Protecting Openflow Switches against Denial of Service Attacks

Ayman M. Bahaa-Eldin
Faculty of Computer Science
Misr International University
On leave from Ain Shmas University
ayman.bahaa@eng.asu.edu.eg

Ebada Essam-Eldin ElDessouky
Department of Management Information Systems, Faculty of Engineering and Natural Sciences,
Kadir Has University
ebada.platenium@gmail.com

Hasan DAĞ
Department of Management Information Systems, Faculty of Engineering and Natural Sciences,
Kadir Has University
hasan.dag@khas.edu.tr

***Abstract*—This paper presents a novel approach to protect Openflow switches against a type of Denial of Service (DoS) attacks. Openflow switches are the core of Software Defined Networks (SDN) and they are very flexible, programmable, and can be used for several functionalities within a network. As the control algorithm of the switch is implemented on a separate computer (Controller), this software can be implemented on any part of the network packet including Layers 2, 3, and 4 headers. Therefore, an Openflow switch can work as a conventional switch, a router or a firewall. The open design of Openflow makes it vulnerable to several types of DoS attacks. One of those attacks is to overwhelm the switch flow table with entities larger than its buffer making legitimate packets unable to traverse the switch. The proposed approach depends on a Sandbox like model, where a second switch and controller is implemented and all new packets with no matching rules are forwarded to the Sandbox. The Sandbox clone is monitored and controlled, so a forwarding rule is always created on the Sandbox switch and transferred only to the working switch when it is classified as a normal rule. Otherwise, a cleanup operation is executed periodically on the sandbox switch to remove malicious rules. Rules are classified based on the statistics entries already existing in Openflow switches flow table. The proposed approach is simple and does not need any extra memory or modifications in the switches. It is proven to mitigate this type of DoS attacks**

Keywords*— *Software Define Networks, SDN, DoS, Openflow, Security

I. INTRODUCTION

Software Defined Networks (SDN) [1] had introduced a paradigm shift in the networking industry. Instead of having a distributed control architecture, it consolidates all the control in a single node called the "Network Controller" which is simply a software running on a commercial server platform. Network forwarding devices no longer participate in the network control and only forward packets based on the set of rules installed from the network controller. The network controller programs the forwarding rules of the forwarding devices through the "Openflow protocol" [2] and hence the network forwarding devices are called "Openflow switches". In order to implement a new network functionality, only a new application is needed to be installed over the network controller while no change is needed from the switching devices side. Figure 1 explains the architecture difference between traditional network and a Software-Defined Network [3].

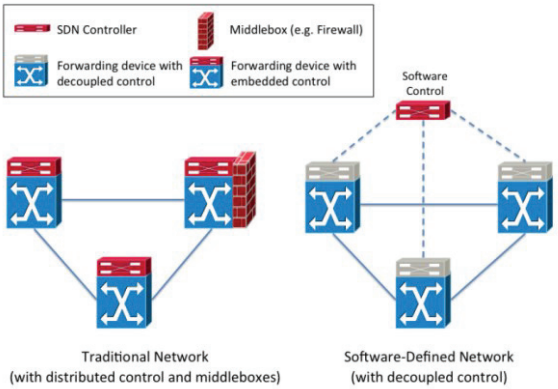

Fig. 1. Architecture Difference between Traditional IP networks and Software-Defined Network [3]

Figure 2 shows the overall architecture of an SDN.
The openness of SDNs and especially the core component of Openflow makes the architecture vulnerable to several security attacks. One of those major attacks are the Denial of Service (DoS) attacks. DoS attacks can target the Openflow switches, the Openflow controller, or the communication link between them.
Although the flow of configuration and information exchange between the controller and the switch is protected by Secure Socket Layer protocol (SSL) and utilizes public key certificates, still the system is vulnerable to DoS attacks which do not require any access to the exchanged information, rather in those attacks, the open nature of the protocol architecture is exploited to overwhelm the network causing it to become out of service.
Several techniques had been provided to mitigate such attacks. Detecting and mitigating attacks and intrusions is a major work in the field of computer networking. However, the focus was on the applications and host systems [4], [5], [6] and [7].
Some work in the field of protecting the network itself, especially for the trusted routing and DoS attacks on the network core was also introduced in [8] and [9]

