Source Address Validation in Software Defined Networks

Bingyang Liu †‡§ Jun Bi †‡ Yu Zhou †‡
† Institute for Network Sciences and Cyberspace (INSC), Tsinghua University
‡ Tsinghua National Laboratory for Information Science and Technology (TNList)
§ Huawei Technologies Co. Ltd.
bingyangliu.bl@gmail.com, junbi@tsinghua.edu.cn, yuz.thu@gmail.com

1. INTRODUCTION

IETF source address validation improvement (SAVI) work group develops methods to prevent nodes attached to the same IP link from spoofing each other's IP addresses. RFCs are developed for various circumstances with different address assignment mechanisms (AAM), or the mix of them.

To support SAVI, network device vendors need to implement the set of RFCs for various network environments in their switches. However, the new network paradigm, software defined networks (SDN), provides the opportunity to deploy SAVI without implementing extra features in SDN switches. In this paper, we present the preliminary design and implementation of SDN-SAVI, a module/ application on SDN controllers to enable SAVI functionalities in SDN networks. In this proposal, all the functionalities are implemented on the controller without modifying SDN switches. To enforce SAVI on packets in the data plane, the controller installs binding tables in switches using existing SDN techniques, such as OpenFlow. With SDN-SAVI, a network administrator can now enforce SAVI in her network by merely integrating a module on the controller, rather than purchasing SAVI-capable switches and replacing legacy ones.

Our contribution is the design and implementation of an SDN-based SAVI solution. We have overcome three challenges in the design. First, to avoid redundant AAM snooping and address binding on all switches, we classify switch ports into three categories and define different behaviors on them, so as to make snooping and binding effective and efficient (Section 2.1). Second, to prevent the controller and switches from resource exhaustion attacks, we rate-limit AAM packets in the data plane and offload the controller's verification job to switches (Section 2.3). Third, to solve the flow table explosion problem caused by the coexistence of the rules generated by multiple applications, we utilize the "multiple tables" feature of OpenFlow and make the flow table compact and concise (Section 2.4).

This work was supported by the National Science Foundation of China (No.61502267 and No.61472213).
Jun Bi is the corresponding author.

SIGCOMM '16, August 22-26, 2016, Florianopolis , Brazil

DOI: http://dx.doi.org/10.1145/2934872.2960425

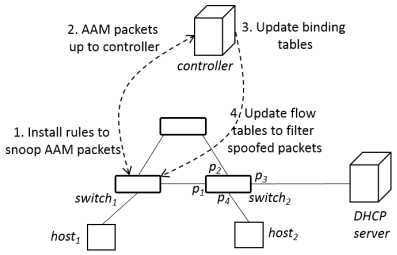

Figure 1: Overview of SDN SAVI

2. DESIGN

We present the design of SAVI-DHCP and SAVI-FCFS for IPv6 (IPv4 is similar). We use OpenFlow as the SDN technique. The overview is illustrated in Figure 1. Initially, the controller installs several rules in an OpenFlow switch to snoop AAM packets. Then the switch will send each AAM packet to the controller as an OpenFlow packet-in. Next, the controller updates the binding tables according to the address assignment state indicated by the snooped AAM packets. Finally, if the binding table is updated, the controller modifies the flow table of the switch to install or remove a filtering entry with respect to the IP address.

2.1 AAM Snooping

To snoop AAM packets, the controller installs several rules into switches in advance. For SAVI-FCFS, the controller snoops duplicated address detection (DAD) packets, so the rules match neighbor solicit (NS) and neighbor advertise (NA) fields. For DHCPv6, the controller snoops DHCPv6 client (DC) messages and DHCPv6 server (DS) messages, so the rules match UDP port 546 and 547. The actions associated with the rules are "output packets to the controller".

We classify switch ports into three categories, i.e., *Switch Ports*, *Host Ports*, and *Trust Ports*, which connect with switch-

es, ordinary hosts, and trust servers (e.g., DHCP servers), respectively. For example, in Figure 1, p_1 and p_2 are Switch Ports, p_3 is a Trust Port, and p_4 is a Host Port. Snooping is performed on Host Ports and Trust Ports, and verification is only performed on Host Ports.

The controller maintains a state machine for each AAM (e.g., RFC 7513 for DHCP [1]). Each IP address has a state, whose transition is triggered by AAM packets. When the state changes to BIND, a binding entry is set for this IP address in the binding table; when the state changes to NO_BIND, the IP address is removed from the table.

2.2 Binding Table

A binding entry binds an IP source address to a switch port, in the format of $< Address, Switch, Port >$, i.e., the entry point of the packet with IP source $Address$ should be $Port$ of $Switch$; otherwise it is a spoofed packet.

2.3 Security Consideration

The SDN-SAVI module snoops AAM packets and maintains state machines and timers on the controller. This may become a new vulnerability for DoS attack, where attackers can send AAM packets to overwhelm the controller. To solve this, we use the *meter* element of OpenFlow switches to rate-limit AAM packets. This is feasible since AAM traffic is low-rate under normal circumstances.

