International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 20 (2018) pp. 14823-14826

A Technique for a Software-Defined and Network-based ARP Spoof Detection and Mitigation

Deepa Balagopal
Research Scholar, Department of Computer Applications,
Karpagam Academy of Higher Education, Eachanari, Coimbatore – 641021, India.
Orcid Id: 0000-0002-1717-5734

X.Agnise Kala Rani
Research Supervisor, Department of Computer Applications,
Karpagam Academy of Higher Education, Eachanari, Coimbatore– 641021, India.
Orcid Id: 0000-0002-1465-0595

Abstract

The centralized Controller of a Software-defined network can be used as an effective medium for ensuring network security. In this paper, the researchers present a restrictive ARP Cache Recovery module which identifies the poisoned ARP packets and takes necessary action to block the attacking device. Apart from issuing alert and blocking the attacker, it also reestablishes the corrupted link between the victim and other devices so as not to affect the functioning of the network.

Keywords: ARP Poisoning, Firewall, POX, SDN, Spoofing.

INTRODUCTION

The Software-defined network paradigm strives to improve the programmability and flexibility of the traditional computer network. It not only provides network virtualization and dynamic network policy management but also attempts to have a greater control over the network devices at a much lesser operational cost. These positives aside, the new architecture necessitates the need to rewrite the old networking software and applications to fit in with the current scenario.

With SDN and its programmability, the network administrator has better control over the security of the network not only from rogue elements outside, but also from within [1]. For instance, a host can fake the identity of someone else and flood the network with spoofed packets thus upsetting its performance and consuming the bandwidth. The presence of such malicious end-hosts can cause chaos to a network. Though traditional networks too have reported similar attacks and have some solutions in place, the researchers believe that SDN can provide a better solution.

When a device that is directly connected to a software-defined network is compromised, for instance, a server or a user workstation, it can pass wrong information about the network topology and the location of target hosts and confuse the Controller so as to take over a target host or traffic of interest [2,3]. These are predominantly attacks that target the Address Resolution Protocol. The programmability of SDN presents ample scope to identify and thwart such attacks in such a way that only minimal manual intervention is required for the same.

PROBLEM IDENTIFICATION

The ARP protocol maps the MAC address of the host with the IP address and is positioned below the Internet layer. The ARP packet structure constitutes the Sender Hardware Address (SHA), the Target Hardware Address (THA), the Sender Protocol Address (SPA) and the Target Protocol Address (TPA). The IP address values of the sender and receiver are stored within SPA & TPA respectively. MAC addresses are available in the SHA and THA.

Within a network, the ARP request packets are broadcast by a source and the corresponding MAC is obtained from the intended host via an ARP reply packet. The host devices within the network maintain an ARP cache (dynamic or static) consisting of these address mappings and this is updated periodically based on the reply packets received without any authentication.

An ARP cache entry is usually created when an ARP request or reply message arrives at a host and if the cache does not contain any entry for the source IP in the received ARP packet. The entry is updated, and timeout time is renewed when an ARP request or reply message arrives at the host and the entry for the source IP in ARP header is already present. The major problem here is that the cache cannot verify the sender of ARP request/ reply packet [4].

One of the strategies used by hostile attackers towards the ARP attack is to corrupt the ARP cache of the targeted device to change the IP-MAC mapping stored in these devices. This will cause the traffic to be redirected to the wrong destination. A Man-In-The-Middle (MITM) attack occurs when an attacker poisons the ARP cache of two devices with the (48-bit) MAC address of their Ethernet NIC (Network Interface Card). Once the ARP cache has been successfully poisoned, the victim devices send all their packets to the attacker when communicating to the other device. With just the act of corrupting the ARP cache, an attacker can easily monitor or intercept all communication between target devices.

In traditional networks, several countermeasures like cache table static management, S-ARP [5], T-ARP [6], and ARP Table server synchronization method have been proposed. But practical application of these schemes poses difficulties in architecture like SDN. Many schemes involve the redirection of ARP traffic into an intermediate component, which would

then perform an analysis of the packet headers to identify the tampering [7]. While this idea may be practical for the traditional network architecture, it will turn out to be a choking hazard for the SDN controller, since the software-defined network functions with the Controller as the central component. In [8] the above said packet by packet inspection for SDN based ARP security has been presented.

