Software-defined network flow table overflow attacks and countermeasures

Wanqing You* and Kai Qian

Department of Computer Science,
Southern Polytechnic State University, USA
Email: wyou@spsu.edu
Email: kqian@spsu.edu
*Corresponding author

Ying Qian

Department of Computer Science,
East China Normal University, China
Email: yqian@cs.ecnu.edu.cn

Abstract: Software-defined network (SDN) is proposed as a new concept in computer networks, which separates the control plane from the data plane. And it provides a programmable network architecture that could facilitate rapid network innovation. OpenFlow is a network protocol that standardises 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 analyses the flow table overflow attack, a type of denial of service attacks, and proposes a novel eviction algorithm, dynamic in/out balancing with least frequently used eviction (DIOB/LFU), at service level to defend against the flow table overflow attacks.

Keywords: OpenFlow; flow table; overflow; attack; mitigation.

Reference to this paper should be made as follows: You, W., Qian, K. and Qian, Y. (2016) 'Software-defined network flow table overflow attacks and countermeasures', *Int. J. Soft Computing and Networking*, Vol. 1, No. 1, pp.70–81.

Biographical notes: Wanqing You received her Bachelor degree from Xiamen University in 2013. She received her Master degree from Department of Computer Science from Southern Polytechnic State University. She was a Research Assistant while doing her Master's. Her research interests include mobile, network security, software defined networking (SDN), and cloud computing. She is currently a Software Engineer in Liaison Tech.

Kai Qian received his PhD in Computer Science from University of Nebraska, Linton, USA in 1990. He is a Full Professor of Computer Science at Southern Polytechnic State University, USA. His research interests include mobile and network security, and advanced learning technology.

Ying Qian received her Bachelor degree from Department of Electronics Engineering, Shanghai Jiao Tong University, Shanghai, China in 1998. She received her Master and PhD in Department of Electrical and Computer Engineering from Queen's University, Kingston, Ontario, Canada in 2005 and 2010, respectively. She is an Associate Professor in the Department of Computer Science and Technology, at East China Normal University, Shanghai, China. Before she joined East China Normal University in 2013, she was a computational Scientist in the Supercomputer Laboratory at King Abdullah University of Science and Technology from 2009. Her research interests include software defined network, high-performance scientific computation, and parallel programming.

1 Introduction

Software defined networks (SDNs) separates the control plane from the data plane and provides an 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 the 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 (http://yuba.stanford.edu/~srini/papers/comnet13.pdf). The controller is the heart of the 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 an 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 manipulate incoming packets following 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 fails to match any existing rules. Each flow rule consists of three main parts and some other fields:

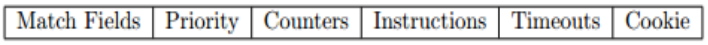
1 rule matching pattern section, which specifies the packet flow delivery from source to destination

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

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, which is shown in Figure 1.

The OpenFlow switch and controller communicate with many important event messages, such as modify-state, read-state, send-packet, flow-modify, port-status, echo request/reply, which are classified as controller-to-switch messages, asynchronous messages and symmetric messages. These messages can be used to notify event handler in controller to handle and process these events.

Figure 1 Main component in flow table



OpenFlow SDN revolutionises the network management and enables network innovations. There are an increasing number of SDN security research activities on the deployment of novel security applications over OpenFlow network. For example, Shin and Gu (2012) propose the OpenFlow-based CloudWatcher for network security monitoring for secure dynamic cloud environment. However, the security of SDN network is also an 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 and propose a novel DIOB/LFU eviction algorithm and evaluate its efficiency and effectiveness. The paper is organised as follows: Section 2 gives an overview of related work. Section 3 provides the flow table attack analysis. Section 4 presents our solution to the overflow attacks and its implementation. Section 5 performs an evaluation of the defense approach and discusses the results. Section 6 concludes the paper.

