SECURITY FUNCTION VIRTUALIZATION IN SOFTWARE DEFINED INFRASTRUCTURE

Pouya Yasrebi, Sina Monfared, Hadi Bannazadeh, Alberto Leon-Garcia

Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada M5S 3G4
Emails: {pouya.yasrebi,sina.monfared,hadi.bannazadeh,alberto.leongarcia }@utoronto.ca

ABSTRACT

In this paper we present an approach to implement security as a Virtualized Network Function (VNF) that is implemented within a Software-Defined Infrastructure (SDI). We present a scalable, flexible, and seamless design for a Deep Packet Inspection (DPI) system for network intrusion detection and prevention. We discuss how our design introduces significant reductions in both capital and operational expenses (CAPEX and OPEX). As proof of concept, we describe an implementation for a modular security solution that uses the SAVI SDI testbed to first detect and then block an attack or to re-direct it to a honey-pot for further analysis. We discuss our testing methodology and provide measurement results for the test cases where an application faces various security attacks.

I. INTRODUCTION

Conventional network security systems consist of proprietary hardware boxes, usually including Application Specific Integrated Circuits (ASIC) to perform Deep Packet Inspection (DPI). These devices are able to detect malicious traffic, and prevent it from passing by blocking it. However, as hardware implementations, they provide limited flexibility to be customized based on specific customer needs. At the same time, these systems are costly and not easily deployed in a cloud. Their installation involves all the tasks associated with physical installation of a hardware system. In most cases, the operation of such security devices may not be downsized when traffic drops and consequently the equipment goes underutilized. In addition capacity upgrades require an additional installation with the associated disruption in network operations.

In contrast to hardware systems, a software-based approach based on commodity and/or sharable hardware can provide capex savings as well as scalability, flexibility, and customizability. In this paper, we present an approach to provide security services in software that can be run on Virtual Machines (VMs). The approach takes advantage of Software Defined Infrastructure (SDI) to enhance the security service provided by the network.

II. SAVI AND SDI

Smart Applications on Virtual Infrastructure (SAVI) is a project that is an outcome of industry and academia partnership in Canada. The core idea of SAVI is to combine Cloud-computing and Software Defined networking that is managed by an integrated SDI Resource Management System (RMS). SAVI [1] provides Infrastructure as a Service (Iaas) as well as Network as a Service (NaaS), since it provides all the services provided in each of these.

Network Function Virtualization (NFV) is a network architecture model for virtualizing network services. Network functions that were previously executed in hardware are now performed using commodity computing resources.

Many cloud computing management systems, such as OpenStack, do not provide a dedicated security system. Examples for SDN security projects include FortNOX[2], Cloud Police[3], Flowvisor[4], and FRESCO[5]. In this paper we consider providing security using NFV on SDI and hence leveraging cloud computing and SDN.

II-A. Central SDI manager

In this section we review SAVI SDI RMS and its components. The aim is to introduce the SAVI SDI to provide the reader with an understanding of the functionalities used in the security VNF.

As in [6] and [1] the SDI Manager, code-named Janus consists of a number of modules and a module manager. These include a scheduling module, networking control module and fault tolerant module. The module manager will overlook all modules. Janus, which encompasses the modules and module manager, is accountable for management and control tasks. We will use the networking control module to implement security.

A topology manager, code-named Whale, stores network node and link status information in the form of graph. This graph contains how nodes are connected, what routes or flows exists between each nodes, and other general measurements. Whale is directly connected to an Openflow controller to obtain and update physical and virtual network properties. It also acquires physical and virtual node properties. The cloud controller in SAVI is based on OpenStack. Whale directly transmits required information to Janus for control purposes[7].

In the next sections we discuss network attacks and how our design can employ Janus to act upon them.

III. NETWORK ATTACKS

Rapid increase of internet speed and computational power has enabled attackers to launch Denial of Service (DOS) attacks. The attackers may use personal computers to initiate

an attack to bring down target servers. These servers will not be able to handle their incoming traffic, therefore they will fail to provide service to normal users in the worst case.
In a Distributed Denial Of Service (DDoS) Attack. A bot-herder (malware producer) will slave a set of geographically distributed computers to send fake requests to target servers[8]. Since these requests are not directly coming from the bot-herder, it is usually very difficult to detect the bot-herder in DDoS attacks.

