Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

AXARPSC: Scalable ARP Snooping Using Policy-based Mirroring of Core Switches with ARP Log Contraction

Motoyuki Ohmori[1,a)] Naoki Miyata[1,b)] Koji Okamura[2,c)]

Received: June 22, 2020, Accepted: December 1, 2020

Abstract: In order to handle a computer security incident or network failure, it is important to grasp a list of pairs of IP and MAC addresses of the hosts. A traditional method based upon ARP table polling, however, has two major drawbacks that 1) some pairs of IP and MAC addresses may not be obtained and 2) it incurs a heavy load on a core switch. In order to overcome these drawbacks, this paper proposes *AXARPSC* that is the novel scalable ARP snooping to build a list of pairs of IP and MAC addresses. AXARPSC can avoid missing pairs of IP and MAC addresses by monitoring all ARP traffic. AXARPSC also can reduce a CPU load on a recent high-end core switch by approximately 20%. AXARPSC is scalable because AXARPSC incurs no additional CPU load even though the number of hosts increases. AXARPSC employs a *policy-based mirroring* of a switch that mirrors traffic that matches a specified filter. The policy-based mirroring can mirror ARP traffic only, and reduce the load on an ARP parsing server. AXARPSC can also *contract* multiple contiguous ARP messages that have the same pair of an IP address and MAC address, as if one ARP message is observed.

Keywords: ARP snooping, scalability, computer security incident

1. Introduction

In order to handle a security incident or network failure, it is important to grasp a list of pairs of IP addresses and MAC addresses of hosts [1], [2]. For example, only an IP address of a suspicious host is alerted by an outer organization, Security Operations Center (SOC) or security equipment such as a next-generation firewall when a security incident occurs. If the suspicious host uses an IP address without any prior authorization or if an IP address assigned list is not properly maintained, the suspicious host cannot be identified. Even under such a situation, the suspicious host must be able to be identified and isolated from a network as soon as possible. In order to identify the suspicious host, a MAC address of the suspicious host must be obtained. It is, therefore, important to grasp a list of pairs of IP addresses and MAC addresses of hosts. To this end, ones may poll an Address Resolution Protocol (ARP) [3] table of a core switch at an interval using SNMP or other methods. We have, however, experienced that a MAC address of an IP address was not correctly obtained. The cause was that the IP address was duplicately assigned to different hosts. One of them was authorized but another was not. The unauthorized host then induced a security incident, and it was difficult to identify the host. It can be said that polling an ARP table is not accurate.

This paper proposes *AXARPSC* that is the novel scalable ARP snooping to build a list of pairs of IP addresses and MAC addresses. The challenge of this paper is to build the list without any extra load on a core switch. AXARPSC employs a *policy-based mirroring* [4] of a core switch that mirrors traffic that matches a specified filter. AXARPSC is, therefore, passive snooping, and incurs almost no load on a core switch while an active snooping or active MAC address table polling incurs many loads on a core switch. In addition, AXARPSC can reduce a load on an ARP parsing server because the policy-based mirroring can mirror ARP traffic only. AXARPSC is scalable because AXARPSC incurs no additional CPU load even though the number of hosts increases. AXARPSC can also regard many contiguous ARP messages that have the same pair of an IP address and MAC address as one ARP message in order to reduce the storage size.

Note that this paper is an extended version of our preliminary paper [5]. This paper is newly proposing and implementing contractions of logging messages that were not considered in the preliminary paper. These contractions are one of the main extended points from the preliminary paper, and are effective because they dramatically reduce the storage size of ARP log messages.

The rest of this paper is organized as follows. Section 2 presents existing ARP table polling and ARP table snooping and their problems. Section 3 proposes AXARPSC. Section 4 explains a prototype implementation of AXARPSC. Section 5 evaluates AXARPSC using the prototype implementation in an actual campus network. Section 6 discusses the validity of preconditions in this paper and further applied usage of AXARPSC. Section 7 refers to related work. Section 8 finally concludes this paper.

[1] Tottori University, Tottori 680–8550, Japan
[2] Kyushu University, Fukuoka 819–0395, Japan
[a)] ohmori@tottori-u.ac.jp
[b)] miyata@tottori-u.ac.jp
[c)] oka@ec.kyushu-u.ac.jp

2. Problems of Existing ARP Table Polling and Watching

2.1 Problems of ARP Table Polling

A traditional way to obtain a list of pairs of IP addresses and MAC addresses in a network is to poll an ARP table of a core switch. A polling server remotely polls an ARP table of a core switch using SNMP or Command Line Interface (CLI). The problems of this method are:

(1) an ARP table polling cannot track a quick IP address change less than an polling interval, and
(2) imposes a heavy load on a core switch.

