EuCNC2016-SoftDefInfra 1570248444

# OpenFlow Flow Table Overflow Attacks and Countermeasures

Ying Qian
Dept. of Computer Science
East China Normal University
Shanghai, China
yqian@cs.ecnu.edu.cn

Wanqing You
Dept. of Computer Science
Southern Polytechnic State University
Marietta, GA, USA
wyou@spsu.edu

Kai Qian
Dept. of Computer Science
Kennesaw State University
Marietta, GA, USA
kqian@kennesaw.edu

***Abstract*— Software-defined Network (SDN) is proposed as a new concept in computer networks, which separates the control plane from data plane. And it provides a programmable network architecture that could facilitate network innovation rapidly. OpenFlow is a network protocol that standardizes the communications between OpenFlow controllers and OpenFlow switches. It is considered as an enabler of SDN. The flow table in OpenFlow switches plays a critical role in OpenFlow-based SDN, which stores the rules populated by the controllers for controlling and directing the packet flows in SDN. Nevertheless, they also become a new target of malicious attacks. This paper analyzes the flow table overflow attack, a type of denial of service attacks, and proposes a systematic way to mitigate the overflow in flow table.**

***Keywords—OpenFlow; flow table; overflow; attack; mitigation***

## I. INTRODUCTION

Software Defined Networks (SDN) separates the control plane from data plane and provides open, scalable, secure, and programmable network architecture, which can facilitate network innovation and can operate with different types of switches and at different protocol layers. SDN controllers and different types of switches can be implemented at different layers (L2-L4). The OpenFlow protocol is an open standard for SDN that specifies the communications between controllers in control plane and switches in data plane. OpenFlow controller has a global view of the whole network, while OpenFlow switch consists of a flow table for flow entries and secure channel for communicating with controller. The protocol of OpenFlow was proposed in 2009 and it has been receiving increasing adaptation by commercial networking devices and increasing deployment in campus and enterprise networks [1]. The *controller* is the heart of OpenFlow network and decides the packet flows in the *data plane* by assigning flow rule entries in the OpenFlow switches' flow tables. The data path of each OpenFlow switch has its flow table and each flow table consists of a finite set of flow entries. The flow table is a key component of OpenFlow switch. The performance of entire network can be severely affected by the malfunction of the flow table, such as resource exhaustion, rule conflicts, and malicious rule manipulation. The performance and the security of the OpenFlow flow table shall be well addressed.

Switches follow the flow rules in flow tables populated by the controller. The flow rules can either be set by the controller proactively, or generated by controller reactively per request from switch where a packet failing to match any existing rules. Each flow rule consists of three parts: (1) rule matching pattern section, which specifies the packet flow delivery from source to destination; (2) associated actions on packet process, e.g., controller may generate a new flow entry with "forward" action for the first packet in a new flow, with "drop" action to reduce traffic, or with a "modify" action to rewrite the packet header; and (3) statistics data that keep track of the number of times the rule has been used, length of each flow, and the recent time when the rule is used for removal reference.

OpenFlow SDN revolutionizes the network management and enables network innovations. There are more and more SDN security research activities on the deployment of novel security applications over OpenFlow network. For example, Gu et al [2] proposed the OpenFlow-based CloudWatcher for network security monitoring for secure dynamic cloud environment. However, the security of SDN network is also a very important and challenging task and few research works have been conducted on the security of key SDN components such as flow table.

In this paper, we focus on flow table overflow attack analysis, propose systematic method to mitigate flow table overflow and evaluate its efficiency and effectiveness. The paper is organized as follows: Section II gives an overview of related work. Section III provides the flow table attack analysis. Section IV presents our solution to the overflow attacks and its implementation. Section V performs an evaluation of the defense approach and discusses the results. Section VI concludes the paper.

## II. RELATED WORK

Security analysis for SDN is still an active research area. A number of related works had analyzed the SDN or OpenFlow vulnerability and identified security threats [3-7, 8-

11] and some of them also summarized general possible solutions to malicious attacks, such as FortNox and FRESCO [12-15] that extended NOX [15] with a security kernel and security programming interface. Kreutz et al. [6] analyzed and identified several threat vectors that might enable the exploit of SDN vulnerabilities. They have summarized SDN security kernel work that is capable of ensuring prioritized switch flow rules for security related applications. Wen at al. [16] also argued that minimum privilege should be put on applications. However, none of these works enforces the security of SDN itself such as flow table components.

