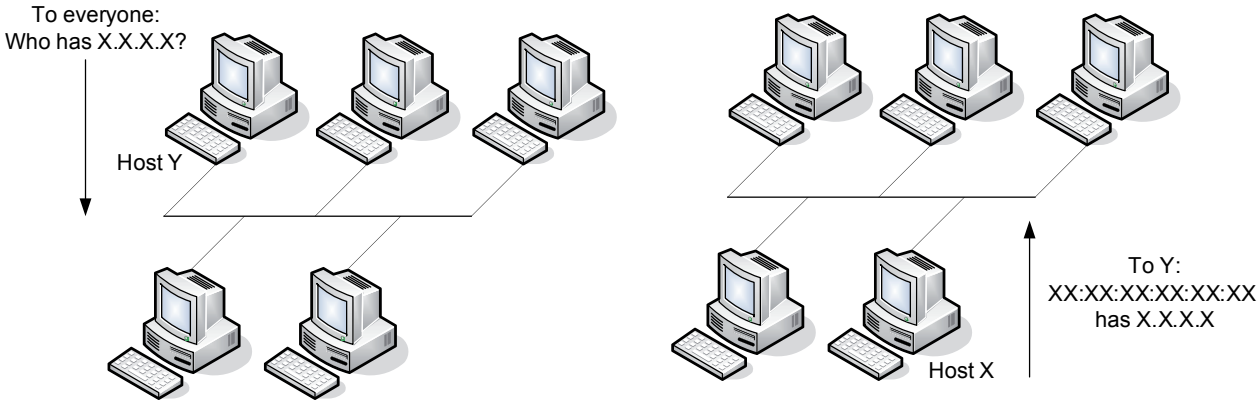
A. Address Resolution Protocol

The Address Resolution Protocol (ARP) is used by computers to map network addresses (IP) to physical addresses (MAC). This protocol is crucial in LAN communications, as each frame that leaves a host must contain a destination MAC address. ARP can be used with any network layer protocol, but since IP is currently the most commonly used network protocol, we will talk about IP addresses in the rest of this paper. Given the IP address of a host, ARP is used to obtain the host's MAC address so that the frame can be delivered on the network.

ARP is a simple protocol that works as follows (see Fig. 1). First, the host that wants to learn the MAC address of another

978-1-4244-4550-9/09/$25.00 ©2009 IEEE

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.



(a) Step 1: Broadcast ARP request

(b) Step 2: Unicast ARP reply

Fig. 1. ARP request/reply protocol

host, broadcasts an ARP request on the network "Who has IP x.x.x.x? Tell MAC yy:yy:yy:yy:yy:yy.". All the other hosts in the LAN receive the request. The host with the given IP answers back in a unicast ARP reply "I have IP x.x.x.x. My MAC is xx:xx:xx:xx:xx:xx.". The host that issued the request caches the $\langle IP, MAC \rangle$ pairing in a local ARP cache (ARP table) so that it does not have to make the same request in the near future. To accommodate for hosts that come and go and for dynamic IP address assignments, the ARP cache entries expire (typically after 20 minutes). Some operating systems reset the expiration timer of ARP entries every time they use an entry [5].

ARP is a stateless protocol by design. When an ARP reply is received, the host updates its ARP cache even if the host had not issued a corresponding ARP request earlier. Note that the $\langle IP, MAC \rangle$ mapping received in the ARP reply should be used to update the ARP cache, only if that sender's IP address is already in the table [1]. As a performance improvement, some operating systems (e.g., Linux and Windows) cache replies received from hosts whose IP addresses were not previously in the ARP table [6].

There are a couple of special cases of ARP requests/replies that do not work as described above: proxy ARP and gratuitous ARP. Proxy ARP [7] was designed to allow seamless implementation of transparent subnet gateways. Using proxy ARP, a host can contact another host on the other side of a router, even if the original host does not have a default gateway configured. The basic idea is that the router is configured to reply to ARP requests on behalf of the hosts on the other side of the router. When the original host receives the reply, it is not aware that the MAC address it is receiving does not belong to the destination host, but to the interface of the router on the current network.

Gratuitous ARP are non-solicited ARP messages sent by hosts, directed to their own IP addresses. Hosts commonly use this type of messages when joining a network with a dynamically assigned IP address. These hosts use gratuitous ARP to confirm that the newly assigned IP address is not currently in use by another host in the network.