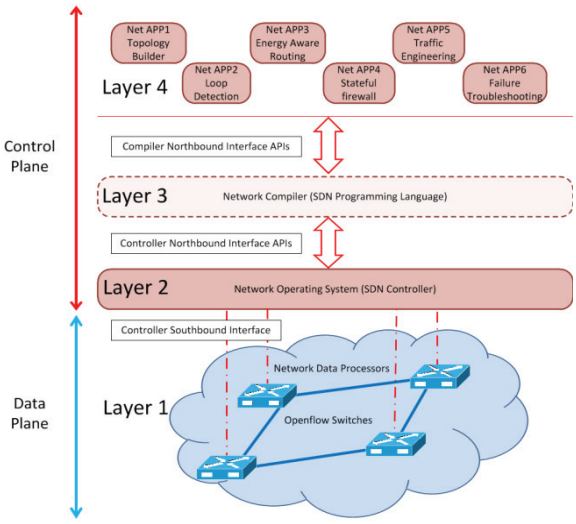

Fig. 2. Layered architecture of an SDN network

In this work, the proposed mitigation model is based on the idea of Sandboxing. A sandbox is a method to force malicious activities to be performed in a controlled environment. The activities are monitored and classified to be legitimate or illegal. Legitimate activities are sent back to the working environment. Sandboxing was used in protecting computers and virtualization environments [10], and [11]

In this paper, a DoS attack targeting an Openflow switch is described followed by a novel mitigation model to protect the switch against such an attack. The proposed model is discussed and shown to successfully protect the switch against the assault.

II. OPENFLOW

Openflow [2] is a protocol and a methodology of providing programmable networks. It had been originally proposed as a scientific experimentation platform enabling researchers to design and implement experimental protocols over an Ethernet switch with flow tables. The main idea in Openflow is to separate the forwarding algorithm (control plane) from the forwarding circuit (data plane).

The architecture of Openflow is simple, as shown in figure 3; an Ethernet switch is used as the core component of the network. The switch is divided into 2 planes: the first one is a hardware switching fabric with a flow table that indicates a forwarding action to be taken. The forwarding (Switching) action is to copy the incoming frame from an input port to a specific output port.

In contrast to a classical Ethernet switch, where the switch flow table is built by a software running on the switch itself, (in Layer 2, or data link layer) in a plug and play using the Address Resolution Protocol (ARP) [12]. Openflow flow tables are built using the software running on the controller, they are more complex, and a switch can have many flow tables organized in a pipeline.

The Controller is usually a commodity computer (a PC or a Server) running an algorithm that builds the flow table(s) of the Openflow switch.

A. Openflow Controller

The Openflow controller is in charge of deciding how to handle switches without substantial flow entities. In addition, it deals with the switch stream table by including and evacuating flow entities over the maintained channel, which utilize the Openflow protocol. The controller is in charge of keeping up all the system conventions and strategies, dispersing fitting guidelines to the network devices. The controller incorporates system knowledge. The switch must have the capacity to build up correspondence with a controller at a user configurable IP address to utilize the specified port. The switch starts a standard TLS or TCP association with the controller when it knows its IP address. In this manner, the switch must recognize approaching movement as neighborhood before checking it against the flow tables because the Openflow channel does not travel through its pipeline. While, pipeline handling directions enable packets to be sent to subsequent tables for further handling and enable cluster data (metadata) to be conveyed between tables. With various controllers or single controller, the switch may build up connections.