As a first step towards ensuring ARP security using SDN, but without constantly monitoring the packets, in [9] the researchers presented a technique which analysed the flow statistics in the switches. This technique was able to identify unnatural ARP packet flooding. The spoofed packet was subsequently identified, and the source was blocked through a new flow entry in the switches. The drawback of this method was that the ARP cache was getting corrupted for a brief period during the attack discovery and hence required a network re-initialisation for recovery. A new scheme for detecting and recovering from ARP cache corruption had to be devised which could protect the internal network from ARP spoofing.

ARP CACHE CORRUPTION RECOVERY: AN OVERVIEW

The collection of statistics from the network and controlling of forwarding devices connected to the network are facilitated in a convenient manner through the software- defined concept. The methodology suggested in this paper is based on periodic collection of flow statistics from the connected switches, which are subsequently analysed for possible poisoning. This is an improvement of the method presented in [9]. The application has been written using Python over POX controller [10].

Threat model

It is assumed that the Controller of the network is a trusted system, but the hosts can turn malicious. Therefore, the communication from Controller to switches is reliable but not the other way around. In this paper, the researchers are considering only the internal attacks and hence the focus is only on the Open Flow messages inside the network.

Algorithm functionality

First and foremost, the application is required to intermittently analyse the ARP traffic through the network. If an unnaturally high amount of ARP packets is seen, the traffic is diverted to the Controller. The Controller attempts to detect spoofing within the packets. If spoofing is identified, an entry is inserted in the switches to drop the ARP and IP packets from the malicious source for a pre-defined period. The spoofing identification involves the capturing the IP and MAC of the attacker, the victim and the host that has been spoofed.

Fig. 1 represents the algorithm's workflow which consists of two stages. First, it initiates the process of flow statistics collection and instructs the connected switches to transmit the ARP packets to the Controller in order to obtain the mapping between IP addresses and MAC addresses of the hosts. The IP-MAC database stores the mapping information collected during this stage. Simultaneous entries are inserted into the switch flow tables so as to allow the already mapped host devices to communicate among themselves.

Secondly, having collected the network device information, the application no longer processes the ARP packets at the Controller level and focuses on the flow statistics. From this stage the Controller is involved only when a fake mapping is detected in the packet headers or when a new flow is identified. This prevents the traffic bottleneck at the Controller level.

To send packet to controller instruction is re-issued only if an unusual amount of ARP packet movement is observed. In this scenario, the Assessor module collects the ARP Packet data for analysis and suspends the traffic temporarily between the devices. It uses the information collected in the first stage to identify a mismatch and issues an alert.

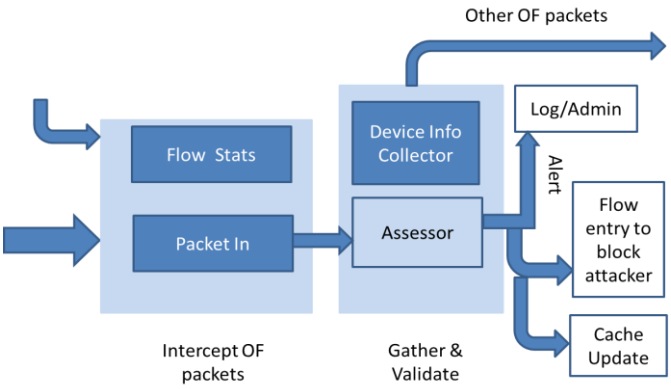

Figure 1. ARP Cache Recovery

Along with the alert, a flow entry is inserted in the switch flow tables to drop all the packets from the attacker. The algorithm then crafts ARP request probe packet on behalf of the victim and floods the rest of the network. The network is not waiting for the automatic ARP cache update to take place and instead a proactive ARP cache update on behalf of the Controller is carried out. The network continues to function in the normal manner without any disruption.