B. ARP Cache Poisoning and Spoofing

ARP is a trusting protocol and was not designed to cope with malicious hosts. There are several ways in which a malicious host can make an unsuspecting host modify its ARP cache to add/update an entry with an $\langle IP, MAC \rangle$ mapping to enable the attacker to impersonate another host, perform man-in-the-middle attacks, gain access to sensitive information and perform Denial of Service (DoS) attacks. When a host adds an incorrect $\langle IP, MAC \rangle$ mapping to its ARP cache, this is known as *ARP cache poisoning* (or simply *ARP poisoning*) or *ARP spoofing*. The last terminology refers to the fact that an attacker uses fake or "spoofed" ARP packets to poison an ARP cache.

In an ARP cache poisoning attack, the attacker sends ARP replies with fake $\langle IP, MAC \rangle$ mappings, in an attempt to poison the ARP cache of other host(s) on the network. For example, if the attacker wants to impersonate host X so that host Y sends data destined to X to the attacker instead, the attacker can send an ARP reply indicating that the host with the IP x.x.x.x has the MAC mm:mm:mm:mm:mm:mm (the MAC address of the attacker). Since ARP is an stateless protocol, the receiver will gladly update its ARP cache with the $\langle IP, MAC \rangle$ pairing received. Furthermore, some operating systems may even update static cache entries with information received from unsolicited ARP replies [8].

Even if ARP is configured to be stateful, an attacker can still perform an ARP spoofing attack by sending a fake ICMP (Internet Control Message Protocol) echo request to Y indicating it comes from X, but using the MAC address of the attacker. Depending on the implementation, the operating system can either use the $\langle IP, MAC \rangle$ pairing inferred from the received Ethernet frame and ICMP packet, or it can issue an ARP request to learn the mapping (before sending the ICMP echo reply). In the latter case, the attacker can instantly reply to the ARP request, poisoning the cache.

The ARP poisoning attacks just described are often used as part of other serious attacks:

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

TABLE I
MECHANISMS USED TO SECURE ARP

Scheme	Mechanism
arpwatch [9]	Sniffing and heuristic rules to generate alarms
Intrusion Detection Systems [10]	Sniffing and heuristic rules to generate alarms
Carnut et al. [11]	Sniffing and SNMP heuristics to generate alarms
ARP-Guard [12]	Sniffing and heuristics to generate alarms
ebtables [11], [13]	Heuristic ingress rules in Linux-based switch/bridge
Port security [14]	Binding MAC addresses to switch ports
Solaris' approach [6], [15]	Heuristics used to block ARP replies at receiver
Tripunitara et al. [16]	Heuristics used to block ARP replies at receiver
Static entries in cache [15]	Static ARP cache entries
Anticap [17]	Heuristics used to block ARP replies at receiver
Gouda et al. [18]	Secure server resolves queries
S-ARP [6]	Signed ARP replies
TARP [19]	Centrally issued tickets authenticate $\langle IP, MAC \rangle$ associations
Goyal et al. [20]	Digital signatures and one time passwords based on hash chains to authenticate $\langle IP, MAC \rangle$ associations
Pansa & Chomsiri [21]	A NIC has a certificate and private/public key; DHCP server authenticates hosts and "DHCP Request-MAC" queries
S-UARP [22]	DHCP+ server keeps $\langle IP, MAC \rangle$ associations and answers S-UARP (unicast) requests
Dynamic ARP Inspection [23]	Switch learns $\langle IP, MAC \rangle$ bindings from DHCP snooping and ARP ACLs, and drops invalid ARP packets

- *DoS attacks:* An attacker can poison an ARP table of a host so that every packet that the host sends is sent to the attacker instead of its real destination. In this way, the attacker blocks the communication from the host being attacked.
- *Host impersonation:* Instead of just dropping the packets received from the host being attacked, the attacker can respond, impersonating any host in the network.
- *Man-in-the-middle (MITM) attacks:* By spoofing two hosts in the network at the same time, an attacker can silently sit in between the two hosts so that they think they are communicating with each other. This attacker is then able to listen to all the traffic sent in both directions. This attack can also be performed between any host in the LAN and an outside host, as the attacker can perform the attack between the host and the default gateway. With a MITM attack, the attacker can gain access to sensitive information (e.g., passwords) or he/she can even modify the data being sent, compromising the data's integrity.
- *MITM attacks on encrypted connections:* The attacker can even sit in between a secure connection (e.g., SSL or SSH). The application (e.g., the Web browser) will likely warn the user that the certificate provided is not valid, but many users tend to ignore this kind of warnings. Furthermore, a bug in some versions of Internet Explorer enable attackers to hijack SSL sessions without the browser displaying a warning [8].

The significance of the vulnerability to these attacks becomes greater if we realize that even an outsider can perform an ARP attack on our LAN. All he/she needs to do is compromise a host in our LAN by exploiting a vulnerability in the host's operating system or in one of its installed services. Then, the outsider can use ARP spoofing techniques to learn passwords, among other things. Furthermore, it is important to keep in mind that such an attacker does not need to be experienced. *Script kiddies* can easily perform ARP attacks using widely available tools [2], [3].