Regarding the former, a polling interval is traditionally 5 minutes or longer. 5 minutes would be the minimal interval, and even the 5-minute interval cannot track a quick IP address change. We have actually experienced that an unauthorized host was assigned an IP address to its NIC for a very short duration within 5 minutes while an authorized host was always assigned the IP address. A pair of a MAC address and an IP address of the unauthorized host could not be listed in our ARP table tracking. The unauthorized host, unfortunately, caused a security incident, and it took a few days to identify the unauthorized host in 2018. Before identifying the unauthorized host, we wrongly regarded the authorized host as the cause of the incident, and wrongly isolated the authorized host from a network. This wrong operation decreased the availability of the authorized host. On the other hand, the unauthorized host might have compromised confidential information because of this long delay, and the incident might have been critical.

Regarding the latter, we have experienced more than 20% CPU load increase of a core switch when we polled an ARP table from the core switch that was AX8616 which is made by AlaxalA Networks Corporation. **Figure 1** shows a CPU load of a core switch without ARP table polling a day where about unique 6,000 IP addresses were assigned to hosts. In Fig. 1, CPU load 1/1 and 1/2 represent main and backup CPUs, respectively. The backup CPU load is always about 1% because backup CPU does not process any traffic in the authors' configuration. We, hereafter, focus on the main CPU load only. As shown in Fig. 1, the CPU load was basically 1% or less. On the other hand, **Fig. 2** shows a CPU load of the same core switch with ARP table polling using SNMP a day. A polling interval was 5 minutes. As shown in Fig. 2, the CPU load dramatically increased, and the maximum CPU load of that day was 31%. There were about unique 6,001 hosts, and the load might then double, i.e., 61%, if there were 12,000 hosts. In addition, this CPU load value was 5-minute average, and the peak maximum CPU load might have been 100%. When ARP table polling is enabled, other processes such as BGP routing with full routes may fail. This tendency could also be seen everyday when ARP table polling was enabled. It can, therefore, be said that ARP table polling imposes an unacceptably heavy load on a core switch.

We have also experienced that another core switch, AX8608, dropped necessary SNMP monitoring traffic by AlaxalA Networks Corporation AX-Security-Controller (AX-SC) [6] that employs ARP table polling by SNMP during IPSJ SIG Internet

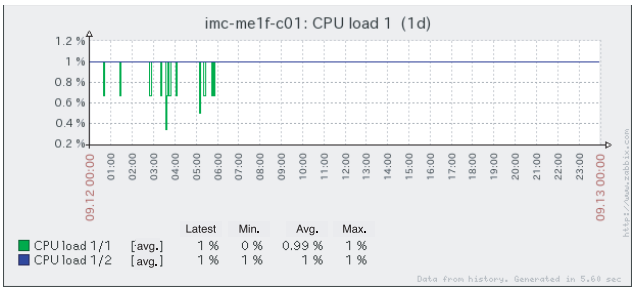

Fig. 1 CPU load of a core switch without ARP table polling.

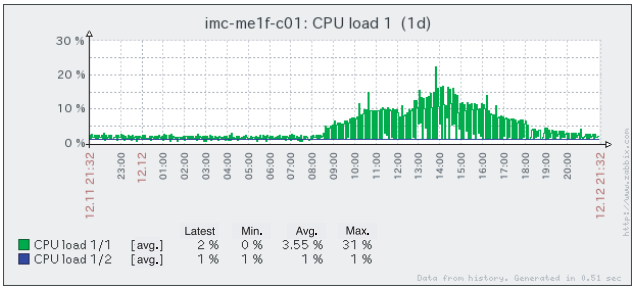

Fig. 2 CPU load of a core switch with ARP table polling using SNMP.

and Operation Technology Symposium 2018 (IOTS2018). In IOTS2018, there were fewer than 200 hosts observed on a network. In order to monitor network usage, we retrieved network traffic information, i.e., sent and received bits per second, of the core switch by SNMP, and it had worked at first. On even such a small network, the network traffic information could not be retrieved by SNMP right after enabling AX-SC. When AX-SC was disabled, the network traffic information could be retrieved again. ARP table polling of AX-SC seems to have incurred a heavy load on the CPU of the core switch. Again, it can be said that ARP table polling imposes an unacceptably heavy load on a core switch.

As described above, an ARP table polling table has critical problems, and these problems should be solved or mitigated.