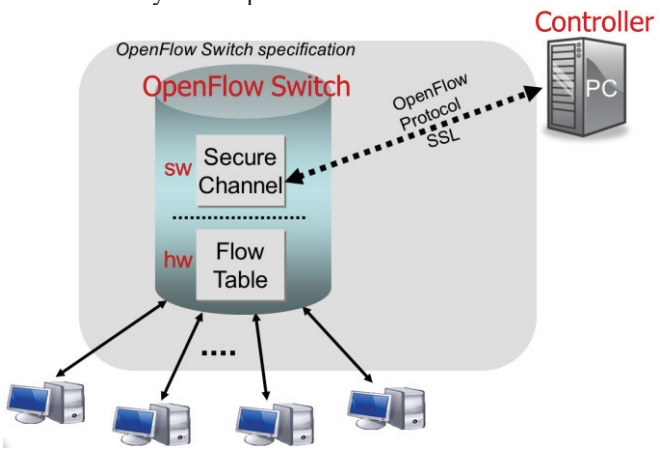

Fig. 3. An Openflow platform [2]

B. Openflow Switches

In this layer, data processing devices exist. Data packets are handled based on the actions installed from the network controller in every individual device. As shown in figure 4, an Openflow rule consists of four main parts: priority, matching condition, action and related active counters. Priority field is used to define the order of matching of a data packet against the set of installed rules; hence, once a higher priority rule is matched other lower priority rules are ignored. Matching condition consists of any combination of several IP/Ethernet headers (such as source IP, destination IP, port numbers, Mac address and VLAN id). The action field defines the action that should be taken upon receiving the packet such as forwarding the packet to an outgoing interface, continue processing in the flow table pipelines, or to modify some of the header fields before forwarding or to drop the packet. Finally, the counters field defines the associated counters to this rule such as the number of matching packets to this rule, its lifetime, and its last used timestamp and so on. Those values are sent to the controller to get some statistics about the data plane.

Priority	Matching Condition	Action	Counter

Fig. 4. Structure of an Openflow rule

On arrival of a new packet from the network, the openflow switch process this frame as shown in figure 5. If a matching rule is found, then the frame is forwarded to the next flow table in the pipeline for further processing. A list of actions is updated. At the end of the pipeline, all the cations in the cation list are executed and hence the frame is either dropped, forwarded to an output port or sent to the controller. In many openflow switches implementation, only one flow table is used instead of a pipeline. IN case of no matching rule, the frame is sent to the controller with an appropriate state code and some statistics.

All rules in the flow table(s) are dynamically generated and deleted. In most cases deletion is performed on a periodical basis where a timeout period is defined and if a rule exceeds this value without being used, it should be deleted. Figure 6 shows the operation of the Openflow controller and switch to process incoming data frames and manage the flow table(s) rule.

