Published in the Proceedings of the 1st Indian Conference on Computational Intelligence and Information Security, 2007 (ICCIIS, 07), pp:22-25.

Middleware Distributed Approach to Synchronous Detection of ARP Cache Poisoning

Md. Sadek Ferdous[1] **and Farida Chowdhury**[2]

Abstract—**ARP cache poisoning based attack has been one of the most successful attack methods for years inside a LAN. There are a few solutions to detect and sometimes prevent an ARP based attack but they have some restrictions. In this paper we present a novel way to detect ARP cache poisoning inside a LAN. We propose a middleware and synchronous solution that has to be implemented in a distributed approach. Our solution requires no need have access and change to any Operating System code, but needs to be activated in timely manner and more than one host inside a LAN will be utilized to detect ARP cache poisoning based attack.**

Key Words: ARP, ARP Cache Poisoning, Middleware, Synchronous, Distributed.

I. Introduction

Address Resolution Protocol (ARP) is the mandatory protocol to be followed to transfer data inside a LAN. According to the TCP/IP model, ARP is used by a host inside a Local Area Network (LAN) to find the Data Link Layer address providing a Network Layer address [2, 4, 13]. According to the context of this paper, we assume a Network Layer address is an IP address, and a Data Link Layer address is an Ethernet address or Media Access Control (MAC) address. In theory, MAC address is a globally unique and totally unchangeable value that is hard coded burned into its Network Interface Card (NIC) by the manufacturer [5]. In the other hand, Internet Protocol (IP) is protocol used by applications blind to whatever network technology operating underneath it [5]. Each host inside a network should have a unique IP address to communicate with each other. IP address is virtual and assigned via software. ARP is used to map between MAC address and IP address.

To smoothly perform the mapping between MAC and IP address, every host in a LAN maintains a local table called ARP cache. Due to the flaw in the design of TCP/IP layer, this ARP cache can be deliberately and maliciously altered. This act of changing the ARP cache deliberately and maliciously is known as ARP cache poisoning [12, 14]. Based on such ARP cache poisoning several very effective attacks such as Sniffing (Man in the Middle, MAC Flood), DoS, Session Hijacking, etc can be generated.

Following this introduction, this paper is organized as follows: Section 2 discusses detailed ARP operation and how ARP cache can be poisoned and how ARP cache poisoning can be used to generate attacks inside a LAN. Section 3 provides a brief description of some techniques to detect and/or prevent ARP cache poisoning. Section 4 discusses some of the works related to detect ARP cache poisoning. Section 5 elaborates our proposed solution. We conclude in section 6.

II. ARP & ARP Cache Poisoning

To make communication between two hosts in a LAN two pairs of addresses are necessary [2, 4, 13]. One pair contains source IP address and source MAC address (MAC address of the source host) and the other pair contains destination IP address and destination MAC address (MAC address of the destination host). All the data are encapsulated in the Ethernet frame with these two pairs before they are transmitted. Before this encapsulation takes place, sender host needs MAC address of the destination host. Given an IP address, ARP can resolve the MAC address of the corresponding host dynamically. A figure of an ARP frame is given in Figure 1.

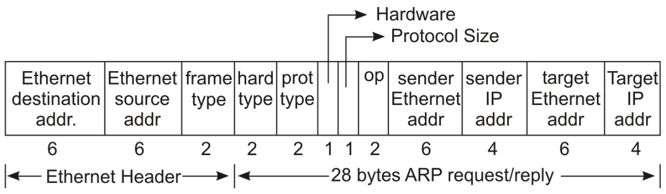

Fig. 1: Format of ARP Request or Reply Packet When Used on an Ethernet

ARP is based on two functions: Request and Reply. Same ARP frame with different configurations are used for Request and Reply functions. In the Request function, an ARP packet is broadcast over the network which contains, with other information, Sender's IP address, Sender's MAC address and Destination IP address. If there is a host which has the IP address same as the Destination IP address, it responds with an ARP Reply frame with its MAC address.

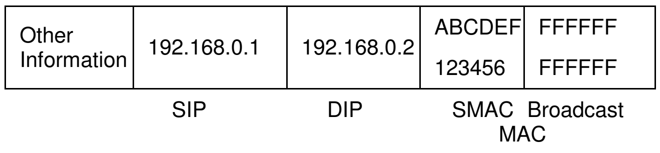

Fig. 2: Request ARP Packet

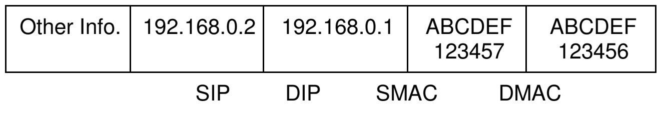

Fig. 3: Reply ARP Packet

[1]Dept. of Information & Communication Technology, Metropolitan University, Sylhet, Bangladesh. E-mail: sferdous@metrouni.edu.bd

