Volume: 6 Issue: 3 Pages: 2347-2351 (2014) ISSN : 0975-0290

OpenFlow Security Threat Detection and Defense Services

Wanqing You
Department of Computer Science, Southern Polytechnic State University, Georgia
Email: wyou@spsu.edu
Kai Qian
Department of Computer Science, Southern Polytechnic State University, Georgia
Email: kqian@spsu.edu
Xi He
Department of Computer Science, Georgia State University, Georgia
Email: xhe8@student.gsu.edu
Ying Qian
Department of Computer Science, East China Normal University, China
Email: yqian@cs.ecnu.edu.cn

ABSTRACT

The emergence of OpenFlow-capable switches de- couples control plane from the data flow plane so that they support programmable network and allow network administrators to have programmable central control of network traffic via a controller. The controller and its communication with switches and users become a malicious attack target. This paper explores major possible security threats and attacks on the controller of SDN and proposes a new approach to automatically and dynamically detect and monitor malicious behaviors on flow message passing and defend such attacks to ensure the security of SDN. We have built a FlowEye prototype at service level on Mininet API, and simulation tests are done on two feasible attacks on OpenFlow Beacon platform. The paper provides the feasibility study of such attacks and defense protection strategies in SDN security research.

Keywords - **OpenFlow; Software Defined Network; security**

Date of Submission: September 09, 2014 Date of Acceptance: November 13, 2014

I. INTRODUCTION

The advent of SDN (Software Defined Network) brings a set of concepts of network organizing techniques. It breaks up the limitations of traditional network framework and pulls out the controlled units to a logically centralized controller plane, which aims to separate control plane from data plane. While a centralized controller provides great flexibility to manage the entire network, there are also many concerns in SDN security. OpenFlow is the communication protocol between network devices and controllers. It defines message format and the associated action upon receiving messages, and is considered an important implementation of SDN.

The position paper [1] classifies potential SDN threats into seven categories. Shown in Fig. 1, these SDN threats can happen on the data plane, on the control plane or the communication between the data plane and the control plane. Some of the threats only exist in SDN, while the impact of the other traditional threats is potentially augmented in SDN. The dynamic flow tunneling attack we are discussing in this paper is the attack introduced in SDN, which attacks on vulnerabilities in controllers.

The advent of SDN has brought the network security community both opportunities and challenges. On one hand, the centralized network management offers a platform for addressing the traditional network concerns. New approaches to defending network attacks are enabled with global data collected from the whole network. On the other hand, the controller components and its connection with network devices have become the main targets of new network attacks. Potential software bugs or backdoor in the controller can make the entire network vulnerable to network attacks.

In this paper, we discuss two kinds of network attacks, ARP spoofing and dynamic flow tunneling Attack. We propose the corresponding detection and defense strategies for them. ARP spoofing attack is a typical network attack that also exists in the traditional network. Dynamic flow tunneling Attack is caused by the malicious software and is specific to SDN. We design new approaches to detecting two attacks in the SDN, and implement defense strategies. The rest of the paper is organized as follows: Section II briefly reviews the recent work on SDN attack and defense. Then two kinds of network attacks are discussed and analyzed in Section III and IV. Detailed implementation is presented in V. We offer some conclusions and future roadmap in Section VI.

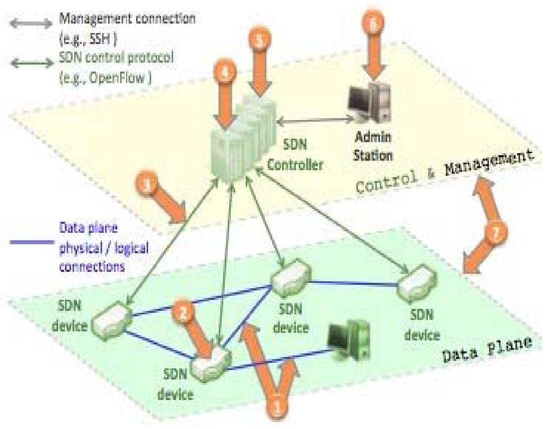

Management connection (e.g., SSH)
SDN control protocol (e.g., OpenFlow)
SDN Controller
Admin Station
Control & Management
Data plane physical / logical connections
SDN device
SDN device
SDN device
SDN device
Data Plane

Fig. 1. SDN Main Threat Map [1]. Seven types of SDN threats are identified.