IMPLEMENTATION

The ARP Cache recovery module has been implemented and integrated with the SDN Controller. The module is written in Python and is compatible with POX. This section describes the implementation details of our application.

Experimental setup

The setup consists of Ubuntu Virtual Machine with 2 cores and 2 GB of RAM. The host machine has Windows 8 as the Operating system and has Intel Core i5 processor with 4 GB of RAM.

A network emulator is used to create the virtual network with Linux OS and Open Virtual Switch software. The Mininet emulator is installed on Ubuntu 14.10 Linux virtual machine that runs on VirtualBox. The controller used is POX, and ARP Cache recovery is run as a module on the SDN controller. Each host is connected by a 100 Mbps link to the network and a 1 Gbps link exists between switches. Tests have been performed to verify the capability of the application to detect and thwart the poisoning attack.

Using a socket program, the ARP Cache is corrupted by flooding the victim with fake ARP request/reply packets. For example, host 1 (the attacker) corrupts the ARP cache of host 4 by replacing the host 3's MAC address with its own MAC address. Now, host 1 is pretending to be host 3. This affects the communication between hosts 3 and 4.

Corrupted ARP cache recovery

In this experiment which is similar to the one executed in [9], host 1 is the attacker and poisons host 4 with an invalid entry. It floods the victim with hundreds of spoofed ARP request/reply packets. The *flowstats handler* tracks the ARP packets across the network. Any unusually high amount of ARP traffic is identified, and a new rule is installed in the switch flow table to redirect the ARP packets to the Controller. The *Packet-In* event of the Controller compares the incoming packet IP & MAC parameters with its own IP-MAC database. If the packet has been spoofed, it identifies the attacker and the victim. The procedure ARP cache recovery has been described in detail below. The input is only the IP and MAC of the ARP packet which has been identified as poisoned.

Once the attack is detected, the module adds a new flow table entry to drop all packets from the attacker for a fixed duration and issues an alert. Subsequently, the ARP Cache recovery module running in the Controller constructs an ARP Request probe packet from the spoofed host to the victim. The probe packets are then flooded. This corrects the victim device's ARP Cache without having to wait for the Cache timeout and thus keeps the device connected to the network. Fig.2 describes the algorithm used for cache recovery.

Algorithm:

Input: (IP, MAC) of ARP packet

Output: ARP packets on behalf of the victim from the ARP cache recover algorithm

1: If (IP, MAC) mismatch with database or packet header

2: { Identify the victim IP_v and attacker IP_a and the host that has been spoofed IP_h;

3: FlowModCommand(drop all packets from IP_a);

4: Craft ARP request Probe packet on behalf of IP_h asking for IP_v;

Broadcast the packets; }

End Procedure

Figure 2. The ARP Cache Recovery algorithm.

PERFORMANCE ASSESSMENT

The simulation results recorded the performance of ARP Cache recovery module during three stages - before, during and after attack detection and mitigation. Experiments were conducted using tree topology as well as linear topology. A custom Mininet Data Centre topology with four racks was also used for testing. Each rack had four hosts and a single top-of-rack switch. These switches were connected to a central root switch. In all the cases, the to and fro latency and TCP bandwidth between the devices which were affected before, during and after the attack were measured.

The graph in Fig.3 shows that there is no significant variation in latency between host devices in the pre and post attack stages. Of course, if the latency or throughput between the identified attacker and other devices were to be measured, no concrete results would be available for the post attack stage.