Note that these attacks are not restricted to Ethernet networks. Hosts using other Layer 2 protocols like 802.11 are also susceptible to ARP poisoning attacks.

Finally, there is an special type of attack that sometimes is classified as an ARP attack but sometimes it is not. This attack can be referred to as a *MAC spoofing* attack. In this attack, the malicious host clones the MAC and IP addresses of another host and impersonates this host when the host being impersonated is down or cannot respond (e.g., it is suffering from a DoS attack). This type of attack can be easily blocked by using *port security* (described in Section II-C).

Note that these attacks are possible because ARP does not provide a way to verify the authenticity of the requests/replies.

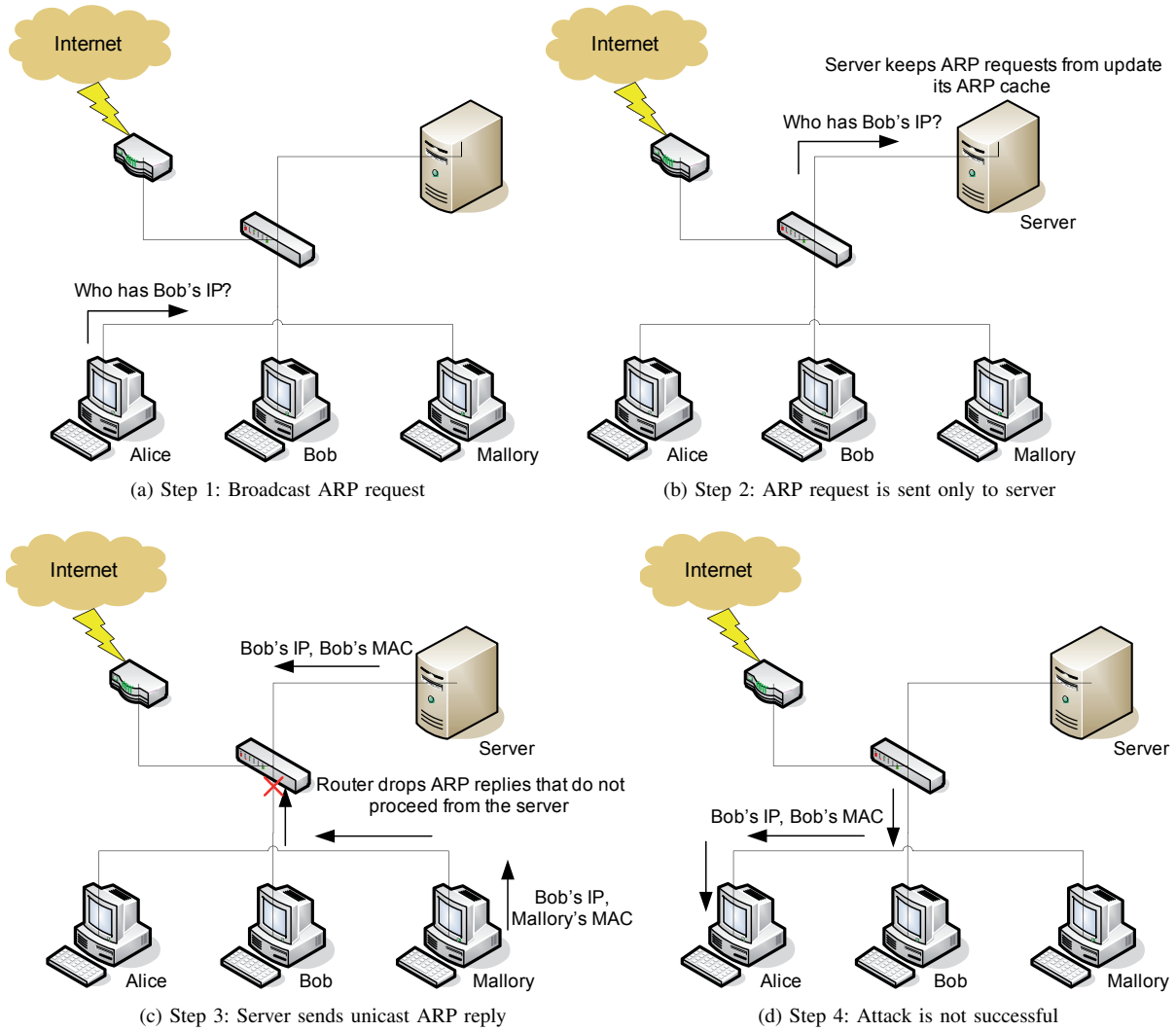
C. Existing Solutions

The existing schemes for adding security to ARP in order to prevent/detect the problems defined in Section II-B can be divided in schemes that concentrate in the detection of the problem, those that mitigate the problem, and those that aim at preventing or blocking the attacks. An earlier study [24] showed that no existing scheme offers an ideal solution for the problem of ARP attacks. Table I summarizes these schemes. For a detailed description of each scheme, read [24].