The simplest way to enforce validation is to do it on the controller. The controller verifies the first packet of each new flow. There are two ways to deal with a spoofed packet. First, ignore it. The main drawback is that an attacker can repeatedly send the same packet to overwhelm the controller's computation resources. Second, passively install in the switch a drop rule matching the new flow. However, both the computation resources on the controller and the flow table resources on the switch can be exhausted when an attacker floods packets with random source addresses.

To protect these resources, SDN-SAVI explicitly installs rules in switches to block unbound IP source addresses. Thus, spoofed packets are verified and dropped in the data plane without consuming flow table resources or computation resources on the controller.

2.4 Enforcement

SDN-SAVI binding entries are enforced with flow rules in switches. The challenge is how these rules coexist with the flow rules generated by other applications such as forwarding. One way is to block unbound IP source addresses, leaving the unmatched (legitimate) packets to be processed by the rules of other applications. However, in practice, the number of unbound addresses may exceed the flow table size. Another way is to combine the rules of SDN-SAVI with other applications, e.g., for each bound IP source address, generate N "two-dimension" rules, where N is the number of forwarding rules which matches IP destination addresses. However, this method causes flow table explosion.

We solve the flow table explosion problem using the "multiple tables" feature of OpenFlow. SDN-SAVI only uses the first flow table. Packets matching bound rules are directed to latter tables for further processing by other applications, while spoofed ones are dropped by the first table. This method does not only eliminate flow table explosion, but also makes SDN-SAVI rules separate and independent.

Name	Match	Instruction1	Inst2
Switch Ports	(SP) in-port=tp_1,ipv6 (SP) ... (SP) in-port=tp_m,ipv6	goto Table 2 ... goto Table 2	
DAD	(NS) ipv6,nw-s=::,nw-d=ff02::/16,icmpv6,icmpv6-type=135 (NA) ipv6,nw-d=ff02::1,icmpv6,icmpv6-type=136	toController toController	meter1 meter2
DHCP	(DC) ipv6,nw-s=fe80::/16,nw-d=ff02::1:2,udp,udp-s=546,udp-d=547 (DS) ipv6,nw-d=fe80::/16,udp,udp-s=547,udp-d=546	toController toController	meter3 meter4
Trust Ports	(TP) in-port=tp_1,ipv6 (TP) ... (TP) in-port=tp_n,ipv6	goto Table 2 ... goto Table 2	
Host Ports	(HP) in-port=hp_i1,ipv6,nw-s=s_1 (HP) ... (HP) in-port=hp_ik,ipv6,nw-s=s_k	goto Table 2 ... goto Table 2	
Spoofed Packets	(SF) ipv6	drop	
Non-IPv6	... ...	... ...	

Figure 2: The first flow table of a switch

3. IMPLEMENTATION & EVALUATION

We implement SDN-SAVI on Floodlight controller. Source code is at https://github.com/netarchlab-savi/savi-floodlight.

Figure 2 show the 6 rule blocks of the SDN-SAVI flow table. 1st block lets pass all the IPv6 packets arriving on the m Switch Ports. Then for Host Ports and Trust Ports, received AAM packets are sent to the controller, including DAD (2nd block) and DHCP packets (3rd block). These two blocks are rate-limited with meters. For non-AAM packets, we let pass all the packets on the n Trust Ports (4th block), and verify all the packets on Host Ports (5th block). Finally, 6th block drops all the spoofed IPv6 packets received on Host Ports. Non-IPv6 blocks are placed thereafter.

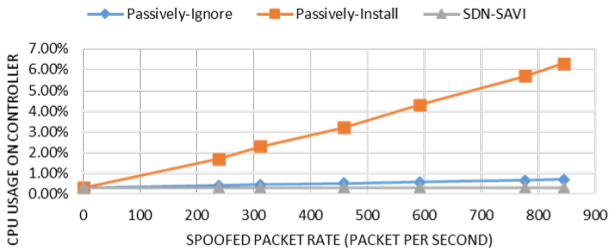

Figure 3: CPU Usage Comparison under Spoofing Attack

We also implemented the strawman controllers described in Section 2.3, namely Passively-Ignore and Passively-Install. We performed spoofing attacks with random source addresses to measure the CPU usage of these controllers running on Intel E5-2637. Figure 3 demonstrates the results under different attacking rates. The CPU usage of Passively-Install increases fast with attacking rate because the controller needs to deal with packet-ins, packet-outs and flow-mods. The slop of Passively-Ignore's curve is much smaller because it only needs to deal with packet-ins. The CPU usage of SDN-SAVI is low and constant since all spoofed packets are dropped in the data plane.

4. REFERENCES

[1] J. Bi, J. Wu, G. Yao, and F. Baker. Source address validation improvement (savi) solution for dhcp. RFC 7513, 2015.