## III. FLOW TABLE OVERFLOW ATTACK ANALYSIS

An asset-centric approach was utilized to help identify security threats in OpenFlow networks. We first identified essential assets in the networks, figured out possible security threats/vulnerabilities, and then analyzed the potential impacts on the network once the vulnerabilities were employed by malicious attackers. Some other methodologies can also be used to analyze the security threats of SDN, such as Microsoft's STRID method [3-5] and attack path [17].

Two security threats related to OpenFlow flow table and their potential consequences are shown in Table 1. The first threat involves the exhaustion of flow entries and the flow table overflows. The consequence of the threat affects the availability of the system, e.g. denial of new rule installation and thus causing packet loss. Another threat is related to the malicious manipulation of the rules in the flow table and its consequences affect the availability, integrity and confidentiality of the system. For example, a malicious app deployed in controller can insert a purposed flow rule into flow table or rewrite packet header proactively that would lead to rule conflict.

TABLE I. TWO SECURITY THREATS RELATED TO FLOW TABLE

| Asset | Threat | Consequence |
|---|---|---|
| Flow Table | Overflow | **Availability**<br>Denial of New Rule Installation;<br>Packet loss. |
| | Rule Insertion & Manipulation | **Integrity**: Rule Conflicts<br>**Confidentiality**: flow sniffing<br>**Availability**: packet loss |

In this paper we focus on the flow table overflow attacks that would lead to denial of service both on switches and controllers. When the flow table is overflowed, the switches can not take any more flow entries and the controller would not be able to reply to legitimate clients' requests in time or even worse, the controller can become unavailable due to the great volume of requests from attackers. As the controller only installs rules for packets that have no matched rules in flow table, it is only necessary to permute some packet header to cause the installation of new flow entries. The following are two ways that may cause overflow in flow table.

### A. Attacks from Malicious App on Controller

OpenFlow controller is the heart of entire network and takes charge of the packets that have no matched entries in flow table. After making decision to deal with the requests from switches, controller would install a new flow rule for each packet into flow table. Attacks would take place if a malicious app is deployed in controller to take care of messages from switches. Therefore, to reduce the chance that attack might come from internal, it is recommended to test applications carefully before deploying them on controller. Kostic et al. [18] proposed a NICE way to do such inspections. In a typical internal DoS attack a malicious app starts an infinite loop to install multiple new flow rules into flow table when receiving a Switch-Connected message. To overflow flow table, the source IP address and destination IP address are permuted in our experimentation so that a large number of new rules can be inserted.

### B. Attacks from Packets

While an internal threat is possible, such as the malicious app analyzed above, more often, attacks come from external attackers. External attackers can have more flexible methods to launch DoS attack, even to make DDoS if there are multiple attackers. To cause overflow in flow table, attackers can generate a large number of new packets and send them to the controller to result in installation of new rules for each packet, thus exhausting the flow table. In our experiment, we make use of *Scapy* [19] to craft a great number of UDP packets by permuting the source and destination port fields in packet header, and the send out the packets at user-defined rate.

## IV. EVICTION CONTERMEASURES

### A. General strategy such as rate limit, timeout adjustment, etc.

The objective of DoS attacks in OpenFlow is to over-consume the resources of controller and/or switch, as well as the communication bandwidth between nodes in network, and thus makes the normal function of the network unavailable. For this purpose, a great volume of traffic is involved. The QoS framework is based on queues and rate limiters. OpenFlow 1.0 already implements queues to support slicing feature, which can be achieved via FlowVisor to slice the whole network into multiple logical sub-networks. The other aspect of QoS is rate limiting. A rate limiter controls the rate of packets passing through it. In order to mitigate overflow of the assets listed in previous section, several use cases of rate limiting can be taken into consideration, such as limiting the amount of traffic that a single port can send to the switch, limiting the amount of packets sent to the controller, or limiting the number of rules that controller insert into switch in a short period time. In our experiment, we also notice that flow timeout, which decides how long a flow entry can inhabit in flow table, can be adjusted to mitigate the negative effect of DoS. It is discovered that a longer timeout lends itself to the flow's time-to-live property, thus making flow tables easier to be overflowed.

### B. Eviction algorithm

To decrease the impact of DoS, a more effective way is a reactive and dynamic event-driven method. On receiving the notification switch sends, the event handler can be triggered to take respective actions. For example, evict rules from the flow

tables, vacate some flow entries for accepting new rules, and thus mitigate the overflow of the flow table. Based on this principle, we constructed independent modules in controller to monitor the traffic went through controller. The modules utilized the messages between switches and controllers. The messages used and actions taken when receiving the messages were summarized as Table 2 and Table 3.