2.2 Problems of ARP Watching

Another way to obtain a list of pairs of IP addresses and MAC addresses in a network is to watch an ARP. The most famous implementation of this method is *arpwatch* [7]. Employing *arpwatch*, we can collect pairs of IP addresses and MAC addresses of hosts that broadcasts ARP requests into a network. An ARP request is usually broadcasted and it includes a MAC address and an IP address of a host which is sending an ARP request. This way, however, cannot avoid a malicious host escaping from having its MAC address collected. This is because that a malicious host can statically configure an ARP entry for a default gateway in order to avoid emitting an ARP request.

In order to avoid this kind of malicious host escape, we may be able to capture all traffic and snoop all ARP requests and replies. Since a default gateway must send an ARP request to the malicious host and receive an ARP reply from the malicious host, a MAC address of the malicious host can be obtained. It is, however, difficult to implement because all traffic in a network should be snooped and many snooping servers are necessary.

As described above, ARP watching has flaws of security and scalability.

3. AXARPSC

This section presents AXARPSC that is the novel scalable ARP snooping and solves all issues described in Section 2. AXARPSC is novel because AXARPSC is the first proposal to snoop ARP traffic only, and can reduce a storage size by *contracting* ARP log messages. The basic idea of AXARPSC is very simple:

(1) mirroring only ARP traffic by employing *policy-based mirroring* [4], and
(2) parsing ARP requests and replies, and
(3) storing a pair of an IP address and MAC address, and
(4) *contracting* multiple ARP messages that have the same pair of an IP address and MAC address to one pair of an IP address and MAC address.

Policy-based mirroring is a traffic mirroring function which improves a traditional mirroring function of a core switch. A traditional mirroring can mirror all traffics on a specific VLAN or specific physical port. A traditional mirroring, however, cannot mirror only a specific protocol traffic, i.e., cannot restrict mirrored traffic based upon a protocol. On the other hand, policy-based mirroring can selectively mirror traffic by protocol. Policy-based mirroring can support mirroring only an ARP request and reply. This feature enables to reduce unnecessary traffic when only an ARP request and reply are necessary.

AXARPSC parses not only an ARP reply but also an ARP request in order to accurately obtain the most recent pair of an IP address and MAC address. An ARP request includes an IP address and MAC address of a sender, and AXARPSC snoops an ARP request. If the pair of an IP address and MAC address is firstly observed or different from the previous pair, AXARPSC stores the new pair as a new entry. AXARPSC also stores a new pair of an IP address and MAC address of a target host, i.e., a host asked to respond to an ARP request, seen in an ARP reply. When AXARPSC stores a new pair of an IP address and MAC address, the time when the new pair was observed is also stored. If a different MAC address for the same IP address is previously stored in a different old entry, the time is also stored in the old entry as the last seen time, and the new entry is created.

In addition, AXARPSC *contracts* ARP syslog messages. For example, regarding an incident, the most important is which IP address is assigned to which host at the specified time. It is not necessary to store all logs of all ARP requests and replies for this purpose. Before an ARP entry expires, the ARP entry continues to exist on an ARP table unless the ARP entry is overridden by another MAC address. Only the latest ARP request or reply is, therefore, necessary before an ARP entry expires. In order to reduce a storage size, AXARPSC regards the same contiguous pair of an IP address and MAC address as one entry even if many ARP messages that have the same pair are observed. AXARPSC just updates the last seen time of the entry, and does not create a new entry.

Figure 3 depicts an overview of the AXARPSC system. In Fig. 3, a core switch has multiple VLANs configured, and those VLANs are necessary to be snooped. ARP requests and replies on those VLANs are mirrored to an ARP snooping server. An ARP snooping server parses ARP packets, generates syslog messages, and sends these syslog messages to a logging server. A logging server then stores syslog messages in order to provide a user with a pair of an IP address and a MAC address when an incident or network failure occurs.

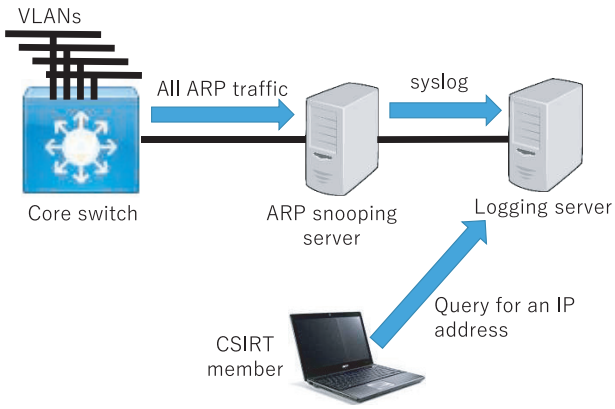

Fig. 3 An overview of AXARPSC system.

```
interface tengigabitethernet 1/1
  advance access-group ARPfilter in-mirror
  advance access-group ARPfilter out-mirror
!
destination-interface-list ARPmirror mode mirror
  destination interface gigabitethernet 7/1
!
advance access-list ARPfilter
  1000 permit mac any any arp tag-vlan 1000
       action policy-mirror-list ARPmirror
  1001 permit mac any any arp tag-vlan 1001
       action policy-mirror-list ARPmirror
```

Fig. 4 An example of policy-based mirroring configuration.