2 Related work

Security analysis for SDN is still an active research area. A number of related works have analysed the SDN or OpenFlow vulnerability and identified security threats (Eason et al., 2013; Kloti et al., 2013; ftp://ftp.tik.ee.ethz.ch/pub/students/2012-HS/MA-2012-20.pdf; Kreutz et al., 2013; Benton et al., 2013; Khurshid et al., 2012, 2013; Jafarian et al., 2012; Kobayashia et al., 2012) and some of them also summarise general possible solutions to malicious attacks, such as FortNox and FRESCO (SDN Security Seminars, http://www.OpenFlowsec.org/SDN_SecuritySeminar_Feb2012.pdf; Porras et al., 2012; Shin et al., 2013a, 2013b) that extends NOX (Gude et al., 2008) with a security kernel and security programming interface. Kreutz et al. (2013) analysed and identified several threat vectors that may enable the exploiting of SDN vulnerabilities. They have summarised SDN security kernel work that was capable of ensuring prioritised switch flow rules for security related applications and for other all remaining applications. The first type represents specialised programs used to ensure security control policies in the network, such as to guarantee or restrict specific accesses to the network or take actions to control malicious data traffic. Flow rules generated by security applications have a high priority over the others. Wen at al. (2013) 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.

Eason et al. (2013) had conducted an OpenFlow security analysis about denial of service (DoS) attacks that target the OpenFlow controller and/or the switches, aiming at crippling the communication between the components or the components themselves. Especially, they explored one aspect of flow table overflow attack by experiments and showed the impact on packet loss. They proposed three categories of defense methods, including

1 rate limiting, event filtering, packet dropping, and timeout adjustment

2 flow aggregation: proactively flow aggregate such that each flow rule matches multiple network flows

3 access control: enforcing access control lists in the form of flow rules in the flow table.

Nevertheless, their implementation and evaluation of the proposed defense methods are insufficient.

3 Flow table overflow attack analysis

An asset-centric approach was utilised to help identify security threats in OpenFlow networks. We first identified essential assets in the networks, figured out the possible security threats/vulnerabilities on those assets, and then analysed the potential impacts on the network once the vulnerabilities were employed by malicious attackers. Some other methodologies can also be used to analyse the security threats of SDN, such as Microsoft's STRID method (Eason et al., 2013; Kloti et al., 2013; tp://ftp.tik.ee.ethz.ch/pub/students/2012-HS/MA-2012-20.pdf) and attack path (Chen et al., 2006).

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 overflows the flow table. 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 the 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.

In this paper, we focus on the flow table overflow attacks that would lead to DoS both on switches and controllers. When the flow table is overflowed, the switches cannot 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 make an overflow in flow table.

Table 1 Two security threats related to flow table

Asset	*Threat*	*Consequence*
Flow table	Overflow	Availability
		Denial of new rule
		Installation
		Packet loss
	Rule insertion and manipulation	Integrity
		Rule conflicts
		Confidentiality
		flow sniffing
		Availability
		packet loss

3.1 Attacks from malicious app on controller

OpenFlow controller is the heart of the 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. Canini et al. (2012) 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.

3.2 Attacks from packets

While an internal threat is possible, such as the malicious app analysed 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 (http://www.secdev.org/projects/scapy/doc/usage.html) 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.

4 Eviction conter measures

4.1 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 make the normal function of the network unavailable. For this purpose, a great volume of traffic is involved. The techniques of queues and rate limiting can be applied to ensure a high performance system. 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. 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 noticed 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.

4.2 Eviction algorithm

To decrease the impact of DoS, a more effective way is a reactive and dynamic event-driven method. On receiving the notification that the flow table is full, an event handler can be triggered to 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 propose an eviction algorithm, named dynamic in/out balancing with least frequently used (DIOB/LFU) eviction algorithm. The algorithm utilises the messages between switches and controllers. The messages that are used and actions taken when receiving the message in our DIOB/LFU algorithm are summarised as Table 2.

Table 2 Messages used in our DIOB/LFU algorithm

Message	*Action*
PACKET_IN	New rule installation
ERROR	t1: time of first overflow $\Delta r = r_{in} - r_{out}$ t2: time of second overflow
FLOW_MOD	$r_{in} = r_{in} + 1$
FLOW_REMOVED	$r_{out} = r_{out} + 1$
STATS_REPLY	Remove entries to mitigate overflow