TABLE II. MESSAGES USED IN LEARNING SWITCH MODULE

| Message | Action |
|---|---|
| PACKET_IN | Monitor the changing of flow table capacity |
| FLOW_MOD | New rule installed |
| FLOW_REMOVED | Count the flows removed from flow table |
| STATS_REPLY | Poll and analyze the statistics from switch |

TABLE III. MESSAGES USED IN FLOW CHECKING MODULE

| Message | Action |
|---|---|
| PACKET_IN | Source address validation, calculate parameters needed to determine if this source address is malicious or not, find out flows that are malicious |

As a common network security issue, complete elimination of DoS/DDoS attacks to SDN seems to be impossible. Nevertheless, efforts shall be made to mitigate the effect of this kind of attack, as discussed in [20].

The event-handler eviction model presented in this paper as was shown in Fig.1. The learning switch module listened to the events to take care of the basic monitoring. It handled events as following:

- *forwardAsLearningSwitch(): handle PACKET_IN messages to add new rule into flow table, send out flow-mod messages to accumulate the number of rules inserted into flow table, and track the capacity change of flow table.*
- *statsReplyHandler(): hander STATS_REPLY messages to analyze the statistics from switch.*
- *flowRemovedMsgHandler(): take care of FLOW_REMOVED messges to accumulate the number of rules being removed out and record the reason for removing.*

A FlowChecking module was implemented to analyze every new incoming packet. The jobs of this module: Validate the source address by querying log and add it to black list if it does not pass the validation and remove flows that exceed Packet per Second (PPS) threshold and packet bytes threshold.

- *monitorFlows(): handle PACKET_IN messages to validate the source address, calculate PPS value and count the bytes has already sent by the source.*

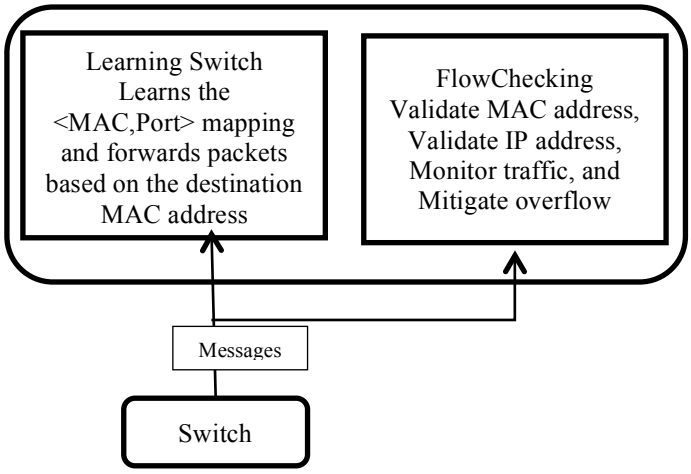


Fig. 1. Event-handler model.

The algorithm used in FlowChecking module is quiet straightforward, however the pre-defined thresholds should be adjusted according to the real system settings, shown in Fig. 2.

To overflow flow table, attackers may also need to take advantages of some fields in the flow entries' matching fields. The field *idle_timeout* and *hard_timeout* in the matching fields specify the time to remove the flow rule if it is not used during a period of time and how long a rule can stay in flow table, respectively. An infinite *idle_timeout* or *hard_timeout* will make a rule to stay in flow table forever. Second, the priority in matching fields. A rule with higher priority will stay longer than its counterparts that have lower priority. These properties would be considered in further work.

```
// Input: incoming packet
// Alert suspicious behavior and take action
checkingFlow(Switch switch, PacketIn pkt):
    if (! pkt.src is valid):
        remove flows for this address from switch;
        add new entries to drop new incoming packets from
this src;
        add pkt src to black list;
    else:
        if (exceed pre-define threshold PPS)||
          (exceed pre-define threshold bytes count):
            remove flows for this address from switch;
            add new entries to drop new incoming packets
from this src;
            add pkt src to black list;
```

Fig. 2. FlowChecking algorithm.

## V. EVALUATION

### A. Setup and Emulation Environment

In this section, we provided an overview of the simulation environment, the packets generation tools and the network configurations that were used for the evaluation of the DoS attacks by external attackers.

- Simulation Environment: We used the Mininet framework to create virtual networks, which implements Open vSwitch that we should need in

emulation. Mininet is an easy and instant tool to create virtual network, running real kernel switch and application code, on a single machine with single commands. It provides Command Line Interface for developers to conveniently manipulate each specific node in the network to help a performance analysis, such as bandwidth, dump packets. Commands *ovs-vsctl* and *ovs-ofctl* are most used in our case, which are used to configure parameters in Open vSwitch and flow table respectively.