Note that AXARPSC assumes that:

(1) a core switch supports policy-based mirroring or similar mirroring as described in Section 6.2,
(2) there are few L3 core switches in a site or campus, and
(3) many VLANs are routed at L3 by a core switch.

4. Implementation

This section presents our first prototype implementation of AXARPSC.

We have configured policy-based mirroring on AX8616 which is made by AlaxalA Networks Corporation. We have configured 220 VLANs on which all ARP traffic were to be mirrored to an ARP snooping server via a physical GbE link using *advanced access-list* as shown in **Fig. 4**.

We have prepared a physical server, not a virtual machine, as an ARP snooping server. We have then implemented an ARP snooping server using python3 [8]. In order to parse ARP packets, we have utilized Scapy [9] that provides simple but strong APIs to parse ARP packets. An ARP snooping server then extracts a source MAC address and IP address included in an ARP packet. Note that not only an ARP reply but also an ARP request is parsed in order to obtain the most recent pair of an IP address and MAC address. An ARP snooping server generate a syslog message as shown in **Fig. 5**. An ARP snooping server then sends a generated syslog message to a logging server.

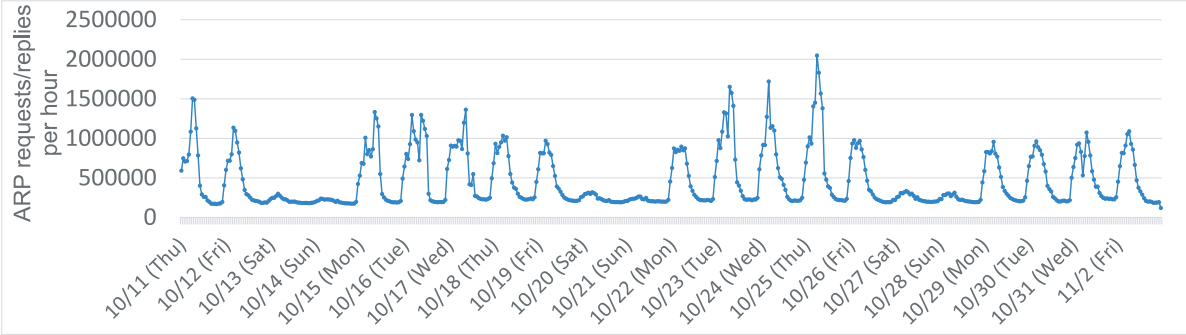

Fig. 6 The number of ARP requests and replies per hour (from 2018/10/11 9:00 to 2018/11/3 4:00).

```
Nov  3 07:36:00 hostname pname: de:ad:be:ef:de:ad 10.0.0.1
```

Fig. 5 An example of syslog output of AXARPSC.

Table 1 ARP snooping server specification.

CPU	Intel(R) Xeon(R) CPU E5506 2.13 GHz (8 core)
Memory	8 GB
OS	CentOS Linux release 7.5.1804 (64 bit)

We have implemented a logging server using fluentd [10] and mongoDB [11] in order to shorten the search time on an incident or network failure.

5. Evaluations

We here evaluate AXARPSC using our prototype implementation. This section firstly introduces the evaluation environment. The number of ARP requests and replies in an actual environment is shown in order to make sure that our prototype implementation works properly. It is then proved that core switch load increase of AXARPSC is smaller. This section then looks into a load on an ARP snooping server. Finally, we present how big storage is necessary to hold all pairs of IP addresses and MAC addresses of all hosts.

5.1 Evaluation Environment

We have implemented the scalable ARP snooping at a core switch, AX8616, in our Koyama campus network in Tottori city where about 6,000 unique hosts can be seen at maximum. In this campus, we have only one core switches, and almost all VLANs are accomodated and routed at L3 by the core switch. We have configured policy-based mirroring for 220 VLANs on the core switch. The core switch accommodates all wireless traffic using wireless LAN controller and all (i.e., wired and wireless) traffic in this campus goes through this core switch to the Internet. The external link to the Internet is 1 Gbps and almost all internal links are 10 Gbps in the core switch. An ARP expiry timer on the core switch is configured to be 5 minutes. The reason of this configuration is not clear, and the previous network vendor might configure in order to minimize an ARP table size. We have installed one physical ARP snooping server, and its specification is shown in **Table 1**.