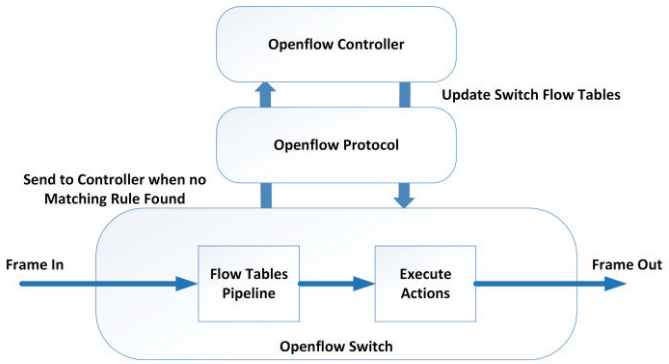

Fig. 5. Openflow Switch flow diagram

III. DENIAL OF SERVICE ATTACKS

A Denial of Service (DoS) attacker may attempt to perform a DoS assault to the controller or use different intends to bring about the controller to come up short. For instance, assailants may attempt a few types of asset utilization assaults on the controller to stall it, cause it to react to a great degree gradually to approaching parcels, and make it moderate to send messages out. Shin and Gu [13] exhibited a doable and viable DoS assault to SDN systems, which contains two stages. To unique mark whether a given system utilizes SDN/OF switches. The conspicuous verification of a SDN switch relies on the recognition that the response times for receiving the packet, may be particular since the stream setup time can be incorporated the instance of new flow. To create made stream demands from the information plane to the control plane. This kind of assault is a noteworthy risk for Openflow based SDN, as it can impair the entire system, for example, corrupting system execution, dropping packets. Likewise, as Openflow controller is a key segment for system control and administration, it has turned into a bottleneck and is incredibly debilitated by such assaults [14].

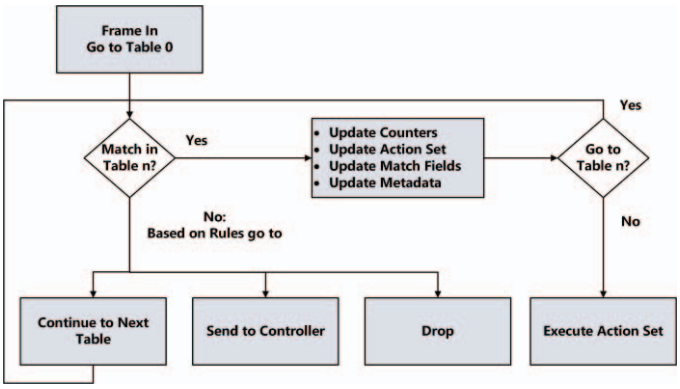

Fig. 6. Openflow Switch and Controller Proceeing

Dependable encryption can't secure SDN from being assaulted, however improved trust components and access approaches can moderate such assaults. At the switch level, it is alluring to have an ability to re-assess stream table principles, guaranteeing administrations to honest to goodness has and denying unapproved access. Current Openflow switches have not actualized such knowledge while additional overhead is the significant concern.

Overall, an attractive security system ought to guarantee both the controller and the switches can rapidly recuperate from huge volume movement assaults. Legitimate checking and reaction systems are relied upon to be dynamic when identifying DoS assaults. Case in point, Shirali-Shahreza and Ganjali [15] proposed FleXam, an adaptable testing expansion for Openflow that empowers the controller to get to bundle level data, which can distinguish DoS assaults. Additionally, stream data is helpful in identifying most DoS assaults, through breaking down activity [16]. For instance, stream headers can be utilized to distinguish DoS assaults if there is an unbalance amongst approaching and active movement.

DDoS attackers can be propelled to surge the connection between the controller and the system in order to keep the controller from handling streams. To avert such assault, the controller ought to be put inside the border, and streams to the controller should be cut on oversaw interfaces on the off chance that they are not from Openflow gadgets. Along these lines, the controller can be secured from flooding assault.

IV. DENIAL OF SERVICE ATTACK MODEL

In [17] and [18], the authors introduced an attack targeting the Openflow switch flow tables. The attack fills up the forwarding rules memory. An Openflow switch usually has a fixed size memory to store its forwarding rule list. The memory size is limited to N rules where N varies according to the switch brand. For example, in HP 5406zl switch N=1500 [13], while in CpQD's Openflow 1.3 Software Switch [19] N=4096.

If the flow table memory of the Openflow switch is full and the controller instructs the switch to install a new flow rule, the switch detects that its flow table is full. As the switch cannot install this rule, it sends an OFPT_ERROR message to the

controller with error code OFPFMFC_TABLE_FULL. It then drops this packet. The switch cannot forward buffered packets until there is space in the flow table to install new flow rules.

A. Attack Environment

Figure 7 shows the attack environment. A simple network with one Openflow switch (S1), one Controller (C0) and two hosts (h1 and h2). Host-1 (h1) will act as the attacker targeting the switch (S1).