- Packet Generation: To simulate attacker, the packet generation tool *Scapy* is used, which is embedded in Mininet framework. It is a Python-based framework that can be used to craft packets in different network layers. It also provides a rich number of methods that enable the developers to send packets under specific condition, such as specify packets sending rate.
- Network Setup: The network setup consists of two parts. One is the topology structure used in the experimentation, which includes 20 hosts, an Open vSwitch and a Beacon-based controller. More details about Beacon can be obtained in Erickson's work [21]. Each node has a unique connection to switch, and the switch is under the control of controller. To simulate the ways to attack flow table from client side, we have scripts listed in Table 4. Attackers may take control of one or more hosts. If there are several hosts being controlled by attackers, there is DDoS attack, which will cause a more serious attack.

### B. Experimentation Description

The experiment carried out in this paper was explained in this part. A topology with 20 hosts, one switch and one controller was set up first. Then, the attacking scripts listed in Table 4 were executed in a sequence. When the experiment arrived at the fifth, tenth and fifteenth minutes during the whole time period, they were triggered respectively. In other words, all attacks were on-going at the fifteenth minute. While the attacks were taking, some data were collected to analysis packet loss and bandwidth consumed between legitimate clients.

### C. Experimentation Description

The experiment carried out in this paper was explained in this part. A topology with 20 hosts, one switch and one controller was set up first. Then, the attacking scripts listed in Table 4 were executed in a sequence. When the experiment arrived at the fifth, tenth and fifteenth minutes during the whole time period, they were triggered respectively. In other words, all attacks were on-going at the fifteenth minute. While the attacks were taking, some data were collected to analysis packet loss and bandwidth consumed between legitimate clients.

### D. Results

With a limited storage of flow table, attackers are able to overflow flow table easily by sending out a large number of packets to controller and causing the installation of new rules for each packet. The results include packets loss and service delaying, and so on. As shown in Fig.3, the flows kept growing quickly as the attack came from outside without any detecting or defensing strategies. The result was that the storage space of flow table would be consumed largely by the useless flow rules generated for attackers' packets. With this FlowChecking module, the useless rules would be removed to release some of the space of flow table to mitigate flow table overflow.

TABLE IV. SIMULATE WAYS TO DOS OPENFLOW FLOW TABLE IN CLIENT SIDE

| Script name | Attacking way | Result |
|---|---|---|
| macflood.py | Forged MAC | Overflow flow table and information disclosure |
| multidst.py | Use Scapy to generate huge traffic and send out from same source (assume the source host is hacked) | Overflow flow table and traffic congestion |
| udp-multi.py | Use Scapy to generate huge traffic and send from same source (assume hacked) to same destination but uses different ports every time | Overflow flow table and traffic congestion |

In Fig. 3 (FT: flow table), the number of flow entries in flow table was polled every time when controller installed a hundred of rules in flow table. Without the FlowChecking module, more rule entries were inserted into flow table and made the size of flow table grew fast, which led to the overflow of flow table. In the pt12 of Fig.3, when all attacks were on-going, without FlowChecking, the number of flow entries reached a peak, while most of the useless flow entries were removed by FlowChecking.

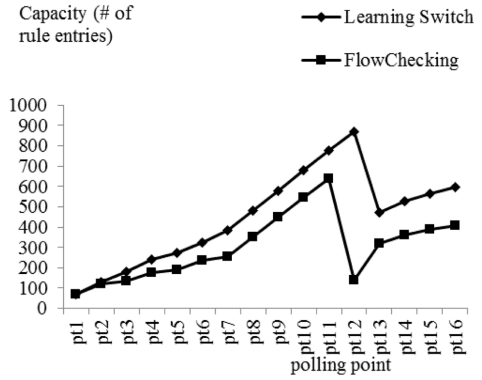


Fig. 3. Flow table capacity change.

If multiple client hosts are controlled by attackers and become zombies, the attacker can initiate a Distributed DoS attack. We collected data of packet loss and the bandwidth consuming between legitimate clients during the attacks. In Fig. 4, the bandwidth consuming between a pair of legitimate clients was shown. The two lines were identical to each other; the bandwidth was consumed mostly when all the attacks were ongoing, in other words, in the fifteenth minute during the whole experimentation. According to Fig. 4, we can conclude that the event handling module has no significant impact on

the performance of the communications between network nodes. Legitimate clients still communicate with each other as if no more handling modules are involved.