Out of the detection schemes, the one proposed by Carnut et al. [11] seems ideal in terms of reducing false positives, but requires a complex setup, and the authors have not made their software widely available yet. ARP-Guard [12] seems to be a good alternative, but it is not free. On the other hand, tools like arpwatch [9] and Snort [10] are free, but tend to generate a high-number of false positives, increasing the work of the network administrator.

Out of the mitigation schemes, alternatives like port security [14] and good ARP implementations at the operating system level [15] are straightforward to use and have no significant impact on performance, but are only able to block some types of ARP attacks. The middleware approach proposed by Tripunitara et al. [16] is not practical, as it requires changes on all the hosts in the network, and furthermore, no implementation is widely available for download. Finally, the

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.



(a) Step 1: Broadcast ARP request

(b) Step 2: ARP request is sent only to server

(c) Step 3: Server sends unicast ARP reply

(d) Step 4: Attack is not successful

Fig. 2. Proposed scheme for blocking ARP attacks

proposed `ebtables` filtering [13], [15] can be applied only in very specific network setups, and the rules have to be hand-crafted by the administrator.

The prevention/blocking schemes are the most ambitious ones, but either they require complex installations that do not scale well [6], [18]–[21], are limited to static networks (like the static ARP cache tables approach described in [15]), require changes on all hosts on the network [6], [17]–[22], or require the use of expensive high-end switches [23]. Furthermore, the schemes that depend on cryptographic techniques tend to reduce significantly the performance of address resolution, making their adoption impractical [6], [18], [19]. Out of all these schemes, we believe that the Dynamic ARP Inspection (DAI) mechanism [23] is the best because it is non-intrusive, requires no changes on the network hosts, and does not slow down ARP. Unfortunately, the high costs of the switches with DAI available makes this solution prohibitive for many companies.

Our solution is similar to Cisco's DAI, but can be implemented in SOHOs using low-end networking equipment running the OpenWrt firmware.

III. Design

We based our design on the requirements for an ideal solution identified in [24].

In our scheme, a switch blocks all ARP messages, except the ones coming from a specialized server (or from the switch itself, if the server program is set to run on the device) (see Fig. 2). The scheme consists of two elements: (1) a computer that responds ARP requests, which will be called the server from now on, and (2) a switch/router with the OpenWrt firmware installed.

In our proof of concept scenario, the router is also the DHCP server of the network. Through a program that the router runs continuously, it resends DHCP messages to the server. Using `iptables` and its ULOG extension, DHCP frames are

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

logged to a named pipe[1]. Our program reads DHCP messages from the pipe and forwards them to the server.

The server is responsible for updating the ARP cache on the system and this is possible through the information contained in DHCP frames. The `update_arp_cache` program executes this function. The server listens to ARP requests on the network and responds with an $\langle IP, MAC \rangle$ mapping from the ARP cache, being the only one who can do this. The program responsible for the replies is called `send_arp_reply`. In the router, `ebtables` blocking rules are used to block all ARP replies except those coming from the server. Since fake ARP requests could also poison the ARP cache of hosts on the LAN, the router redirects them to the server through the use of `ebtables` rules, so only the server can receive them. The server does not allow these requests to update its ARP cache, to avoid having its cache poisoned. This is achieved with the use of `arptables`.

Note that the program that the router runs to forward DHCP frames is unnecessary if the scheme is implemented in the router.

If the DHCP server is not located in the router but on another computer, the DHCP frames should be forwarded to the server that responds the ARP requests so it can maintain its $\langle IP, MAC \rangle$ mappings, and this could be achieved with `iptables` rules.

If static IPs are used, then this have to be added manually to the server's ARP cache.

A. Blocking and forwarding rules

Using `iptables` and its ULOG extension, the DHCP frames are sent to a named pipe, which is used by the program that forwards the frames to the server.

The `iptables` rules used to save DHCP frames received by the router inside the named pipe are listed in Table II (see rules in rows 1 through 4). Rule 3 works by identifying server through vlan3. The configuration of the router's vlans was modified for this purpose.

The rules in row 4 are used for preventing malicious ARP requests from poisoning the ARP caches of the hosts in the LAN. ARP requests that are broadcasted and those which destination is unicast, are forwarded to the server only.