II. RELATED WORK

While the emergence of SDN brings network operators more flexibility to program their networks, these capabilities actually introduce new fault and attack plane enabling threats that did not exist or were harder to exploit in the traditional networks. A few related works are targeting at the prevention and detection of SDN network attacks.

FortNOX [2] is a security enforcement kernel developed by extending the NOX [3] controller. It is capable of prioritizing the flow rule installed on switches according to different type of applications, and checking flow rule contradictions in real time. FRESCO [4] is an OpenFlow security application development framework which can facilitate the creation and deployment of security service in SDN. CloudWatcher [5] is a framework that provides security monitoring services for large and dynamic cloud networks and detour network packets to be inspected by pre-installed network security devices automatically. PermOF [6], a fine-grained permission system on OpenFlow network, minimizes the privileges applications to mitigate the privilege abuse problems. A NICE system [7] presented by Princeton University aims to test unmodified controller applications in an efficient and systematic way, which applies the model checking to explore the state space of the entire network system.

III. ARP SPOOFING ATTACK AND DETECTION APPLICATION

An ARP Spoofing or ARP Poisoning attack is the egression of unsolicited ARP messages. These ARP messages contain IP addresses of network resources, such as the default gateway, or a DNS server, and replace the MAC address of the corresponding network resource with its own MAC address.

Fig. 2 shows what is ARP and when an ARP spoofing can happen. In the Fig. 2, Host 1 wants to send packets to Host 2, but without any knowledge of the MAC address of Host 2. It would send out an ARP request in order to get the MAC address of Host 2. At this time, the attacker in the network pretends to be Host 2 and sends back its MAC. Thus the packets from Host 1 would be sent to a wrong destination. This is ARP spoofing.

We propose a new approach to detect the ARP spoofing, and a simple defense strategy which prompts an alert when we can detect the ARP spoofing. More systematical defense strategies should be designed in the future. A possible solution (illustrated in Fig. 3) is shown as follows:

1) ARP Spoofing defense application manages a buffer which saves the IP addresses and MAC addresses of all network resources. Let the ARP Spoofing defense application collect every ARP reply message.

2) ARP Spoofing defense application extracts (IP, MAC) pair from the ARP reply message. Compare this pair with buffered (IP, MAC) pairs to check if any two pairs have same IP address but different MAC addresses. If yes, prompt an alert. Otherwise, add the new (IP, MAC) pair into the buffer.

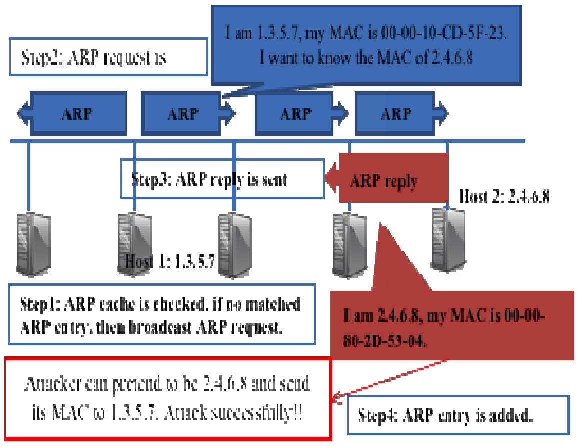

Step2: ARP request is
I am 1.3.5.7, my MAC is 00-00-10-CD-5F-23. I want to know the MAC of 2.4.6.8
ARP
ARP
ARP
ARP
Step3: ARP reply is sent
ARP reply
Host 2: 2.4.6.8
Host 1: 1.3.5.7
Step1: ARP cache is checked. if no matched ARP entry, then broadcast ARP request.
I am 2.4.6.8, my MAC is 00-00-80-2D-53-04.
Attacker can pretend to be 2.4.6.8 and send its MAC to 1.3.5.7. Attack success fully!!
Step4: ARP entry is added.

Fig. 2. ARP Spoofing.