[2]Dept. of Computer Science & Engineering, Shah Jalal University of Science & Technology, Sylhet, Bangladesh. E-mail: farida-cse@sust.edu

Let's illustrate the scenario with some examples. Suppose in a LAN there are some hosts A to F. A communication has to be established from host A to host B with IP address 192.168.0.1 and 192.168.0.2 respectively and with MAC address of ABCDEF123456 and ABCDEF123457 respectively. A knows the IP address of B but has no idea about the MAC address of B. So it uses ARP to resolve the MAC address of B. It builds an ARP request packet like the figure 2.

In figure 2, SIP means Source IP, DIP means Destination IP and SMAC means Source MAC. This packet is broadcast over the LAN. Every host in the LAN checks the destination IP address field. Only host B finds that destination IP address of this packet matches with its IP address. So it responds by sending an ARP reply packet with its MAC address. The reply ARP packet looks like the figure 3.

In figure 3, DMAC means Destination MAC. After receiving the ARP reply packet IP and MAC addresses of source and destination host are known. Now an Ethernet frame is built with these addresses and communication takes place.

A. ARP Cache

For the efficient operation of the ARP, a table called ARP cache is maintained in each host [11, 13]. This table caches the recent mapping from IP address to MAC address. Each row of this table has two columns. One column contains the IP address and the other column contains the corresponding MAC address. Such ARP cache can be built with the following ways:

1 *Statically*: In the static ARP cache, the table has to be built with manual entry. User/administrator of a LAN builds up the ARP cache by manually inserting the IP address for each host of the LAN with its corresponding MAC address.

2 *Dynamically*: In the dynamic ARP cache, ARP table is built dynamically. When a host needs to know the MAC address of another host, it sends an ARP request packet and when it receives an ARP reply packet, it caches the result in the ARP cache. So when the next time the same host needs to know the MAC address of the same other host, it first checks its ARP cache to find out the MAC address thus eliminates the need for broadcasting an ARP request packet to resolve the MAC address [9]. In the same way when a host receives an ARP request packet, that may or may not be destined for it, it caches the source IP and MAC address from ARP request packet.

B. ARP Cache Poisoning

ARP cache poisoning is the act of changing the ARP table deliberately and dynamically by introducing false mapping between IP and MAC address in a selected host or all hosts in a LAN through some predefined manners. Then using this poisoning different modes of attack such as Sniffing (Man in The Middle (MiTM) attack, MAC Flooding Attack), Denial of Service Session Hijacking, etc can be generated [5, 10, 14].

Such poisoning can be furnished in the following ways [7, 8]:

Unsolicited Response: If a host receives an ARP reply packet by some other host it will update its ARP Cache without checking the validity of the ARP reply, that means the receiving host will not check whether that ARP reply is generated for an ARP request. So an attacker has to send out an ARP reply packet with false mapping information to one/any number of hosts that the attacker wants to victimize by poisoning the ARP Cache.

ARP Request: Sometimes an attacker can poison the ARP cache of a host by using a legitimate ARP response. In this case the attacker will wait for an ARP request packet generated by a host. So when the first host will generate an ARP request packet to resolve the MAC address of the second host in the LAN, the attacker will involve in a race condition. The second host will simply generate a legitimate ARP reply packet with legitimate value. The attacker also will generate an ARP reply packet with spurious mapping. The reply packet that will be received later by the first host will cause the first host to alter the ARP cache according to its information. So if the attacker can win over the race condition, the ARP cache of the first host will be poisoned.

ARP Response: Sometimes an attacker can poison the ARP cache of a host by using a legitimate ARP response. In this case the attacker will wait for an ARP request packet generated by a host. So when the first host will generate an ARP request packet to resolve the MAC address of the second host in the LAN, the attacker will involve in a race condition. The second host will simply generate a legitimate ARP reply packet with legitimate value. The attacker also will generate an ARP reply packet with spurious mapping. The reply packet that will be received later by the first host will cause the first host to alter the ARP cache according to its information. So if the attacker can win over the race condition, the ARP cache of the first host will be poisoned.

