An Approach for Protecting the OpenFlow Switch from the Saturation Attack

Mingxin Wang[1,a], Huachun Zhou[2,b], Jia Chen[3,c], Bo Tong[4,d]

[1,2,3,4]National Laboratory of Next Generation Internet Interconnection Devices, School of Electronic and Information Engineering Beijing Jiaotong University, Beijing 100044, China

[a]email: 14111007@bjtu.edu.cn, [b]email: hchzhou@bjtu.edu.cn, [c]email: chenjia@bjtu.edu.cn, [d]email: 14120125@bjtu.edu.cn

Keywords: SDN, security, OpenFlow, cache, threshold

Abstract. Security is always a serious issue influencing the development of Software-Defined Network (SDN). The central control mechanism makes the SDN controller a bottleneck of the network which is vulnerable to network saturation attack. In this paper, we propose an approach to defense this kind attack. Firstly, we add a miss matched packet cache module in the OpenFlow switch which can temporarily cache the packets that don't match in the flow table. Besides, we apply the mechanism of separating the header and payload of packets in the cache queue once the switch detects the volume of cache queue exceeding the threshold of the cache size. In addition, the switch can classify the packets headers and send it in an alert message to the SDN controller for further processing. At last in the paper, we evaluate the effort of our proposed approach in Mininet. With our approach, the SDN network can effectively defend the network saturation attack.

I. Introduction

The Software Defined Networking (SDN)[1] is a disruptive innovation in the networking industry which decouples the control plane from the data plane. SDN supports fine-grained network management policies so that the network configuration and management can be handled by the centralized controller which facilitates the upgrade of functionality and debugging. OpenFlow[2] is a standardized protocol which defines the southband interface of SDN. OpenFlow promotes the programmability of the network making it more flexible and efficient.

However, security is a serious issue which influences the development of SDN. Current OpenFlow protocol adopts a reactive caching mechanism where whenever a switch does not find a matching rule for the flow of one of its incoming packets, the packets will be send to the controller asking for the flow rules and the other packets belongs to the same flow will be stored temporarily in the switch's buffer. This mechanism brings two main drawbacks. Firstly, in the SDN network, controller becomes the single point which is vulnerable to several of attacks. A large number of table miss messages from the attackers may exceed the processing capabilities of the controller which will cause a DoS attack so that the controller cannot process legal packets. Secondly, the reactive caching mechanism also makes switches vulnerable to a DoS attack where malicious users flood the switch with large payload packets that belong to different flows. These malicious packets are likely to be no match with the flow tables in the switch which require sending queries to the controller. The switch has to store these large payload packets in the buffer for forwarding that will easily fill up the buffer. As a result, some new coming legitimate packets will be dropped. In this paper, we mainly focus on solving the two issues above.

We add a cache module which stores the miss match packets in the OpenFlow switch. We implement a threshold based packet header and payload decoupled mechanism in this cache module. When the threshold has been reached, the switch will send all these miss match packets to the cache module and decouple all packets' headers and their payloads. Then the module will arrange the packets into five tuples flows and send them to the controller as an alert message for deep inspection. The application we develop in the controller will analyze the flows and install defense flows in the corresponding switches to restrict the attackers. In this way, the buffer queue can be reduced significantly and the switch will be more robust to the saturation attack. In addition, we verify the effort of our approach in the Mininet environment.

The paper is organized as follows: Section 2 mainly introduces the state of the art of SDN network security threat and the related works on relieving the influence of the attacks. Section 3 presents the design of our approach and gives the implementation method. In section 4, we analyze the performance of our approach and give the experiment result to prove. Section 5 gives the conclusion of our work.

II. Related Work

SDN security has always been a hot topic since SDN concept emerges because that SDN is a double-aged sword for network security. The new features of the SDN paradigm bring great benefits to the networking security. [3] lists three core characteristics that differentiate SDN networks from traditional networks from a security perspective, global network view, self-healing mechanism, increased control capabilities. [4] combines SDN and sFlow[5] monitor to defend the DRDoS attack. It decouples the controlling function from monitoring to minimize the cost of controller. [6] proposes an approach to quickly detect DDoS attack in the network. It takes advantage of the central control concept of SDN and uses entropy to identify the attack in the early time. [7] proposes a lightweight DDoS flooding attack detection approach using OpenFlow which uses neural network technique to classify the anomaly traffic. [8] proposes a SDN based malware detection system in mobile network which monitors the network traffic to detect mobile malwares and cooperates with SDN controller installing rules to implement access control.