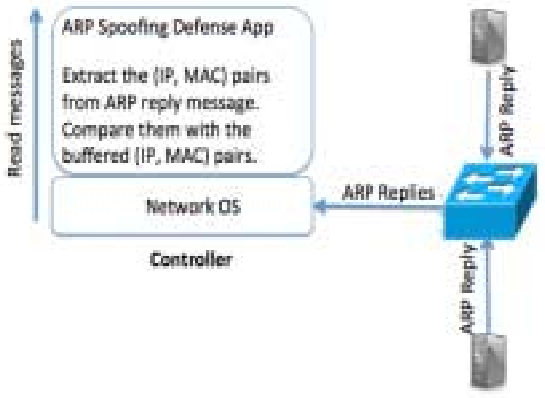

ARP Spoofing Defense App
Extract the (IP, MAC) pairs from ARP reply message. Compare them with the buffered (IP, MAC) pairs.
Read messages
Network OS
Controller
ARP Replies
ARP Reply
ARP Reply

Fig. 3. Simple simulation of ARP Spoofing detection.

IV. Dynamic Flow Tunneling Attack and Detection

A new evasion scenario in SDN is described by Porras et al. in [2]. By injecting fake flow rules into switches via malicious applications on controllers, hackers can bypass the firewall, and invade the network system. In an imaginary network shown in Fig. 4, for example, hacker computer *A* is trying to illegally access computer *C* (10.0.0.2 → 10.0.0.4). Because the firewall prohibits any packets originally from computer *A* arriving at computer *C*, direct communication between these two computers will be denied by the firewall. One possible strategy for hacker computer *A* to bypass the firewall is, on one hand, to send packets to computer *B* (10.0.0.2 → 10.0.0.3), and these packets can go through the firewall and reach the switch. On the other hand, the malicious application on the controllers has forged rules on the switch's flow table. One set of such rules changes the destination IP/MAC address of the packets that are from computer *A* to computer *B*, so that these packets are redirected to computer *C*. Once computer *C* receives packets from computer *A*, it will send reply packets to computer *A*. The other set of fake rules are then applied to these reply packets, and replace the source IP/MAC addresses with computer *B*'s. As a result, the reply packets are camouflaged to be the ones from computer *B*, and pass through the firewall.

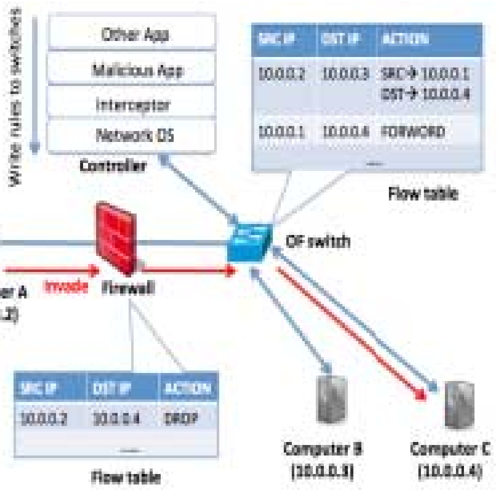


Fig. 4. An Evasion Scenario in SDN.

Our basic idea for detecting such network attack is to maintain an adjacent matrix that keeps track of the connectivity between any pair of computers in the network system. If any new rules are about to write to switches, these rules are intercepted and used to update the adjacent matrix. By comparing the connectivity information in the adjacent matrix with the firewall policy, this network evasion can be detected. In the next paragraph, detailed algorithm for maintaining the adjacent matrix will be explained by a concrete example.

In a small network G with computer nodes *A*, *B*, *C* and *D* (left-hand side of Fig. 5), two computer nodes are connected with a dotted line if they are physically connected but not allowed to communicate, while nodes are connected with a solid line if they can communicate with each other. (We skip switches that connect computer nodes) The right-hand side of Fig. 5 is the adjacent-list representation of G, where one computer node is the neighbor of another computer node if there is a physical connection between these two computer nodes. The left-hand side adjacent matrix in Fig. 6 is the initial adjacent matrix. If at some point the communication path between B and C is enabled, then the communication paths between B and C's neighbors, and between C and B's neighbors are also enabled. In this example, communication paths of (A, C), (B, D), and (A, D) are connected. For those computer nodes (such as computer node A and D) involved in the previous update, the updates also involve their neighbors. The process continues recursively until all related nodes are visited. Finally, the changes on node connectivity are updated to the adjacent matrix. The right-hand side of Fig. 6 is the updated adjacent matrix after enabling communication path (B, C).