5.2 ARP Requests and Replies

In order to make sure that our prototype implementation works properly, we here dig into the number of ARP requests and replies

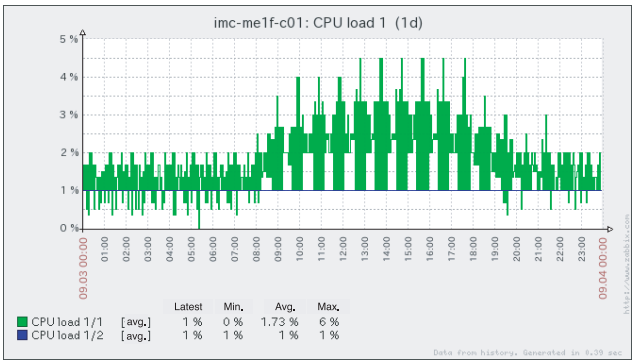

Fig. 7 CPU load of a core switch with scalable ARP snooping.

that are snooped and stored. **Figure 6** shows the number of ARP requests and replies per hour from 2018/10/11 9:00 to 2018/11/3 4:00. The maximum number is 2,047,676 on 2018/10/25 15:00. As shown in Fig. 6, there are more ARP requests and replies on week days than there are on holidays. We have made sure that all MAC addresses were properly stored in comparison with a traditional ARP table polling, and there was no dropped MAC address except for a mis-operation.

On the other hand, we have made sure that there is an effectiveness of policy-based mirroring. When policy-based mirroring was not employed, i.e., all traffic was mirrored, many packets were dropped and many ARP packets were then dropped. It can be said, therefore, that AXARPSC works better than a traditional ARP watching method that needs all traffic to be mirrored.

5.3 Increase of Core Switch Load

As described in Section 2.1, a traditional ARP table polling imposes a heavy load on a core switch. In order to make sure that AXARPSC incurs no heavy load on a core switch, we here check to see if a CPU load is increased or not when AXARPSC is enabled. **Figure 7** shows a CPU load of a core switch on a day, 2018/9/3, when AXARPSC was enabled. AXARPSC was enabled at approximately 13:00. In Fig. 7, we cannot see any spikes around 13:00. It can, therefore, be said that AXARPSC incurs no heavy load on a core switch while a traditional ARP table polling increases a CPU load by more than 30% under some circumstances.

5.4 ARP Snooping Server Load

As described in Section 5.3, AXARPSC incurs almost no additional load. AXARPSC, however, introduces an ARP snooping

Table 2 Data size of all ARP messages on mongoDB.

the number of logs	256,942,822
data size	24,167,277,036 B (approx. 24 GB)
index size	3,713,732,608 B (approx. 3.7 GB)

Table 3 Data size of ARP messages on mongoDB.

logs	duration	size
AXARPSC with contraction	1 year	321 MB
ARP without contraction	1 month	19 GB
SNMP ARP polling (no contraction)	1 month	406 MB

server, and its load might be concerned. The load average of the ARP snooping server was about 10% at maximum in our environment where there were about 6,000 unique hosts and 2,000,000 ARP requests and replies. Our ARP snooping server has 8 cores, and may be able to handle almost 80 times the number of hosts, i.e., 480,000, which is almost the same as the maximum limit, 512,000, of our core switch AX8616.

5.5 Data Size of All ARP Messages

We here examine how big the storage must be to hold all pairs of IP addresses and MAC addresses. Note that no ARP *contraction* was employed here, and all ARP messages were stored in this section. Our logging server that employs fluentd and mongoDB held data from 2018/10/11 9:00 to 2018/11/3 4:00. **Table 2** presents data and index sizes of mongoDB. Note that we here have not configured mongoDB to have indices such as an IP address and MAC address. In order to reduce the searching time on an incident and network failure, it would have been better if we had these indices but that is future work. As shown in Table 2, we need approximately 30GB of storage in order to hold about one month of data.

On the other hand, we also stored the same data in normal syslog text file without compression. The syslog text file size is about 19 GB that stores all ARP requests and replies from 2018/9/3 13:11 to 2018/11/7 22:00. The size of the syslog text file is less than one of MongoDB even though the syslog text file stores longer data. The physical size of the syslog text file is smaller because an underlying file system is an object strage, and an object is automatically redundant and compressed. The physical size is, however, unknown and its computation is for future work.