III. Prevention/Detection of ARP Cache Poisoning

In this section we provide some of the defenses against ARP cache poisoning. ARP was never designed with security in mind. Due to design constraint in the TCP/IP protocol suite and implementation constraint in various platforms, there is no universal defense against ARP cache poisoning. But some steps can be applied to prevent/detect it [3, 5, 6]. Those are elaborated in the following subsections:

A. Static ARP Tables

The most straightforward method to prevent ARP cache poisoning is use static ARP cache. Almost every platform supports static ARP cache in the ARP implementation. Static ARP means manual entry of IP and MAC address pair into each host and neither ARP request is generated nor an ARP reply is honored. So ARP cache can't be altered dynamically using the techniques elaborated in the section 2.2.2. This almost secures a network from ARP cache poisoning. The drawback is that it is almost impossible to implement in large LAN. That's because every device that is added to the network has to be manually added to the ARP script or entered into each machine's ARP table which is very hard to maintain. But for a small network this technique successfully eliminates the possibility of ARP cache poisoning in almost every platform.

B. Using a Switch with Port Securit

Some high-end switches come with an advanced feature known as Port Security/MAC Binding/Port Binding. Port Security enables a switch to allow only one MAC address for each port and this configuration can be only by a network administrator. This prevents MAC/ARP table of the switch to be altered. This feature prevents attacker from changing the MAC address of their machine or from trying to map more than one MAC address to their machine. It can often help prevent ARP-based MITM attack. The main drawback of such feature is that it is very hard to maintain for large LAN. This method also can't be implemented in networks using DHCP.

C. Monitoring Tool

The best option for any network is to defend against ARP cache poisoning is to use monitoring tool. A monitoring tool can be used to monitor the ARP traffic. It will cause an alert when it will find some unusual ARP communication. This kind of monitoring has been regarded as the best option to guard against ARP cache poisoning.

IV. Related Works

Though monitoring tool is being regarded the best option to defend against ARP cache poisoning, very few good monitoring tools can be found in the web.

ARPwatch is a famous ARP monitoring tool [1]. It's a free UNIX program which listens for ARP reply in the network. Upon the ARP reply it builds a table with IP/MAC pairing. If in the next time it finds a discrepancy in the same IP/MAC pairing, it generates some kind of alert. It has some serious restrictions. At the time of building the table, it may be biased by an unsolicited response (a spoofed ARP reply generated by an attacker) and a false pairing of IP and MAC may be accepted by it. So when in the next time it receives an ARP reply with real MAC/IP pairing, it will take it as the spoofed one and will generate a false alarm. ARPwatch can't be trusted if it is implemented in a DHCP enabled network as it'll generate many false alarms.

Mahesh V. Tripunitara and Partha Dutta in [7] has proposed a solution to detect and prevent ARP cache poisoning According to their solution when any host receives an ARP reply, it will be checked if that reply is for any outstanding ARP request. If not, the frame will be dropped. When an ARP request will be received, the validity for the request will be checked. Their solution is almost perfect with one major drawback: to implement their solution kernel level modification in the OS is necessary.

V. Proposed Solution

In this section we propose our solution to detect ARP cache poisoning. At first we'll define the characteristic properties of our solution and then we'll propose an algorithm to define our solution. The characteristic properties of our solution are as follows:

Middleware: Our solution is proposed to be middleware so that implementation based on our solution can act independently without the help of OS and there is no need to access the source code for ARP implementation and network subsystem of the OS.

Distributed: Our solution is proposed to be distributed in nature so that time to time interaction among different agents of different hosts, which will be implemented according to our proposed solution, can be performed.

Synchronous: Our solution is proposed to be Synchronous which involves the necessity of checking the ARP table of a host in a definite time interval.

The key point of the algorithm is that it will create a monitor/agent each for one host which will maintain its own internal table with valid/legitimate IP/MAC mapping. At first IP/MAC mapping will be retrieved from the ARP cache and then they will be verified. If the verification fails, an alarm will be raised and proper step will be taken to remove the false mapping from the ARP cache. For verification, the monitor of different hosts will communicate with each other in a fixed port.

The algorithm of our proposed solution goes below:

Algorithm Detect ARP Cache Poisoning

```
1.  While (TRUE)
2.    Check the ARP cache of the host in a definite time period
3.      If there are some entries in the ARP cache
4.        If there is no entry in the internal table
5.          For every mapping of IP/MAC in each row
            of the ARP cache
6.          Call Procedure BuildInternalTable
            (IP address, MAC address)
7.        Else there are entries in the internal
          table
8.          Match the IP address of each row in the ARP
            cache with each of the IP Address of the
            internal table
9.            If there is a positive match
10.             Check the two MAC addresses of the
                corresponding IP address in the ARP
                cache and internal table
```

```
11.                 If there is a positive match
12.                     Do nothing
13.                 Else If there is negative match
14.                     Potential possibility of ARP cache
                        poisoning. As the internal table is
                        built with valid values, ARP cache
                        contains a spurious mapping of
                        IP/MAC. Raise an alert. Delete the
                        Corresponding IP/MAC mapping
                        from the ARP cache
15.               End If. (end of if of line 11)
16.           Else If there is no match in the internal table
17.               Follow the steps of line 5 and 6
18.           End If (end of if of line 9)
19.       End If (end of if in the line of 4)
20.     Else If there is no entry in the ARP cache
22.         Do nothing.
23.     End If (end of if in the line of 3)
24.   End while
```