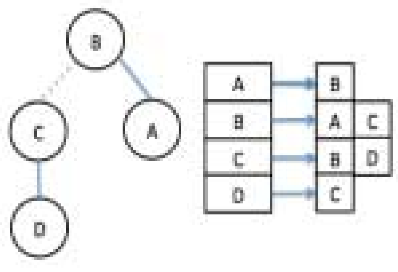
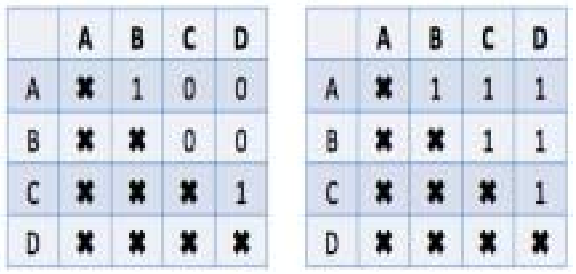


Fig. 5. The topology of a sample network and it adjacent-list.



	A	B	C	D
A	✖	1	0	0
B	✖	✖	0	0
C	✖	✖	✖	1
D	✖	✖	✖	✖

	A	B	C	D
A	✖	1	1	1
B	✖	✖	1	1
C	✖	✖	✖	1
D	✖	✖	✖	✖

Adjacent matrix before and after enabling communication path (B, C). 1/0 denotes two computer nodes are connecting/disconnecting.

V. Implementation

A. Environment Setup

We use Mininet as the network simulation tool in our experiments to build a virtual OpenFlow network, and interact with it via command line interface or APIs. With Python scripts specifying the network architecture, a customized OpenFlow network such as the one shown in Fig. 2 or Fig. 4 can be constructed in Mininet. Each computer node in the virtual network can be manipulated in the same way as in the real network. Moreover, Mininet not only provides default OpenFlow controller to the virtual network, but also allows other controller (POX, Beacon, Floodlight, Trema) connecting to the virtual network. In addition, the firewall in Fig. 4 can be also configured inside a switch in Mininet virtual OpenFlow network.

We choose Beacon [8], a java-based open source project, as our OpenFlow controller. Considered as a network operating system, Beacon not only has implemented efficient I/O operation, communicating with network devices, but also integrates seamlessly with Equinox and Spring framework, providing easy-to-use platform for developers to develop and deploy applications and enabling dynamic addition or removal of Beacon applications. To develop a new Beacon application, developer usually only need to implement Interface *IOFMessageListener*, and handle the incoming messages from OpenFlow switches in accordance to application-specific business logic.

B. ARP Spoofing Detection Application

Due to the limitation of Mininet virtual OpenFlow network, traditional ARP attack tools are not properly working in the network setting of our experiments. For the purpose of the demonstration, we simulate ARP attacks by deploying a special Beacon application, which produces two ARP messages with same host IP addresses yet different MAC addresses.

We implement another Beacon application: the ARP detection program. It overrides the *receive()* function declared in the Interface *IOFMessageListener*, scans all the ARP-specific OpenFlow messages, and adds the (IP, MAC) pairs to the global (IP, MAC) table. Once two pairs in the (IP, MAC) table with same IP address but different MAC addresses are identified, it will alert the Beacon system that an ARP spoofing attack is happening. At the same time, potential hacker hosts, which issue fake ARP reply messages, are recorded in the log file for further investigation.