To avoid malicious ARP messages from poisoning the server's cache, these are dropped with rule 5.

B. Algorithm

Finally, the server needs to be able to learn the correct $\langle IP, MAC \rangle$ mappings (accomplished through the `update_arp_cache` program) and to use these mappings to correctly reply the ARP requests it receives (accomplished through the `send_arp_reply` program).

The algorithm for the `update_arp_cache` program, described in Algorithm 1. The algorithm for the `send_arp_reply` program is described in Algorithm 2.

[1]A *named pipe* is an interprocess communication method based on the traditional Unix pipes.

TABLE II
BLOCKING AND FORWARDING RULES

	Rule	Location	Purpose
1	`iptables -t mangle -A INPUT -s LAN_IP -j MARK --set-mark 100` `iptables -A INPUT -m mark --mark 100 -p udp --sport 68 -j ULOG`	router	Mark and send DHCP frames received by router to named pipe.
2	`iptables -t mangle -A OUTPUT -d LAN_IP -j MARK --set-mark 200` `iptables -A OUTPUT -m mark --mark 200 -p udp --dport 68 -j ULOG`	router	Mark and send DHCP frames sent by router to named pipe.
3	`ebtables -A FORWARD -p ARP --arp-opcode 2 -i ! vlan3 -j DROP`	router	Drop ARP replies that were not sent by server.
4	`ebtables -t nat -A PREROUTING -p ARP --arpopcode 1 -s ! SERVER_MAC -d ff:ff:ff:ff:ff:ff -j dnat --to-destination SERVER_MAC` `ebtables -t nat -A PREROUTING -p ARP --arpopcode 1 -s ! SERVER_MAC -d ! ff:ff:ff:ff:ff:ff -j dnat --to-destination SERVER_MAC`	router	ARP requests (broadcast and unicast) are sent only to server.
5	`arptables -A INPUT --opcode 1 -j DROP`	server	Keep ARP requests from updating server's ARP cache.

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 update_arp_cache

```
1: if DHCP packet is received then
2:   if message type is DHCPACK then
3:     IP ← 'your IP address' field value
4:     if IP ≠ server's IP then
5:       MAC ← 'client's hardware address' field value
6:       Add ⟨IP, MAC⟩ to server's ARP cache
7:       Add ⟨IP, MAC⟩ to backup file
8:     end if
9:   else if message type is DHCPRELEASE then
10:    IP ← 'your IP address' field value
11:    if IP ≠ server's IP then
12:      Remove ⟨IP, ?⟩ from server's ARP cache
13:      Remove ⟨IP, ?⟩ from backup file
14:    end if
15:  else if message type is DHCPDECLINE then
16:    IP ← 'requested IP address' options field value
17:    if IP ≠ server's IP then
18:      Remove ⟨IP, ?⟩ from server's ARP cache
19:      Remove ⟨IP, ?⟩ from backup file
20:    end if
21:  else
22:    NOOP
23:  end if
24: end if
```

Algorithm 2 send_arp_reply

```
1: if ARP message is received then
2:   if operation field = REQUEST then
3:     TPA ← Target Protocol Address field value
4:     Create an ARP REPLY message
5:     Sender Protocol Address field ← TPA
6:     if TPA = server's IP address then
7:       SHA ← server's MAC address
8:     else
9:       Find ⟨TPA, MAC⟩ mapping in ARP cache
10:      if ⟨TPA, MAC⟩ does not exist then
11:        return //No response is sent
12:      end if
13:      SHA ← MAC address in ⟨TPA, MAC⟩
14:    end if
15:    Sender Hardware Address field ← SHA
16:    Send ARP response to requesting host
17:  end if
18: end if
```

IV. Implementation

This section explains the implementation of the scheme described in Section III.

A. Hardware

The choice of hardware was based on which manufacturers were providing source code for their firmware. The router that we used was the WRTSL54GS manufactured by Linksys.

The Linksys WRTSL54GS is a Wi-Fi capable residential gateway from Linksys [25]. The device is capable of sharing Internet connections among several computers via 802.3 Ethernet and 802.11b/g wireless data links. The WRT54G (and variants WRT54GS, WRT54GL, and WRTSL54GS) is notable for being the first consumer-level network device that had its firmware source code released to satisfy the obligations of the GNU GPL. This allows programmers to modify the firmware to change or add functionality to the device.

Various development projects have been done on the WRT54GL's original firmware source code resulting in many third party open source firmware code bases. Some of the third party open source firmware code bases are: DD-WRT, FreeWRT and OpenWrt.

We chose OpenWrt Whiterussian RC6 because it offers greatest stability than other versions, and it also presents support for `ebtables`.