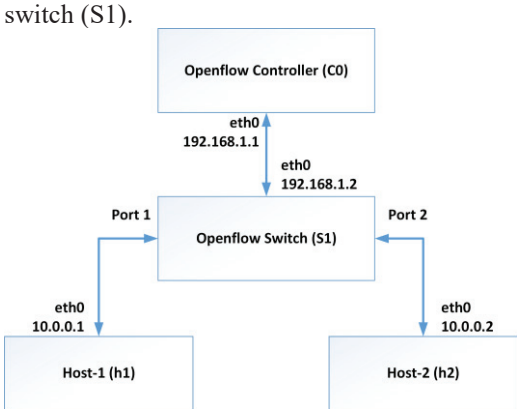

Fig. 7. Emulation Topology

The attacker at (h1) attacks the flow table of (S1) by sending a stream of UDP packets with the destination address of (h2). This means the attacker knows at least one host IP address in the network. Those UDP packets are constructed such that each arriving packet will force the switch (S1) to forward it to the controller (C0) making the controller (C0) to install a new Rule in the switch (S1). The main exploit here is to change the source port number to a new value (sequence) in each sent packet. The traffic is generated using an hping. A total of 50000 UDP packets (with 512 bytes of payload) at a rate of about 1000 packets per second are generated ant sent as described in [17].
Refer to figure 4, the Matching Condition part of the rule is tuple of values extracted from the packet header. Typically, this tuple contains the source MAC address, destination MAC address, source IP address, destination IP address, protocol, source port and destination port. Note that the MAC and port values are those used in classical switches for their ARP table. In addition, wild cards like (*) are used in the flow rule to indicate a field needn't be matched.
Figure 8 shows an example flow table.

B. Attack Results

The results in [17] show that the flow table of (S1) will be flooded by rules, making new arrival packets to be dropped and the switch (S1) effectively out of services.

Pri.	Matching Condition							Act.	Cnt.
	S-MAC	D-MAC	S-IP	D-IP	Prot.	S-Port	D-Port		
1000	*	*	192.168.*	*	*	*	2	For	10
500	*	2d:12:*	*	*	*	2	*	Drop	25

Fig. 8. Flow table entries example

One factor affecting the switch (S1) behavior is the rules time-out preset-value. Openflow switches deletes any unused rule if it is not used for a specific time-out value. If the time-out is small enough, unused rules will be deleted more rapid lowering the effect of the attack, with the side-effect of huge traffic between the controller and switch(s) to install the same rules repeatedly. On the other hand, a larger time-out reduces the controller-switch flow but makes the switch more vulnerable to flow table floods.
Packet drop rate is measured as an indicator for the successfulness of the attack. As expected, with larger idle timeout values, previously installed flow rules remain in the switch's flow table longer. This causes the flow table to be filled up and thereby prevents new flows from being created [17].

V. Proposed DoS Attack Mitigation Model

The idea of Sandboxing is utilized to protect he Openflow switch against the DoS attack of the same type described in section 4. The following subsections describes and analyzes the proposed model.

A. Openflow Sandbox

As shown in figure 9, the same network utilized in the previously described attack is used again. It is the left part of the figure marked as "Operational Environment". Another network is added and marked as "Sandbox Environment".

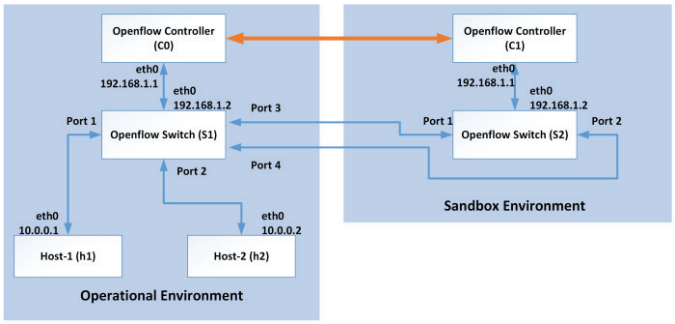

Fig. 9. Switch Sandbox model

The two networks are connected together on 2 levels.
Switch (S1), the target of the attack, has now 2 more ports marked 3 and 4. Those ports are used to connect the switch to the sandbox switch (S2) to its ports 1 and 2.
Also the two controllers (C0) and (C1) have their own protected communication link using SSL and public key certificates.