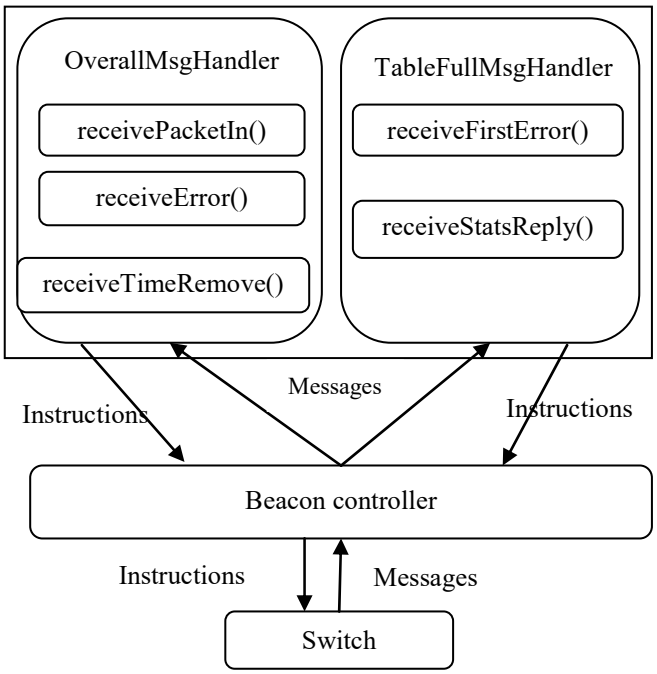
Notes: r_{in} is the number of rules installed;
r_{out} is the number of rules evicted by the algorithm.

As a common network security issue, complete elimination of DoS/DDoS attacks to SDN is proved to be impossible. Nevertheless, efforts shall be made to mitigate the effect of this kind of attack, as discuss in some work (http://www.delaat.net/rp/2013-2014/p42/report.pdf; http://packetpushers.net/OpenFlow-1-0-actual-use-case-rtbh-of-ddos-traffic-while-keeping-the-target-online/).

The event-handler eviction model is presented in this paper as shown in Figure 2. An overall event handler registers for the general events to take care of the basic monitoring. It handles events as following:

- receivePacketIn(): handle PACKET_IN messages to add new rule into flow table, and send out flow-mod messages to accumulate the number of rules inserted into flow table. And the counter for rules installed is increased by one.
- receiveError(): handle ERROR messages to accumulate the number of received error messages.
- receiveTimeoutRemove(): take care of FLOW_REMOVED messages to accumulate the number of rules being removed out because of timeout. And the number of rules removed is increased by one.

Figure 2 Event-handler model



A TableFullMsgHandler is implemented to evict useless and malicious rules from the flow table in order to mitigate the overflow of the flow table.

- receiveFirstError(): handle ERROR messages to calculate the difference between rule_in and rule_out in a period of time, and send out flow statistics request
- receiveStatsReply(): handle STATS-REPLY messages, retrieve statistics information of rules from switch's flow table and apply DIOB/LFU eviction algorithm to remove out rules.

To overflow flow table, attackers need to take advantages of some fields in the flow entries' matching fields. The field *idle_timeout* and *hard_timeout* in the matching fields specifies 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. Another field that can be exploited is the rule priority. A rule with higher priority will stay longer than its counterparts that have lower priority.

In our eviction algorithm, we only made use of some of the parameters and the messages listed above. To make flow table in a relatively safe stage, we aim to remove useless and spared rules when receiving 'table full error' messages from switch. The question here is how many rules should be removed to achieve such balance. To do this, our DIOB/LFU algorithm collects the number of rule-in and rule-out, and ensures that the difference between rule-in and rule-out is less than or equals to zero such that the flow table will not be overflowed frequently. This simple principle is used in our algorithm.

The formulation used to determine the number of rules to evict is illustrated as:

$$\Delta r = r_{in} - r_{out}$$

The main concept of our eviction algorithm is to remove at least Δr rules when receiving 'table full error' messages. The rules that should be deleted have the following properties:

- idle_timeout = ∞
- counter = 0.

The time complexity of our algorithm is $O(n^2)$, because we need to look through the whole flow table to find out the rule that has minimum usage frequency to be deleted, and there are at least Δr ($\Delta r <= n$) rules to be removed from flow table to make a balance. With DIOB/LFU, the frequency of overflow is decreased significantly. Of course there is a trade-off between performance and efficiency. The results are illustrated in the next section to compare the effect of DoS with and without our algorithm.

5 Evaluation

5.1 Setup and emulation environment

In this section, we provide 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 (http://mininet.org/) 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.