C. Dynamic Flow Tunneling Detection and Defense Service

The dynamic flow tunneling detection and defense service is developed inside the Beacon system. Fig. 7 describes the basic workflow inside the Beacon system. The Controller class periodically queries if these are incoming messages. If yes, it will receive message and forward them to the application if the messages are *packet-in* messages. Otherwise, other handlers will process the messages. The applications process the *packet-in* messages in some pre-determined order, and pass the outgoing flow-mod messages to the *OFMessageAsyncStream* class, which will in turn write these messages to the switches. The dynamic flow tunneling detection and defense service lies inside the *OFMessageAsyncStream* class. The *write()* function inside the *OFMessageAsyncStream* is overridden in such a way that when a new flow-mod message is passed to it, it will execute the dynamic flow tunneling detection algorithm, as discussed in Section IV, before actually writing the message to the switches.

VI. Conclusion

In this paper, we examine two important network attacks in SDN and propose respective detection and defense strategies. More comprehensive experiments are being carried out to evaluate the proposed defense strategies. Other than these two attacks, much more work remains in building a rich suite of applications that cover a wide range of security issues. Although dynamic flow tunneling attack is specific to SDN while ARP Spoofing attack exists in both SDN and traditional network, we think both proposed defense approaches rely on SDN's centralized controller which can provide network-wise device control and data access.

We believe that a network security infrastructure built on the centralized controller would provide a good solution for defending network attacks in SDN. Therefore, constructing a network security infrastructure would become our next step.

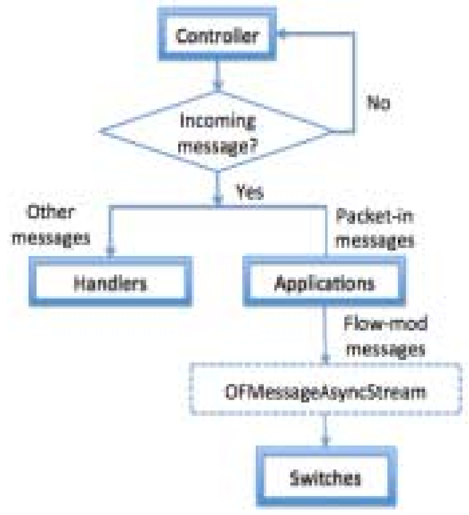


Fig.7. The data flow inside the Beacon. The rectangles represent Beacon class instances.

References

[1] D. Kreutz, F. Ramos, and P. Verissimo, Towards secure and dependable software-defined networks, Proc. the second ACM SIG- COMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 55–60.

[2] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, A security enforcement kernel for openflow networks, Proc. the first workshop on Hot topics in software defined networks, ACM, 2012, pp. 121–126.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, Nox: towards an operating system for networks, ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, pp. 105–110, 2008.

[4] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, Fresco: Modular composable security services for software-defined networks, Proc. Network and Distributed Security Symposium, 2013.

[5] S. Shin and G. Gu, Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?), Proc. 2012 20th IEEE International Conference on Network Protocols (ICNP), IEEE, 2012, pp. 1–6.

[6] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, Towards a secure controller platform for openflow applications, Proc. the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 171–172.

[7] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford, A nice way to test openflow applications, Proc. the 9th USENIX conference on Networked Systems Design and Implementation, Apr, 2012.

[8] D. Erickson, The Beacon OpenFlow Controller, Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 13-18

Biographies and Photographs

Wanqing You is a graduate student in the department of computer science & software engineering at Southern Polytechnic State University. She got her bachelor degree in software engineering at Xiamen University, China, 2013.

Dr. Kai Qian is a computer science professor in the department of computer science & software engineering at Southern Polytechnic State University. He got his Ph.D in computer science and engineering at University of Nebraska-Lincoln, 1990. His research areas include computer network and mobile security, big data analysis for security, machine learning, and pattern recognition. He has published about 100 research papers in these areas in many journals and conferences. He has received a number of research projects on the cybersecurity from NSF these years.

Xi He is a CS research assistant and instructor at Georgia State University. He is specialized in parallel and distributed computing, network architecture, and grid computing. He has many year industrial experiences as a software engineer and has published papers in his research areas.

Dr. Ying Qian is an Associate Professor in the Department of Computer Science and Technology, at East China Normal University, Shanghai, China. She received her Master and Ph.D. degree in Department of Electrical & Computer Engineering from Queen's University, Kingston, Ontario, Canada. Her research interests include Software Defined Network, high-performance scientific computation, and parallel programming. She has published about 20 research papers in these areas in many journals and conferences.