B. Initial Configuration of flow tables

The operational switch (S1) is initialized with 4 low priority rules that directs all traffic from port 1 to port 3 and vice versa, and the same for port 2 and 4 as shown below.
The rules tuple, refer to figure 8, will look like:

- 1, *, *, *, *, * , 1, *, forward to port 3, ---
- 99999, *, *, *, *, * , 3, *, forward to port 1, ---
- 1, *, *, *, *, * , 2, *, forward to port 4, ---
- 99999, *, *, *, *, * , 4, *, forward to port 2, ---

The rules will cause any traffic coming on port 1 or 2 without a matching rule, this flow includes the attack, to be forwarded to witch (S2). The priority level of those rules (first and third) are set to (1) minimum so they are executed only if no other rule is matched.
Also the second ad fourth rule are given a very high priority level (99999) so they are executed before any other rules.

Switch (S2) tables are left to be configured by its controller (C1).
The time-out value of switch (S2) is set to a small value, so rules in its flow table are purged rapidly.

C. Communication between the controller

The two controllers (C0) and (C1) are connected with an out of band secure channel, they communicate dynamically during the operation as flows:
From (C0) to (C1): Periodically (C0) reads the flow table of (S1) and send any rules with port 1 or port 2 as final destination to (C1), where (C1) orders (S2) to implement the exact same rules. This action will make (S2) to look exactly as if it is a 2-ports switch connected to the hosts (h1) and (h2) of the operational network.
From (C1) to (C0): After each N time-outs intervals of (S2), the controller (C1) will read the rules in (S2) flow table, deletes those rules from (S2) flow tables and finally send them to (C0). Immediately, (C0) will implement these rules on (S1) flow table. As those rules are not deleted after N time-outs, they can be classified as safe rules.
The parameter N is adjusted for performance, a value of 0 makes no Sandbox. The smaller N saves some time as lower volumes of traffic will be sent to the sandbox, while a larger N offers more protection.

D. Analysis of the proposed model

The proposal will make the Sandboxing network to get all the new traffic from the operational network if the packets are new and do not have a matching rule, creates prober rules for those new packets on the sandbox switch (S2) and sends the traffic back to the operational network for normal delivery. Traffic is not interrupted but takes some extra time.
Only safe rules are copied back from the Sandbox network to the operational network successfully protecting (S1) from being flooded with DoS attack rules.
The cost of the model is:

1. Double sized switch with extra ports the same number of working ports connected to hosts.
2. Extra time is needed when forwarding the traffic to the sandbox switch and back

E. Results

The model is simulated on Mininet simulator [20] is used to test the proposal with the same attack described in section 4.
The total number of packet dropped is almost 0 meaning that the attack failed.
One final comment is that it is possible to use the Mininet environment with software switches for implementing the Sandbox with real SDNs. A sufficient resourced computer will be able to mitigate DoS attacks with minimal cost.

VI. Conclusion and Future Work

This paper introduced some basic description of SDNs and Openflow protocol and components. The possibility of attacking Openflow switches exploiting the openness of Openflow causing the flow tables of the switch to be flooded is described.
A novel Sandbox protection model was introduced and proven to mitigate the DoS attack presented.
The Mininet environment was used for simulation and also was proposed to implement the actual sandbox for minimizing the cost.
Only the rules time-out was used to judge the validity of the rule and only one attack model is addressed.
In the future, it is recommended to consider more attacks especially sophisticated ones and also to implement more powerful analysis tools on the sandbox controller so as to detect more attacks. This may include information from all over the network devices including security systems and operating systems logs, and may also consider the payload of the packets.