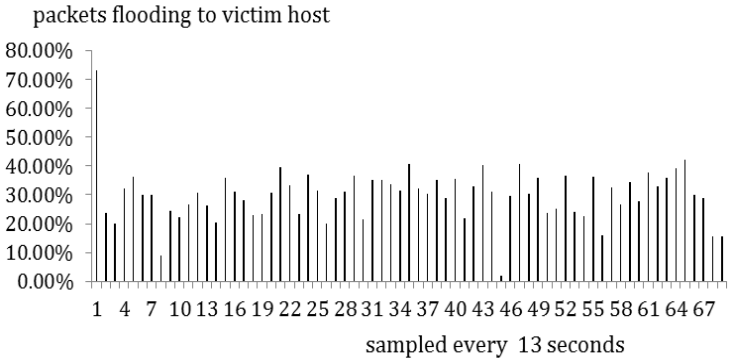
- Packet generation: To simulate attacker, the packet generation tool Scapy is used, which is embedded in Mininet framework. It is a python-based framework that can be used to craft packets in different network layers. It also provides a rich number of methods that enable the developers to send packets under specific condition, such as sending rate.

- Network setup: The network setup consist of two parts. One is the topology structure used in the experimentation, which includes four hosts, an Open vSwitch and a Beacon-based controller. More details about Beacon can be obtained in Erickson's work (Erickson, 2013). Each node has an unique connection to switch, and the switch is under the control of controller. To simulate, attackers take control of one (h1 in our case, and h2 would be the victim. We would test the bandwidth between two legitimate clients h3 and h4 before and during the attack) or more hosts. If there are several hosts being controlled by attackers, there is DDoS attack, which will cause a more serious attack. For this purpose, we have a topology with eight hosts. The other part of network setup is to configure the size of flow table with command *ovs-vsctl*.

5.2 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. With our eviction algorithm DIOB/LFU, the frequency of overflow is decreased. We collected the number of table full error in a fixed time slot to evaluate how this DIOB/LFU algorithm could mitigate flow table overflow. Before employing our eviction algorithm, the flow table of switch was overflowed time to time because the large number of packets was sending, while there was only one overflow notification during a long period of time after applying DIOB/LFU. However, the rule eviction had impact on bandwidth between legitimate clients, which may make packet loss ratio slightly higher during the event handling. On the other hand, this also held back the large number of packets sent by attackers. The statistics shows that nearly 32.29% of the packets were flooding to overwhelm victim hosts illustrated in Figure 3, while only 1.7% of packets were received by victim hosts after applying DIOB/LFU, according to the 70 sample data collected.

Figure 3 Packets flooding to victim hosts per 13 seconds



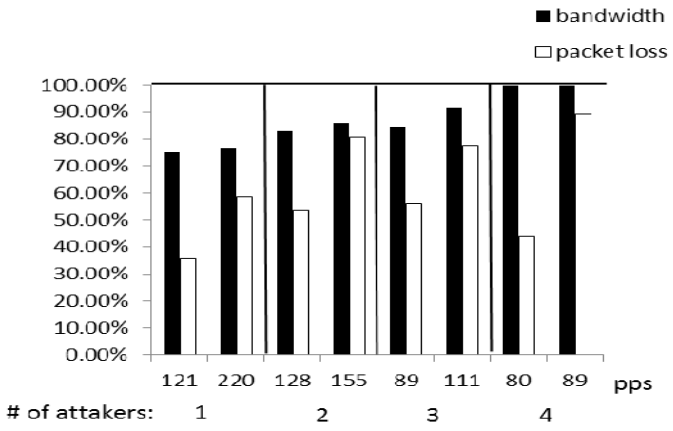
Chances are that attackers control multiple client hosts are to initiate a distributed DoS attack. Therefore, we also tested different number of attackers sending packets at the same time with different sending rate and collected data of packet loss and the bandwidth changes of legitimate clients during the attacks. In Figure 4, we had four groups of testing environment. Inside each group, we had attackers send packet in different rate. According to Figure 4, we can conclude that

1 sending packets at a faster rate will consume more bandwidth, the comparisons referring to each column

2 with more distribution, i.e., with more participated attackers at the same time, the phenomenon of packet loss and bandwidth consume are more severe.