However, the security of SDN or OpenFlow itself remains to be addressed. Some serious security issues are specific to SDN. [9] lists 7 threat vectors in SDN which has to be tackled. Among these, the attacks on control plane communications, the attacks on controllers and the lack of mechanisms to ensure trust between the controller and management applications are specific to SDN. These vulnerabilities may bring serious security disasters to the SDN network. Some work has been done to tackle the SDN security issues. AvantGuard [10] is proposed to solve the SDN network saturation attack. AvantGuard implements a TCP SYN proxy as a module in the SDN switch so that it can only serve the flows which finish the TCP handshake. AvantGuard can alleviate TCP saturation attack effectively which however, has some limitation for other protocols. FloodGuard [11] is another approach which is also proposed to defend saturation attack in both control plane and data plane. FloodGuard adopts proactive flow rules to preserve network policy enforcement and packet migration mechanism to protect the controller from being overloaded. However, it cannot detect who the attacker is and stop it from attacking.

III. Design and Implementation

Our approach is proposed to solve the security of SDN network itself. We mainly focus on SDN network saturation attack on both control plane and data plane. We will discuss the method in the following parts.

A. Overview of the System

Figure 1 shows the architecture of our proposed system. Our work mainly includes two parts. Firstly, we add a cache module to restore the miss match packets in the buffer queue. Then we propose a threshold based packet header and payload decoupled mechanism which is implemented in the proposed module in order to defend the buffer saturation DoS attack. The module will constantly monitor the buffer state. Once the threshold is exceeded, the decoupling process will be triggered and an alert will be send to the controller for further process. Secondly, we develop a security analysis application in the controller to analyze the alert message from the data plane. Once the attackers are confirmed, the corresponding flow rules will be installed to prevent the attack.

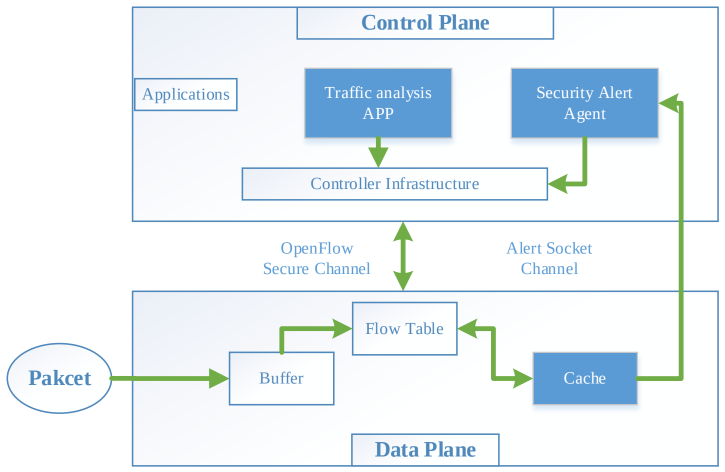


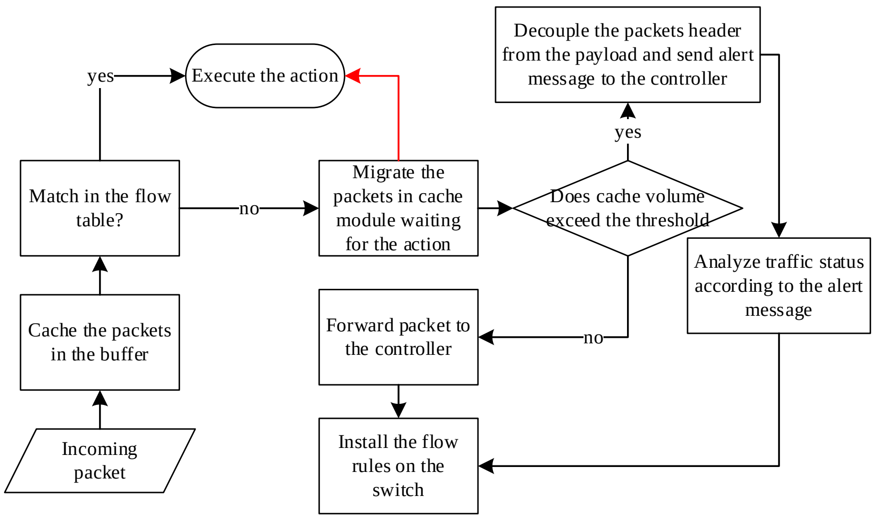
Fig. 1. Overview of the System Architecture

B. Packet's header and payload decoupled mechanism

In the common OpenFlow switch, there is a flow table which is used to store the forwarding rules indicating an action about how to handle the incoming flows. If there comes a new flow that is unmatched in flow table, the switch will forward the packet or packet header of the flow to the controller for inquiring. As OpenFlow protocol adopts the reactive caching mechanism, the switch is vulnerable to the saturation attack where the miss-matched flow's packets will be cached temporarily on the switch's buffer. The malicious users can flood the switch with large payloads packets which belong to different flows. The limited buffer of the switch can be filled up quickly.