5.6 Data Size of Contracted ARP Messages

We here examine how a data size can be contracted. **Table 3** shows data size of entries of pairs of IP address and MAC addresses on mongoDB. Note that the sizes in Table 3 are reported by mongoDB statistics, and not actual file sizes. As shown in Table 3, AXARPSC contracting ARP messages requires only 321 MB for a year (from 2019/6/27 to 2020/6/22), and can apparently reduce the data size. Since the data size of AXARPSC is relatively small, we here hold entries about a year, which is longer than other methods. AXARPSC also can reduce more data than SNMP ARP polling. It can be said that AXARPSC contract is very effective from the point of view of the storage size.

6. Discussions

6.1 Scalability of AXARPSC

As described in Section 5.3, AXARPSC incurs no heavy load on the core switch, and there is almost no additional load. As shown in Fig. 7, the load on the core switch is correlated only with the number of hosts, and this load tendency may result just from ARP processes. It can, therefore, be said that AXARPSC is not correlated with the number of hosts regarding a core switch.

On the other hand, a traditional ARP table polling may incur almost 100% CPU load on a core switch when the number of hosts increases more than three times.

As described in Section 5.4, it can be said that AXARPSC can handle the sufficient number of hosts because our ARP snooping server seems to handle the same number of hosts as our core switch. Even if the number of hosts increases, we may be able to employ a faster server as an ARP snooping server.

It can, therefore, be said that AXARPSC is more scalable than a traditional ARP table polling in terms of the number of hosts.

6.2 Generality of AXARPSC

This section discusses the generality of AXARPSC, i.e., whether AXARPSC can be generally implemented in other network environments. As described in Section 5, we have evaluated AXARPSC with the core switch made by AlaxalA Networks Corporation. It may be conceived that AXARPSC can be implemented only with switches made by AlaxalA Networks Corporation that support policy-based mirroring. Policy-based mirroing is actually a terminology of AlaxalA Networks Corporation, and is implemented only on their switches. Other switch vendors, however, do not have exactly the same although they have similar mirroring functions, and AXARPSC can be imeplemented with such switches. For example, Cisco Catalyst 6500, which is widely used as a core switch in a university, has VLAN Access Control List (VACL) capture [12]. Regarding smaller switches, Cisco Catalyst 3750, 3560 and 2960 do not have VACL capture, but have flow-based SPAN (FSPAN) [13]. Cisco ASR 9000 has flow or ACL-based traffic mirroring [14]. Cisco Nexus 7000 has ACL capture or rule-based Switched Port Analyzer (SPAN) [15]. In the case of AlaxalA, not only AX8600 that we have evaluated in this paper but also smaller, so called *box-type*, switches also support policy based mirroring [16], [17], [18]. In addition, switches of other vendors also have similar mirroring functions. Generally speaking, recent Application Specific Integrated Circuits (ASICs) on switches have similar mirroring functions, and these ASICs are commonly used even on different vendors. Policy-based mirroring or similar mirroring may be implemented on switches which are different from the ones we have evaluated in this paper. It can be said that AXARPSC can be implemented on the switches of several vendors.

Box-type switches that have policy-based mirroring or similar mirroring, however, may have restrictions. For example, policy-based mirroring cannot be enabled when an IP ACL is already configured. Many *box-type* switches in an acutual network may not be able to enable policy-based mirroring. Policy-based mirroring, therefore, may be enabled at a core switch in many cases.

6.3 Operation Cost

As shown in Fig. 4, we currently have to specify each VLAN ID for each VLAN. There is no way to specify all VLAN to be

mirrored, and this configuration might be forgotten by mistake when a new VLAN is created. These kinds of mistakes might cause the case where a MAC address cannot be resolved from a given IP address on an incident. In order to avoid these kinds of mistakes, we may need to automate an operation to create new VLAN. Or a configuration of an access list for policy-based mirroring should be improved so that a VLAN ID can be omitted when we configure policy-based mirroring on a physical port.

6.4 Data Size and Contraction

AXARPSC contracts ARP logs that have the same pair of an IP address and MAC address instead of storing all ARP requests and replies logs. This feature reduces the data storage. People may be, however, able to detect an anomaly when all ARP requests and replies are stored. It is still unclear if the contraction must be implemented in all cases. We will dig into the ARP requests and replies as future work in order to make sure that all ARP requests and replies are worthy of being held.