In summary, this dynamic in/out balancing with LFU provides a proactive event-driven method to mitigate overflow in flow table. It is based on messages sent from switch to controller. By applying this eviction algorithm, the overflow frequency is reduced a lot, while the packet loss is a little bit higher during the rule removing.

Figure 4 The impact on packet-loss and bandwidth with different numbers of attackers at different packet sending rate



Note: pps: packet per second

6 Conclusions

In this paper, we analysed the flow table overflow attack, which could significantly degrade the SDN performance, and showed the feasibility of the attacks with experiment data. We proposed a novel, efficient, effective and dynamic solution to defend against the attacks. The efficiency and effectiveness of the proposed solution had 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, design defense solutions, and perform the evaluation using the SDN simulation in Mininet.

References

(2013) [online] ftp://ftp.tik.ee.ethz.ch/pub/students/2012-HS/MA-2012-20.pdf.

Benton, K., Camp, L.J. and Small, C. (2013) 'OpenFlow vulnerability assessment', *SIGCOMM* [online] http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p151.pdf.

Canini, M., Venzano, D., Peresini, P., Kostic, D. and Rexford, J. (2012) 'A nice way to test OpenFlow applications', *Proc. the 9th USENIX Conference on Networked Systems Design and Implementation*, April.

Chen, Y., Boehm, B. and Sheppard, L. (2006) 'Value driven security threat modeling based on attack path analysis' [online] http://sunset.usc.edu/events/2006/CSSE_Convocation/publications/ChenValueBasedSecurityThreatModel.pdf.

Eason, G., Kloti, R., Kotronis, V. and Smith, P. (2013) 'OpenFlow: a security analysis', *IEEE ICNP*.

Erickson, D. (2013) 'The beacon OpenFlow controller', *Proc. of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ACM, pp.13–18, doi:10.1145/2491185.2491189.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N. and Shenker, S. (2008) 'NOX: towards an operating system for networks', *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 3, pp.105–110, doi:10.1145/1384609.1384625.

Jafarian, J.H. et al. (2012) 'OpenFlow random host mutation: transparent moving target defense using software defined networking', in *Proc. of HotSDN*.

Khurshid, A. et al. (2012) 'VeriFlow: verifying network-wide invariants in real time', in *Proc. of HotSDN*.

Khurshid, A., Zou, X., Zhou, W.X., Caesar, M. and Godfrey, P.B. (2013) 'VeriFlow: verifying network-wide invariants in real time', in *Proceedings of 10th USENIX Symposium on Networked Systems Design and Implementation*, *NSDI '13*, Lombard, Illinois, USA, April 2–5.

Kloti, R., Kotronis, V. and Smith, P. (2013) 'OpenFlow: a security analysis', *IEEE ICNP*.

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

Kreutz, D., Ramos, F. and Verissimo, P. (2013) 'Towards secure and dependable software-defined networks', *Proc. the Second ACM SIG- COMM Workshop on Hot Topics in Software Defined Networking*, ACM, pp.55–60, doi:10.1145/2491185.2491199.

Maturing of OpenFlow and Software-Defined Networking through Deployments (2013) [online] http://yuba.stanford.edu/~srini/papers/comnet13.pdf.

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

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M. and Gu, G. (2012) 'A security enforcement kernel for OpenFlow networks', *Proc. the First Workshop on Hot Topics in Software Defined Networks*, ACM, pp.121–126, doi:10.1145/2342441.2342466.

SDN Security Seminars [online] http://www.OpenFlowsec.org/SDN_SecuritySeminar_Feb2012.pdf.

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

Shin, S. and Gu, G. (2012) 'Cloudwatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)', *Proc. 20th IEEE International Conference on Network Protocols* (*ICNP*), IEEE, pp.1–6, doi:10.1109/ICNP.2012.6459946.

Shin, S. et al. (2013a) 'FRESCO: modular composable security service for software-defined networks', *Internet Society NDSS*.

Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G. and Tyson, M. (2013b) 'FRESCO: modular composable security services for software-defined networks', *Proc. Network and Distributed Security Symposium*.

Wen, X., Chen, Y., Hu, C., Shi, C. and Wang, Y. (2013) 'Towards a secure controller platform for OpenFlow applications', *SIGCOMM* [online] http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p171.pdf.