To address this issue, we proposed a mechanism which decouples the packet payload from the header. This procedure is parallel with the process of common OpenFlow switch. Once the buffer queue size exceeds the threshold T, the decoupling modules start to run. At the same time, switch still forwards the miss match packet header to the controller querying for the flow rules. After the first step of decoupling, the switch will classify the packets header according to 5 tuples including source IP address, destination IP address, source port number, destination port number and protocol type. In addition, the packets' statistics number of each flow will be recorded. The information will be send to the controller as an alert message through an external channel defined by us. The security application in the controller will further process the alert message.

The existing technique such as entropy value[6] or neural network[7] could analyze the statistics characters of the uploaded alert messages. Once the attack behavior is detected, the controller can immediately install the rules to restrict the attacker. The whole procedure of our design is shown in Figure 2.


yes
Execute the action
Decouple the packets header from the payload and send alert message to the controller
yes
Match in the flow table?
no
Migrate the packets in cache module waiting for the action
Does cache volume exceed the threshold
Analyze traffic status according to the alert message
Cache the packets in the buffer
Forward packet to the controller
no
Incoming packet
Install the flow rules on the switch
Fig. 2. The flow diagram of the packet processing

C. The security analysis application in the controller

In Figure 1, security alert agent is used for listening the data plane alert continuously. Once the agent receives an alert message, it will pass it to the traffic analysis application for processing. We develop this APP in the controller which can calculate each tuple's entropy value of the flows. Entropy can reflect the traffic variation of the data plane, which further can identify the attack mode. Lots of researches have proven that some attributes such as IP address, port number and type of protocol, which have a strong character of autocorrelation and heavy-tailed property[12] that can be used to judge whether the traffic is abnormal. So we can utilize entropy to quantize each attribute of the traffic flow.

The traffic analysis application we develop is entropy based which can analyze the alert message. The APP is SVM algorithm based which can train the normal traffic and attack traffic model in advance. When the alert message arrives, the traffic analysis application will transfer the attributes in the message into entropy value to reflect the traffic condition of the switch.

IV. Performance Analysis and Experiment

A. Performance Analysis

In this section, we mainly analyze the performance of the proposed approach. We assume that the packets arrive at the switch according to a Poisson process and the arrival rate is λ under the normal situation while the service time of the switch is exponential with the parameter of μ. The mean service time for processing one miss matched packet is Q. We assume the mean value of the packet length is l. As the length of the buffer is L, we can get that the buffer can store N packets where $N = \frac{L}{l}$ under the normal traffic situation. Once the buffer is filled up, the switch will drop the new incoming packets. So the model of the switch buffer is M/M/1/N. When the attack starts, the arrival rate of the packet turns into λ' while the mean value of the packet length turns into l'. So the buffer can only store N' packets where $N' = \frac{L}{l'}$ when the saturation attack happens. We can get that not only the packet arrival rate increases but also the storing capability of the buffer decrease as the attack packet usually has large payload. We can calculate that the packet loss probability P_{loss} which means the buffer is full in the normal state.

$$P_{loss} = P(n = N) = \rho^N \times P_0 \tag{1}$$

$$P_0 = \begin{cases} \frac{1-\rho}{1-\rho^{N+1}} & \rho \neq 1 \\ \frac{1}{1+N} & \rho = 1 \end{cases} \tag{2}$$

When the attack happens, the buffer size will decrease from N to N' and arriving rate will increase from λ to λ'. The packet loss probability P'_{loss} will increase significantly which means a lot of miss-matched flows will be denied of service.

With our method, all the miss-matched packets will be send to the cache module waiting for process. Once the threshold is reached, the cache will decouple the payload from packet header. With the help of the decoupling, the cache can store more packets just with the header information that is enough for the controller to further analyze.

B. Experiment Result

We use Mininet as our evaluation environment. We apply POX[13] as the SDN controller platform and develop the security application on it. Then we use Scapy[14] to forge UDP packets and send attack packets to the OpenFlow switch as Figure 3 shows. At the same time, the client sends benign TCP request to the server through the switch. We measure the first miss matched packet delay from the client.