6.5 Network Access Control (NAC) Tools

We may consider NAC tools [19], [20] to obtain a list of pairs of IP addresses and MAC addresses. NAC tools utilize DHCP or RADIUS accounting in order to obtain pairs of IP addresses and MAC addresses. In the case of wireless LAN, many NAC tools may utilize RADIUS accounting to obtain a pair of IP address and MAC address. If IEEE802.1x is employed, many NAC tools can properly obtain a correct pair of an IP address and MAC address. IEEE802.1x is very common in wireless LAN nowadays, and NAC tools work fine. On the other hand, in the case of wired LAN, there are a few difficulties to obtain a correct pair of an IP address and MAC address. For example, some switches such as AlaxalA do not report an IP address of a host in RADIUS accounting while switches made by Cisco report an IP address. In order to obtain an IP address of a host, a DHCP server can report a correct IP address of a host if DHCP is configured on a network. If DHCP is not configured, there is no way of obtaining an IP address of a host. As described above, an NAC tool may work fine, but may not in some cases. AXARPSC can properly operate such cases.

7. Related Work

PFU limited has released iNetSecSF [21] that snoops ARP packets for security on 2017. Their sensor that snoops ARP packets can accommodate only 32 VLANs at the same time. If people would like to snoop ARP packets on more VLANs, they need to install more their sensors. AXARPSC is, therefore, more scalable than iNetSecSF.

AlaxalA Networks Corporation has released AX-Security-Controller (AX-SC) [6] on 2017 that can store a list of an IP address and a MAC address. AX-SC, however, employs a traditional method using SNMP to periodically poll an ARP table from a core switch. AX-SC cannot then track a quick IP address change as described in Section 2.1.

Tokyo University of Agriculture and Technology has also implemented automated suspicious host isolation [22], [23]. They, however, adopted, AX-SC, and have similar problems to those described above.

8. Concluding Remarks

This paper has proposed AXARPSC that is the novel scalable ARP snooping using policy-based mirroring. AXARPSC solves critical issues that traditional ARP table polling and watching methods have. Employing policy-based mirroring, only ARP traffic can be mirrored, and then it is getting easier to parse all ARP packets without any packet loss. In addition, AXARPSC does not store all ARP requests/replies but merges multiple ARP messages for the same pair of an IP address and MAC address to one entry. AXARPSC has appeared to reduce a storage size by this contraction to 321 MB for a year in the authors' environment while 19 GB is required for a month without the contraction.

References

[1] Ohmori, M., Higashino, M., Kawato, T., Fujio, S. and Nakashima, K.: On-demand Suspicious Host Isolation Adopting Software Defined Network Approach on a Computer Security Incident Response, *Journal of Information Processing*, Vol.27, pp.234–243 (online), DOI: 10.2197/ipsjjip.27.234 (2019).

[2] Ohmori, M.: On Automation and Orchestration of an Initial Computer Security Incident Response by Introducing Centralized Incident Tracking System, *Journal of Information Processing*, Vol.27, pp.564–573 (online), DOI: 10.2197/ipsjjip.27.564 (2019).

[3] Plummer, D.: Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware, RFC 826 (Standard) (1982).

[4] AlaxalA Networks Corporation: Policy Based Mirroring (in Japanese) (2016), available from ⟨http://www.alaxala.com/jp/solution/network/pbm/pbm/⟩ (accessed 2018-11-07).

[5] Ohmori, M., Miyatal, N. and Suzuta, I.: AXARPS: Scalable ARP Snooping Using Policy-Based Mirroring of Core Switches, *Proc. 33rd International Conference on Advanced Information Networking and Applications* (*AINA-2019*), pp.667–676 (2019).

[6] AlaxalA Networks Corporation: AX-Security-Controller (2017), available from ⟨http://www.alaxala.com/jp/news/press/2017/20170601.html⟩ (accessed 2017-06-03).

[7] Network Research Group, Lawrence Berkeley National Laboratory: arpwatch (2009), available from ⟨https://ee.lbl.gov/⟩ (accessed 2018-11-07).

[8] Python Software Foundation: python (2001), available from ⟨https://www.python.org/⟩ (accessed 2018-11-07).

[9] Philippe Biondi and the Scapy community: Scapy: Packet crafting for Python2 and Python3 (2018), available from ⟨https://scapy.net/⟩ (accessed 2018-11-07).

[10] Fluentd Project: fluentd (2010), available from ⟨https://www.fluentd.org/⟩ (accessed 2018-10-07).

[11] MongoDB, Inc.: mongoDB (2018), available from ⟨https://www.mongodb.com/⟩ (accessed 2018-10-07).

[12] Cisco Systems: VACL Capture for Granular Traffic Analysis with Cisco Catalyst 6000/6500 Running Cisco IOS Software (2007), available from ⟨https://www.cisco.com/c/en/us/support/docs/lan-switching/vlan-access-lists-vacls/89962-vacl-capture.html⟩ (accessed 2020-09-30).