B. OpenWrt firmware

OpenWrt is a Linux distribution, which has its own writable file system and package management system [26]. It contains more than 100 software packages and provides facilities in the source code to include more add-on packages. The OpenWrt source code is oriented towards developers by providing a development kit called OpenWrt Software Development Kit (OpenWrt SDK), which enabled us to develop the program that forwards DHCP frames from the router to the server of our scheme.

C. Router configuration

Source code for embedded systems cannot be compiled within the embedded system itself. Embedded systems have lower processing power and memory; hence it is not possible to run a compiler like gcc directly on a router. Instead, we need to have a cross compiler on our development machine that will compile the source code and produce the binary for our router. The tool used was OpenWrt SDK. After the development and compilation of the source code made to forward DHCP frames, the ipkg package generated by SDK is installed in the router.

The `iptables` rules mentioned in Section III-A were added to the file `/etc/init.d/S35firewall` and the rules to drop and forward ARP frames were included in the file `/etc/firewall.user`.

D. Server configuration

The programs `update_arp_cache` and `send_arp_reply`, described in Section III-B were developed in C. The programming tools and libraries used

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

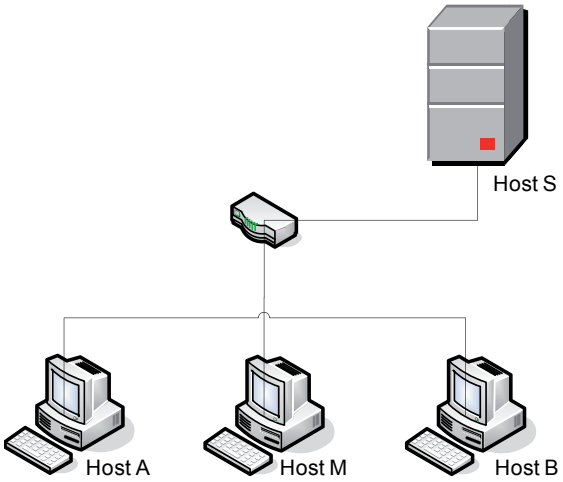


Fig. 3. Attack scenario

were: automake 1.10, gcc 4.1.2, libglib 2.12.11, binutils 2.17, libpcap 0.9.5 y libdumbnet 1.8-1.5.

The `nice` command was used to launch the `send_arp_reply` process with a priority of -20, indicating that its priority is high.

These two programs run permanently, listening to DHCP and ARP messages that arrive and depart from the indicated network interface.

The server rule described in Section III-A was installed by adding it to the file `/etc/rc.local`, so that every time the computer boots, the `arptable` rule is installed.

V. Evaluation

In parallel to this project, ARP attack trees were developed and used to create an in-house tool that tests for all possible ways in which the ARP cache of a host can be poisoned.

This section describes the tests done to poison a host's ARP cache, emulating various attacks with the in-house tool. It also shows how the solution worked.

A. Tests with in-house tool

The setup of the LAN for the attack scenario is as shown in Fig. 3. Host A and host B are computers connected to the router, host S is the server of our scheme. Host M is the attacker.

To perform an ARP cache poisoning attack, the attacker sends 100 spoofed ARP replies to host A with host B's IP address as the source IP address and the attacker's MAC address as the source MAC address.

With our design when the attacker sends the spoofed ARP frames to host A, these frames are dropped by the router, because of the rule that avoids that other computer different from our server, replies ARP requests. The `ebtable` rule checks if the interface from the frame comes is different from the one which is connected our server, if it is true then the ARP reply is dropped. The result of performing this attack against our solution was not successful.

In a different attack, the attacker sends 100 spoofed ARP requests to host A with host B's IP address as the source IP address and the attacker's MAC address as the source MAC address. The PREROUTING ebtables rules redirect the ARP requests to the server, which keeps its ARP cache to be updated. All the other hosts on the LAN can not see these malicious frames. The implementation of this attack was not successful.

The next attack was to send fake ICMP Echo Requests, emitted from the attacker. When sending a request the attacker waits that the host that receives the ICMP, makes an ARP request, and immediately after that, the attacker sends 100 poisoned ARP replies. This attack was also not successful.

The different attacks made by the in-house tool were not successful. All of them were based on spoofed ARP replies and spoofed ARP requests.

B. Tests with Cain & Abel

The same scenario of the previous tests was used for this attack. In this scenario, Host A and Host B want to communicate with each other. The attacker wants to divert the communication between host A and host B to flow through the attacker.