References

[1] M. Mousa, M. Sobh and A. Bahaa-Eldin, "Software Defined Networking Concepts and Challenges," in The 11th IEEE International Conference on Computer Engineering and Systems, IEEE, Cairo, Egypt, 2016.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008.

[3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Communications Surveys & Tutorials , vol. 16, no. 3, pp. 1617-163, 2014.

[4] A. E. Taha, I. Abdel Ghaffar, A. M. Bahaa-Eldin and H. M. Mahdi, "Agent based correlation model for intrusion detection alerts," in 2010 IEEE International Conference on Intelligence and Security Informatics (ISI), IEEE, 2010.

[5] A. M. Bahaa-Eldin, "Time series analysis based models for network abnormal traffic detection," in 2011 International Conference on Computer Engineering and Systems (ICCES), IEEE, 2011.

[6] A. M. Bahaa-Eldin, "A Bio-inspired Comprehensive Distributed Correlation Approach for Intrusion Detection Alerts and Events," in Bio-inspiring Cyber Security and Cloud Services: Trends and Innovations, Springer Berlin Heidelberg, 2014, pp. 3-38.

[7] H. N. Gabra, A. M. Bahaa-Eldin and H. K. Mohammed, "Data Mining Based Technique for Ids Alert Classification," International Journal of Electronic Commerce Studies, vol. 6, no. 1, pp. 119--126, 2015.

[8] I. T. Abdel-Halim, H. M. A. Fahmy and A. M. Bahaa-Eldin, "Agent-based trusted on-demand routing protocol for mobile ad-hoc networks," Wireless Networks, vol. 21, no. 2, pp. 467-483, 2015.

[9] A. M. Bahaa-Eldin, "TARA: Trusted Ant Colony Multi Agent Based Routing Algorithm for Mobile Ad-Hoc Networks," in Bio-inspiring Cyber Security and Cloud Services: Trends and Innovations, Springer Berlin Heidelberg, 2014, pp. 151-184.

[10] A. O. A. El-Mal, M. A. Sobh and A. M. Bahaa-Eldin, "Hard-Detours: A new technique for dynamic code analysis," in 2013 IEEE EUROCON, IEEE, 2013.

[11] O. Abdelrahim, A. Taha and A. M. Bahaa-Eldin, "A framework for virtual machine admission control in cloud environment," in The 11th IEEE International Conference on Computer Engineering and Systems, IEEE, 2016.

[12] D. Plummer, "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48. bit Ethernet address for transmission on Ethernet hardware," IETF, Internet Standard RFC:826, 1982.

[13] S. Shin and G. Gu, "Attacking Software-defined Networks: A First Feasibility Study," in Proceedings of the Second ACM SIGCOMM

Workshop on Hot Topics in Software Defined Networking, ser. HotSDN'13, New York, NY, USA: ACM, 2013, pp. 165–166, 2013.

[14] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, Guofei Gu, "AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks," 4–8 November 2013.

[15] Shirali-Shahreza, Sajad and Y. Ganjali, "FleXam: flexible sampling extension for monitoring and security applications in openflow," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 167-168. ACM, 2013.

[16] Haopei Wang, Lei Xu, Guofei Gu, "FloodGuard: A DoS Attack Prevention Extension in".Software-Defined Networks.

[17] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1322-1326. IEEE, 2015.

[18] R. Kl¨oti, V. Kotronis and P. Smith, "OpenFlow: A security analysis," in 2013 21st IEEE International Conference on Network Protocols (ICNP), PP 1-6, 2013.

[19] "CpQD OpenFlow 1.3 Software Switch," http://cpqd.github.io/ofsoftswitch13/, Accessed on December 1st, 2016.

[20] M. Team, "Mininet: An instant virtual network on your laptop (or other PC)," http://mininet.org/, accessed on October 2nd, 2016, 2012.