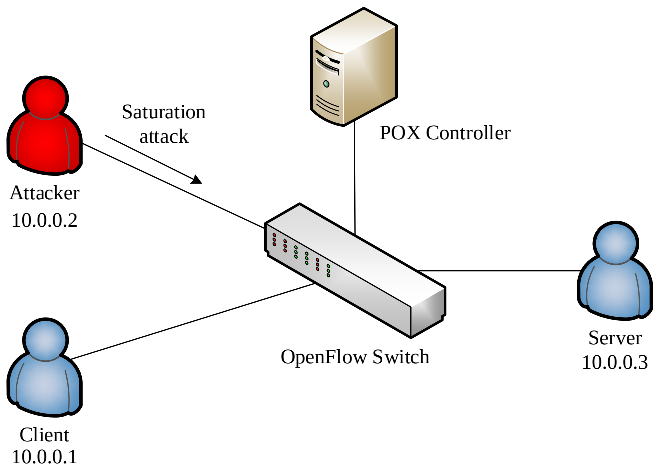


Fig. 3. Experimental Topology

From Table 1 we can see that without flooding traffic it takes an average time about 30ms to process and forward the first packet of a new flow in the original OpenFlow network. However, when the attacker continually sends new flows with large payload to the switch, the delay will be infinite as the buffer is full and all the new miss-matched packets are dropped. As we adapt additional cache module, miss matched packets will be directly sent to the cache module waiting for forwarding to the controller. At normal traffic situation, it also takes about 30ms to forward the first miss match packet and the extra time can be negligible. When the attack begins, it will take the controller about 310ms to handle the alert message and return the new rules. The main delay is on account of the security app's processing time in the controller.

Table 1. Comparison of OpenFlow and our approach about delay

Time delay	**OpenFlow**	**Our Approach**
Normal traffic	30ms	30ms
Under UDP attack	∞	310ms

V. Conclusion

In this paper, we propose to add a cache module in the OpenFlow switch which can decouple the payload from the packet header once the threshold is exceeded. With the help of this method, the cache size in the switch will increase which can accommodate more incoming miss matched packets. What's more, the cache module can organize these packet headers into different flow indexes and forward them to the controller encapsulating in an alert message. The traffic analysis application in the controller will analyze the message based on entropy and SVM algorithm. As a result, the application can install defense flows on the corresponding switches to prevent the attack. In the experiment, we mainly test the performance of our proposed approach in Mininet. The result shows that when the flooding attack happens, the benign user can still be served.

Acknowledgement

In this paper, the research was sponsored by China National Science and Technology Major Projects of the Ministry of Science and Technology of China (Project No. 2013ZX03006002), the National High Technology of China (863 program, Project No. 2015AA010301), Beijing Natural Science Foundation (Project No. 4132053), National Natural Science Foundation of China (Project No. 61471029) and Fundamental Research Funds for the Central Universities (Project No. 2015YJS028).

References

[1] ONF, “Software-Defined Networking: The New Norm for Networks,” 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” SIGCOMM CCR, vol. 38, no. 2, pp. 69–74, 2008

[3] Dabbagh, Mehiar, et al. "Software-Defined Networking Security: Pros and Cons." IEEE Communications Magazine 53(2015):73-79.

[4] Zaalouk, A., et al. "OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions." Network Operations and Management Symposium (NOMS), 2014 IEEE IEEE, 2014:1-9.

[5] Phaal, P., S. Panchen, and N. Mckee. "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks." RFC3176 2001.)

[6] I Mousavi, Seyed Mohammad, and M. St-Hilaire. "Early detection of DDoS attacks against SDN controllers." 2015 International Conference on Computing, Networking and Communications (ICNC) IEEE Computer Society, 2015:77-81.

[7] Braga, Rodrigo, E. Mota, and A. Passito. "Lightweight DDoS flooding attack detection using NOX/OpenFlow.." 38th Annual IEEE Conference on Local Computer Networks IEEE, 2010:408-415.

[8] Jin, Ruofan, and B. Wang. "Malware Detection for Mobile Devices Using Software-Defined Networking." Research and Educational Experiment Workshop (GREE), 2013 Second GENI IEEE, 2013:81 - 88.

[9] Kreutz, Diego, F. M. V. Ramos, and P. Verissimo. "Towards secure and dependable software-defined networks." Second Acm Sigcomm Workshop on Hot Topics in Software Defined Networking 2013:55-60.

[10] Shin, Seungwon, et al. "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks." Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security ACM, 2013:413-424.

[11] Wang, Haopei, L. Xu, and G. Gu. "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks." Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on IEEE, 2015.

[12] Lakhina A, Crovella M, Diot C. Mining anomalies using traffic feature distributions.[J]. Acm Sigcomm, 2005, 35(4):217-228.

[13] POX [Online]. Available at: http://www.noxrepo.org/pox/about-pox/

[14] Scapy. 2014, [Online]. Available at: http://www.secdev.org/projects/scapy/