To perform a man-in-the-middle-attack, we used Cain & Abel [4]. One of the ways to perform this attack is by ARP poisoning both victims. The malicious host sends a spoofed ARP reply to Host A with Host B's IP address as the source IP address and the attacker's MAC address as the source MAC address. Similarly the Attacker sends a spoofed ARP reply packet to Host B with Host A's IP address as the source IP address and the attacker's MAC address as the source MAC address. The man in the middle attack was not successful, when our solution was used.

C. Impact on performance

To calculate the impact of our scheme on the performance of the local area network were used as measuring parameters: time and number of packets. No packets were lost due to the implementation of the solution.

The table included in this section compares three scenarios: (1) router with OpenWrt's default configuration, (2) router with VLANs configured (solution not running), and (3) router with solution running.

Note that the second scenario is included because–in order to force the switched packets to be processed by the rules described in Section III-A–we use bridged VLANs which send the packets to the router for processing. We wanted to know what part of the performance impact was due to our solution, and what part was due to the use of VLANs.

1) Transmission time of ARP request-replies: The results were obtained by sending 40 ICMP echo request packets. The results of ten runs of the experiment were averaged, and its results summarized in Table III.

Before each ping we clear the corresponding entry from the ARP cache of the host, so that the host is forced to send

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

TABLE III
PERFORMANCE COMPARISON

	OpenWrt	VLANs	VLANs and solution
Transmission time of ARP request-replies	0.00010	0.00058	0.01489
Packet transmission time (ICMP packets were used)	0.00017	0.00068	0.00149

an ARP request. The `arp` command was used to remove the $\langle IP, MAC \rangle$ mapping.

As it can be seen in Table III, the use of VLANs adds 0.4759 μsecs to the transmission time of an ARP response. This delay is one of the components of the increased ARP response transmission time seen in the solution; however this delay represents only 3.89% of total delay (when our scheme is being used). The remaining percentage is divided by different factors.

To determine the causes of the high time obtained, a thorough analysis of the delays at each step of the process was performed. The analysis showed that the greatest delay occurs on the server, specifically at the `send_arp_reply` program.

`send_arp_reply` was implemented in C, using the `dumbnet` and `libpcap` libraries. After timing each function of the program, it was determined that the poor performance is due to inefficiencies in the `libpcap` library. In a study performed by Weigle and Feng [27], they showed that:

> *Libpcap suffers from efficiency problems pandemic to implementations of traffic collection at user level. They must ask the operating system to perform the required packet copy in the network stack (for transparency), which can double the time required to process a packet.*
>
> *The exact method used by libpcap and other tools varies by operating system, but always involves a context switch into kernel mode and a copy of memory from the kernel to the user level library. This call-and-copy approach is repeated for every packet observed in Linux.*

Unfortunately the inefficiency of `libpcap` was not predicted at the beginning of the implementation.

The high values in this metric could be avoided by replacing our current implementation with a low-level implementation running in kernel mode.

On the upside, ARP traffic represents a small percentage of the traffic of a network (for example, at a production network in the Operations Department at Telmex in Ecuador, ARP traffic constituted approximately 0.06% of total traffic, so the delay should not impact performance significantly; traffic was measured using the free tool called `ntop`[2]).

2) Packet transmission time: The results were obtained by sending 40 ICMP echo request packets, without resetting the ARP cache of the host for each packet. Table III summarizes the results of ten runs of the experiment. The results confirm that the delay introduced by the use of the solution is significantly smaller for traffic that excludes ARP responses.

[2]http://www.ntop.org

VI. CONCLUSIONS

The security problems that the use of ARP introduce in a local area network (LAN) may create vulnerabilities to the distributed systems that run on these networks, even if these systems use secure channels.

Due to the severity of this problem, several schemes to mitigate, detect and prevent ARP attacks have been proposed, but each has its limitations.

This paper presented a scheme that can be used to block ARP attacks in SOHO LANs. The scheme blocks all ARP attacks, but comes at a cost of reduced performance, specially for ARP responses.

It is expected that from a small proof of concept as our study, a mechanism can be developed to be applied for future networks to prevent further attacks that can occur as a result of an ARP poisoning.

ACKNOWLEDGMENT

This work was financed by a grant from the VLIR-ESPOL program (http://www.vlir.espol.edu.ec).