Procedure Build Internal Table (IP address, MAC address)

```
1.  Build a special frame with some predefined values and
    send it to the host whose IP/MAC mapping has been found
    as the parameter of the procedure. The frame will
    communicate in a defined port set aside for our purpose.
    The purpose of the special frame is to tell the agent of the
    receiving host to check this IP/MAC mapping. After
    checking the receiving agent will send another packet to
    the sending agent
2.  After receiving .the reply packet, check the answer
3.  If the answer is positive
4.    Make an entry with the corresponding source
      IP/MAC address of the frame in the internal table
5.  Else If the answer is negative
6.        Potential possibility of ARP cache poisoning.
          Raise an alert. Delete the this IP/MAC mapping
          from the ARP cache and do not make any entry
          in the internal table for this IP/MAC mapping
7.  Else If there is no answer frame for a definite period of
    time
8.    Follow the steps of line 6
9   End If
10. Return
```

The main advantage of our algorithm is that it gets off the restrictions of different solutions that we've discussed in Section 4. According to our algorithm, no false alarm will be generated like the ARPwatch. This is because ARPwatch relies on only ARP traffic. But as our algorithm validates every IP/MAC mapping by locally checking in an appropriate host, there is no chance to generate false alarm. Our solution can also be implemented in DHCP enabled network unlike the ARPwatch. Again unlike the solution stated in the [7], no kernel level modification is necessary as our solution just checks the local cache, not the ARP packet. The limitation of our solution is that time to time it will create some extra traffic for the verification purpose in the network.

VI. Conclusions

In this paper we presented the different aspects of Address Resolution Protocol (ARP) cache poisoning problem and attack scenarios that could be generated based on it. Then we discussed some related works in this field. At last we presented a unique algorithm to detect ARP cache poisoning in a unique distributed way inside a LAN. This paper did not deal with any kind of implementation detail. In future, an implementation can be deployed and a comparative performance analysis between this implementation and other related works can be accomplished.

References

[1] ARP information from wikipedia

[2] http://en.wikipedia.org/wiki/Arpwatch

[3] Behrouz A. Forouzan, Data Communications and Networking, TATA McGRAW-HILL, 2004.

[4] Corey Nachreiner, "Anatomy of an ARP Poisoning Attack", WatchGuard Life Security Service, 2003. http://www.watchguard. com/ infocenter/editorial/135324.asp

[5] Plummer, D.C., "An ethernet address resolution protocol or converting network protocol addresses to 48.bit Ethernet address for transmission on ethernet hardware". *RFC 826*, November 1982.

[6] Dillinja, "Address Resolution Protocol: Description, Exploitation and Exploitation Prevention", September, 2003. http://www.Govern mentsecurity.org/archive/t2605.html

[7] Josha Bronson, "Protecting your network from arp spoofing based attacks", http://faculty.capitol-college.edu/~amehri/Articles/How_to_ Protect.pdf.

[8] Mahesh C. Tripunitara and Partha Dutta , "A Middleware Approach to Asynchronous and Backward Compatible Detection and Prevention of ARP Cache Poisoning", Annual Computer Security Applications Conference (ACSAC), 1999.

[9] Beekey, Mike, "ARP Vulnerabilities: Indefensible Local Network Attacks", Black Hat Briefings, 2001. http://www.blackhat.com/ presentations/ bh-usa- 01/MikeBeekey/bh-usa-01-Mike-Beekey.ppt

[10] Neil B. Riser, "Spoofing: An Overview of Some the Current Spoofing Threats", July, 2001. From SANS Reading Room.

[11] Wagner, Robert, "Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks", Practical Assignment GSEC, August, 2001.

[12] Whalen, Sean, Sophie Engle & Dominic Romeo, "An Introduction to ARP Spoofing", April 2007, http://www.node99.org/projects/arpspoof/ arpspoof-slides.ppt.

[13] Manwani, Silky, "ARP Cache Poisoning Detection and Prevention", Dec 2003. http://faculty.capitolcollege.edu/~amehri/Articles/Spoof_ report.pdf.

[14] Stevens, W. Richard, TCP/IP Illustrated, Volume 1: The Protocols, Addison Wesley Longman, Inc. 1994.

[15] Volobuev, Y., "Playing redir games with ARP and ICMP", BUGTRAQ mailing list, September 1997. http://www.goth.net/iceburg/ tcp/arp.games.html.