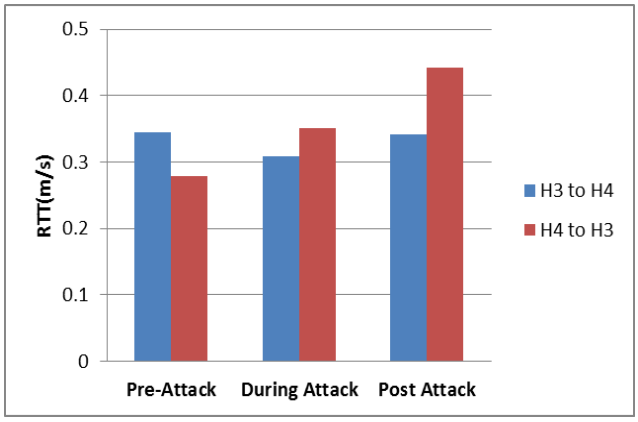

Figure 3. Comparison of latency experienced in different scenarios

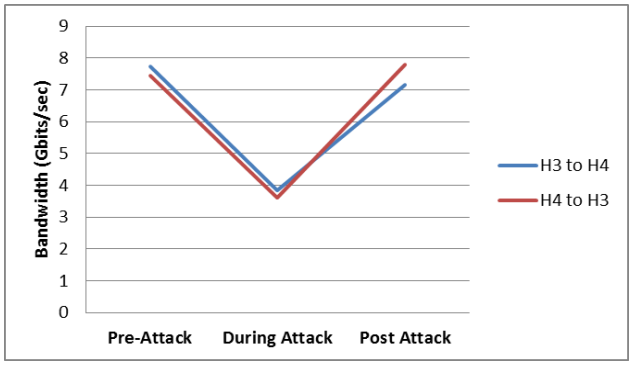

Figure 4. Comparison of bandwidth between devices in different scenarios.

As far as bandwidth (Fig. 4) was concerned there were no significant variations between pre-attack and post attack values. Unlike in [9] there was no down-time of the network at the moment when the ARP attack was discovered. Understandably, the throughput does reduce at this point since the Controller is involved in analysing the suspicious packets and inserting the drop rules in place for the attacker. The network continues its working state after the mitigation.

CONCLUSION

In this paper, an improved solution for detecting and mitigating ARP poison routing for SDN networks has been introduced. This feature can identify ARP poison routing and overcome it without degrading the network performance. The module not only successfully blocks the malicious host from further attack but also recovers the ARP Cache of the victim so that it can continue functioning within the network.

REFERENCES

[1] Cabaj, K., Wytrebowicz, J., Kuklinski, S., Radziszewski, P. and Dinh, K.T., "SDN Architecture Impact on Network Security," in FedCSIS Position Papers, pp. 143-148, Sep. 2014.

[2] Dhawan, M., Poddar, R., Mahajan, K., Mann, V.: SPHINX: detecting security attacks in Software-Defined Networks. In: Proc. of NDSS (2015)

[3] Hong, S., Xu, L., Wang, H., Gu, G.: Poisoning network visibility in software-defined networks: New attacks and countermeasures. In: Proc. of NDSS. pp. 8–11 (2015)

[4] S.Whalen. (2001)"An introduction to ARP spoofing." Node99 [Online].

[5] D. Bruschi, A. Ornaghi, and E. Rosti, "S-arp: a secure address resolution protocol," in Computer Security Applications Conference, 2003.Proceedings. 19th Annual. IEEE, 2003, pp. 66–74.

[6] W. Lootah, W. Enck, and P. McDaniel, "Tarp: Ticket-based address resolution protocol," Computer Networks, vol. 51, no. 15, pp. 4322– 4337, 2007.

[7] C.L.Abad and R.I. Bonilla, "An analysis on the schemes for detecting and preventing ARP cache poisoning attacks," in 2007 International Conference on Distributed Computing Systems Workshops (ICDCSW'07), pp. 60-60. IEEE 2007.

[8] Masoud, Mohammad Z., Yousf Jaradat, and Ismael Jannoud. "On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm." Applied Electrical Engineering and Computing Technologies (AEECT), 2015 IEEE Jordan Conference on. IEEE, 2015

[9] D. Balagopal and X.A.K Rani. "Empowering SDN Firewall against ARP Poison Routing." International Journal of Applied Engineering Research 12.18 (2017): 7466-7469

[10] A. Al-shabibi and M. McCauley, 'Pox Wiki', Openflow at Stanford, 2014.[Online].Available: https://OpenFlow.stanford.edu/display/ONL/POX+.