REFERENCES

[1] D. Plummer, "An ethernet address resolution protocol," Nov. 1982, RFC 826.
[2] D. Song, "dsniff," (accessed 28-July-2009). [Online]. Available: http://www.monkey.org/~dugsong/dsniff/
[3] S. Buer, "Arpoison," (accessed 28-July-2009). [Online]. Available: http://arpoison.sourceforge.net
[4] M. Montoro, "Cain & abel," 2009, (accessed 28-July-2009). [Online]. Available: http://www.oxid.it/cain.html
[5] R. W. Stevens, *TCP/IP Illustrated, vol 1*. Addison Wesley, 2001.
[6] D. Bruschi, A. Ornaghi, and E. Rosti, "S-ARP: A secure address resolution protocol," in *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*, Dec. 2003.
[7] S. Carl-Mitchell and J. Quarterman, "Using ARP to implement transparent subnet gateways," Oct. 1987, RFC 1027.
[8] T. Demuth and A. Leitner, "ARP spoofing and poisoning: Traffic tricks," *Linux Magazine*, vol. 56, pp. 26–31, Jul. 2005.
[9] L. N. R. Group, "arpwatch, the ethernet monitor program; for keeping track of ethernet/ip address pairings," (accessed 28-July-2009). [Online]. Available: ftp://ftp.ee.lbl.gov/arpwatch.tar.gz
[10] Snort Project, The, *ARP Spoof Preprocessor*, Apr. 2009, p. 68, (accessed 28-July-2009). [Online]. Available: http://www.snort.org/assets/82/snort_manual.pdf
[11] M. Carnut and J. Gondim, "ARP spoofing detection on switched Ethernet networks: A feasibility study," in *Proceedings of the 5th Simpósio Segurança em Informática*, Nov. 2003.
[12] "ARP-Guard," (accessed 28-July-2009). [Online]. Available: http://www.arp-guard.com
[13] B. D. Schuymer, "ebtables: Ethernet bridge tables," Mar. 2006, (accessed 28-July-2009). [Online]. Available: http://ebtables.sourceforge.net
[14] C. Schluting, "Configure your Catalyst for a more secure layer 2," *Enterprise Networking Planet*, Jan. 2005, (accessed 28-July-2009). [Online]. Available: http://www.enterprisenetworkingplanet.com/netsecur/article.php/3462211
[15] S. Whalen, "An introduction to ARP spoofing," *2600: The Hacker Quarterly*, vol. 18, no. 3, Fall 2001, (accessed 28-July-2009). [Online]. Available: http://servv89pn0aj.sn.sourcedns.com/~gbpprorg/2600/arp_spoofing_intro.pdf

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.

[16] M. Tripunitara and P. Dutta, "A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning," in *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC '99)*, Dec. 1999.

[17] M. Barnaba, "anticap," (accessed 17-April-2006). [Online]. Available: http://www.antifork.org/viewcvs/trunk/anticap

[18] M. Gouda and C.-T. Huang, "A secure address resolution protocol," *Computer Networks*, vol. 41, no. 1, pp. 57–71, Jan. 2003.

[19] W. Lootah, W. Enck, and P. McDaniel, "TARP: Ticket-based address resolution protocol," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, no. 15, pp. 4322–4337, 2007.

[20] V. Goyal and A. Abraham, "An efficient solution to the ARP cache poisoning problem," in *Proceedings of the 10th Australasian Conference on Information Security and Privacy (ACISP '05), published in Lecture Notes in Computer Science (LNCS 3574)*, Jul. 2005, pp. 40–51.

[21] D. Pansa and T. Chomsiri, "Architecture and protocols for secure lan by using a software-level certificate and cancellation of ARP protocol," in *Third International Conference on Convergence and Hybrid Information Technology*, vol. 2, Nov. 2008, pp. 21–26.

[22] B. Issac, "Secure ARP and secure DHCP protocols to mitigate security attacks," *International Journal of Network Security*, vol. 8, no. 2, pp. 107–118, 2009.

[23] Cisco Systems, *Configuring Dynamic ARP Inspection*, 2006, ch. 39, pp. 39:1–39:22, Catalyst 6500 Series Switch Cisco IOS Sofware Configuration Guide, Release 12.2SX.

[24] C. L. Abad and R. I. Bonilla, "An analysis of the schemes for detecting and preventing ARP cache poisoning attacks," in *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*, Jun. 2007, p. 60.

[25] Wikipedia, "Linksys wrt54g series — wikipedia, the free encyclopedia," 2009, (accessed 28-July-2009). [Online]. Available: http://en.wikipedia.org/w/index.php?title=Linksys_WRT54G_series&oldid=304767428

[26] "Openwrt," 2009, (accessed 28-July-2009). [Online]. Available: http://openwrt.org/

[27] E. Weigle and W. chun Feng, "TICKETing high-speed traffic with commodity hardware and software," in *Proceedings of the Third Annual Passive and Active Measurement Workshop (PAM2002)*, 2002, pp. 156–166.

Authorized licensed use limited to: Hochschul- und Kreisbibliothek Bonn-Rhein-Sieg. Downloaded on November 02,2020 at 12:48:37 UTC from IEEE Xplore. Restrictions apply.