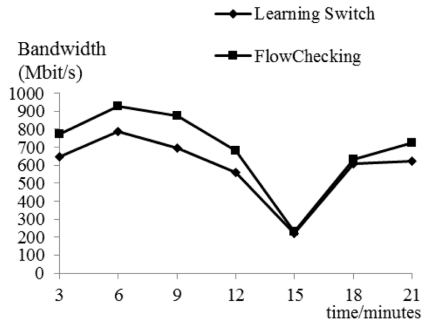


Fig. 4. The impact on bandwidth.

However, it would have a trivial impact on the packet loss excluding some noisy points, shown in Fig.5. Because the system proposed in this paper would report any suspicious behavior in the network and take actions. The data of packet loss were also collected during the whole experimentation. Data was polled every time when fifteen packets were sent to a specific port. We made the query to get the number of received packets. When employing the FlowChecking module, the packets loss was a little higher.

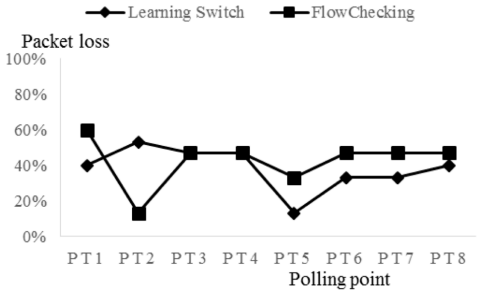


Fig. 5. Packet loss.

In summary, the system proposed in this paper provides a proactive event-driven method to mitigate overflow in flow table. By applying this strategy, the advantages are as following: the overflow frequency is reduced a lot; most of the useless flow entries are removed; no obvious impact on the communication between legitimate clients. The disadvantage is that the packet loss is a little bit higher.

## VI. Conclusion

In this paper, we analyze the flow table overflow attack, which could significantly degrade the SDN performance, and showed the feasibility of the attacks with experiment data. We propose a novel cost efficient and effective dynamic solution to defend against the attacks. The efficiency and effectiveness of the proposed solution has been evaluated in the simulated SDN networks using Mininet.

In the future, we will conduct the security analysis for other important assets in the OpenFlow-based SDN, consider the useful attributes in flow table' matching fields, then design defense solutions。

## References


[1] Maturing of OpenFlow and Software-Defined Networking through Deployments, http://yuba.stanford.edu/~srini/papers/comnet13.pdf.

[2] 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.

[3] G. Eason, Rowan Kloti, Vasileios Kotronis, Paul Smith, OpenFlow: A Security Analysis, IEEE ICNP 2013.

[4] R. Kloti, V. Kotronis, P. Smith, OpenFlow: A Security Analysis, IEEE ICNP 2013.

[5] ftp://ftp.tik.ee.ethz.ch/pub/students/2012-HS/MA-2012-20.pdf, 2013.

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

[7] K. Benton, L. J. Camp, C. Small, OpenFlow Vulnerability Assessment, SIGCOMM, 2013

[8] A. Khurshid, X. Zou, W. X. Zhou, M. Caesar, and P. B. Godfrey, Veriflow: Verifying network-wide invariants in real time. In *Proceedings of 10th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'13, USA, April 2-5, 2013.

[9] J. H. Jafarian et al., "Openflow random host mutation: transparent moving target defense using software defined networking," in Proc. Of HotSDN, 2012.

[10] A. Khurshid et al., "VeriFlow: verifying network-wide invariants in real time," in Proc. of HotSDN, 2012.

[11] M. Kobayashia, S. Seetharamanb, G. Parulkarc, G. Appenzellerd, J.Littlec, J. van Reijendamc, P. Weissmannb, N. McKeownc, Maturing of SDN Security Seminars 2012, http://www.openflowsec.org/SDN_SecuritySeminar_Feb2012.pdf

[12] 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.

[13] S. Shin, et al. , "FRESCO: Modular Composable Security Service for Software-Defined Networks, Internet Society, NDSS, 2013

[14] 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.

[15] 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.

[16] X. Wen, Y. Chen, C. Hu, C. Shi, Y. Wang, Towards a Secure Controller Platform for OpenFlow Applications, SIGCOMM 2013.

[17] Y. Chen, B. Boehm, L. Sheppard, Value Driven Security Threat Modeling Based on Attack Path Analysis, http://sunset.usc.edu/events/2006/CSSE_Convocation/publications/ChenValueBasedSecurityThreatModel.pdf

[18] 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.

[19] http://www.secdev.org/projects/scapy/doc/usage.html

[20] OpenFlow (D)DoS Mitigation http://www.delaat.net/rp/2013-2014/p42/report.pdf

SDN Solution aids DDoS attack detection and mitigation. http://packetpushers.net/openflow-1-0-actual-use-case-rtbh-of-ddos-traffic-while-keeping-the-target-online/

[21] 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.