[13] Cisco Systems: Flow-Based SPAN Alternative to VACL Capture (2013), available from ⟨https://www.cisco.com/c/en/us/support/docs/lan-switching/switched-port-analyzer-span/116133-maintain-flow-span-00.html⟩ (accessed 2020-09-30).

[14] Cisco Systems: Configuring Traffic Mirroring on the Cisco ASR 9000 Series Router (2014), available from ⟨https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r4-1/interfaces/configuration/guide/hc41asr9kbook/hc41span.html⟩ (accessed 2020-09-30).

[15] Cisco Systems: Cisco Nexus 7000 Series NX-OS System Management Configuration Guide (2020), available from ⟨https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus7000/sw/system-management/guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide/b_Cisco_Nexus_7000_Series_NX-OS_System_Management_Configuration_Guide_chapter_010100.html⟩ (accessed 2020-09-30).

[16] AlaxalA Networks Corporation: AX2500S Software Manual Configuration Guide Vol.2 (in Japanese) (2019), available from ⟨https://www.alaxala.com/jp/techinfo/archive/manual/AX2500S/PDF/4_19/CFGUIDE2/CFGUIDE2.pdf#page779⟩ (accessed 2020-09-30).

[17] AlaxalA Networks Corporation: AX3660S Software Manual Configuration Guide Vol.2: Policy-Based Mirroring (in Japanese) (2020), available from ⟨https://www.alaxala.com/jp/techinfo/archive/manual/AX3660S/HTML/12_1_L/CFGUIDE2/0376.HTM⟩ (accessed 2020-09-30).
[18] AlaxalA Networks Corporation: AX4630S Software Manual Configuration Guide Vol.2: Policy-Based Mirroring (in Japanese) (2019), available from ⟨https://www.alaxala.com/jp/techinfo/archive/manual/AX4600S/HTML/11_15_G/CFGUIDE2/0381.HTM⟩ (accessed 2020-09-30).
[19] HC Networks, Ltd.: Account@Adapter+ (in Japanese) (2020), available from ⟨https://www.hcnet.co.jp/products/adapter/accountadapter.html⟩ (accessed 2020-09-30).
[20] Infoblox: Infoblox (2020), available from ⟨https://www.infoblox.com/⟩ (accessed 2020-09-30).
[21] iNetSecSF: SiNetSec SF: Detecting security risk and blocking (in Japanese) (2017), available from ⟨https://www.pfu.fujitsu.com/inetsec/products/sf/⟩ (accessed 2018-11-07).
[22] Tsujisawa, T., Sakurada, T., Segawa, H., Kawamura, Y., Mishima, K. and Hagiwara, Y.: A challenge for full deployment both 802.1x authentication and an automatically isolation function in a campus network. Task and correspondence in the operation, *Journal for Academic Computing and Networking*, Vol.22, No.1, pp.36–43 (2018).
[23] AlaxalA Networks Corporation: Case Study: Tokyo University of Agriculture and Technology (2017), available from ⟨https://www.alaxala.com/jp/introduce/case36/⟩ (accessed 2018-10-01).

Motoyuki Ohmori was born in 1976. He received his B.S. and M.S. degrees in Computer Science and Communication Engineering from Kyushu University in 1999 and 2001, respectively. He joined the Information Processing Society of Japan in 2001. He was a lecturer at Chikushi Jogakuen University since 2004. He was an associate professor at Tottori University since 2013. He has been a professor at Tottori University since 2020. His research interests include network architecture, multicasting, routing, mobile networking and energy efficient network operation. He is a member of the IPSJ, IEICE, JSSST, IEEE CS/ComSoc and ACM.

Naoki Miyata was born in 1982. He received his B.S. degree in Hiroshima Institute of Technology in 2001. He has been a technical staff member at Tottori University since 2006. He has been working as a technical staff member of Center for the Information Infrastructure & Multimedia, Tottori University since 2006. His research interests include document-oriented database, database algorithms, routing, logging.

Koji Okamura was born in 1965. He received his B.S., M.S. and Ph.D. degrees from Kyushu University in 1988, 1990, and 1998. He worked as an Associate Professor of Computer Center and Graduate School of Information Science and Electrical Engineering, Kyushu University. Since 2011, he has been a Professor of Kyushu University. He is the Director of the Cybersecurity Center and the Vice Director of the Research Institute for Information Technology, Kyushu University. He is also Vice CISO of Kyushu University. His current research interests are Cybersecurity for information network and social infrastructure and advanced operation technologies for Internet and Future Internet such as Openflow and Virtual Network. He is also researching power-aware and security-aware network operation and developing green power and secure network equipment system.