IV. SAVI SDI TO IMPLEMENT NFV SECURITY

We propose SAVI SDI for network and cloud controlling purposes to address and resolve security. Our solution is to be scalable, dynamic, and secure for traffic management.
Since SAVI SDI is connected to Whale, it is aware of every network route that exists on the OpenFlow switches. This allows SDI to centrally control and re-route or manage security hazards depending on the type of security issue. Central control of SDI and its access to Whale permits it to be aware of the system as a whole.
There are a number of features in SAVI SDI that makes it a powerful platform in terms of security implementations: dynamical placement of VMs, allocation of network resources via cloud computing, smart functionality based module associator, and smart traffic and routing controller.
Since the SDI Manager has control over both network resources, cloud computing resources, as well as Whale, it is capable of being programmed to optimally place security modules according to network topology. This placement can be enforced by Service Providers or Internal networks according to their target in terms of quality of service.
SDI allows dynamic allocation of network resources for security modules. Such modules can therefore vary in terms of processing power and monitoring functionality purposes. This feature becomes handy as incoming traffic grows due to higher demand of a server or as a Service Provider expands its security policy.
We will show that SAVI/SDI is capable of implementing layers of security modules for different network attacks. Therefore it is capable of blocking attacks based on their severity in different parts of the network. This feature takes advantage of SDI Manager accessing Whale as well as network manager. Whale may identify the critical points for layer implementations and SDI will take the order and put software defined functions in place.
SAVI/SDI has access to SDN controllers and shall act upon a malicious traffic either by blocking it or forwarding to a different server for study purposes. These ”fake” servers are generally called Honey Pots that are used to analyze security attacks SDI uses Whale's knowledge of the network to determine a proper network point for applying preventing measure.
It is usual for an attacker to forge an IP address of a user, meaning an attacker sends packets with a different IP than his. This is usually done since an attacker does not want his own IP to become evident. Therefore an attacker floods a server with random different IP addresses. This kind of attack is inherently prohibited by Janus. Janus prevents this by using security measures on its OpenFlow switches. Hence, it sets forwarding-rules according to connected internal IP addresses that are issued and authenticated via the SDI itself. Hence any divergence between source IP address and the IP rules that are in place will cause the packets to be dropped. Moreover, flooding with different source IP addresses will be rejected in the closest OpenFlow switch. This synchronicity will prevent attackers to forge IP addresses through SAVI SDI.
As previously mentioned, some attackers use malwares to initiate attacks from other insecure devices generally known as bots. These compromised machines behave as puppets for the attacker and initiate an attack. One type of security attack is a DDOS attack. This attack will be detected via Intrusion Detection Systems (IDS). IDS is to identify malicious system activities and violation of system policies. Usually a detection is followed by some way of reporting the intrusion to system administrators. Some Network Intrusion detection systems have the capability of preventing intrusion themselves.
One way to detect an intrusion is to use deep packet inspection. In the deep packet inspection method, an IDS will go over packets and compare them with previous well-known network traffic patterns. Reports contain any digression from these patterns. Two possible methods for IDS are statistical anomaly-based IDS and signature-based IDS. In the former, either a previously written rules or a set of learning algorithms try to identify the intrusion with a high probability. In the latter, the packet will be inspected and a set of signatures will be compared with a signature database to assure the legitimacy of traffic. Both methods have pros and cons that are out of scope of this paper. In the following we will describe how the IDS is implemented in the SAVI Testbed.
As mentioned SDI is capable of design and allocation of dynamic security modules. Lets consider one security module (shown in Fig.1). The SDI Security Module manager requires several information sources to assist its decision making: SDI Monitoring and Measurement, Whale and SDI Network Control Module.
Consider a security module that detects certain types of DOS attack. For the victim, we use a webserver that contains a webpage that is to be a point of interest for attackers. Attackers will try to flood the webserver using their bandwidth and high processing power. A typical result would be exhaustion of webserver's resources to respond back to the requests sent by the attackers. Consequently the webserver will be unable to provide service to normal users. The proposed security module employs a statistical anomaly based IDS as a deep packet inspector service to study and monitor the behavior of packets moving toward the targeted webserver. Right after the attack occurs, the deep packet inspector will identify the attack and raises a flag indicating that an attack has occurred. There are two methods for using the IDS. One is the case that the IDS only monitors the data by tapping from the line and the other could be placing the IDS on the line as a serial module. We tested both of these scenarios. In comparison, the former had a near line speed performance since the computation is completely separate from the forwarding switch. Thus, the performance of switch does not degrade

in the tapping case. In the latter, IDS system may be further employed to perform the job of a Network Intrusion Prevention System as well. The issue is that the inspection part is CPU intensive. Since the CPU allocated for the IDS gets occupied by packet inspection, the performance of forwarding section of the VM gets affected. Therefore if the DPI was used as an inline mode, the line speed becomes very limited and hence a bottleneck.
It is true that all of these depend on the resources of the VM that the security module is operating on and one can argue that by increasing the computation power on the inline mode, the forwarding will not be affected. In this paper we will focus on tapping which seems better suited version for this task. Since the number of security rules that needs to be processed might increase over time, the tapping solution is more scalable. Using a combination of IDS in the tap mode and SDIs security module manager as the prevention system will be a good match to secure the web server.
The deep packet inspector also has the capability of identifying the attackers (in this case compromised users). Right after the attack was detected and the bots were flagged, the attack will be sent to SDI Security module manager as a report. This report allows Janus take a global decision on the network. In here, SDI Security manager contacts Whale to acquire the location of the reported IP addresses. Topology refers to its graph based database to locate IP addresses. At this point Janus makes a decision based on its security module manager to either study the attack or to block the attack.

IV-A. Possible Intrusion Prevention Scenarios

There are three possible Intrusion Prevention scenarios: Blocking the attack in the back end, blocking the attack in the front end, and study the attack in a different server (Fig.1).
In the case of an attack, a first short term solution is to

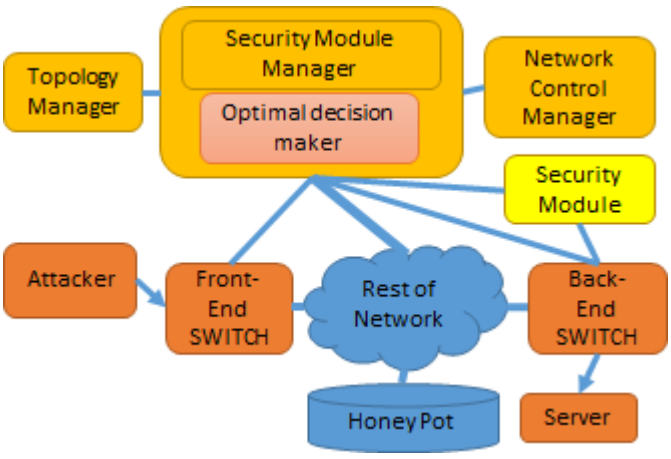

Fig. 1. Three possible Attack Prevention methods on SAVI/SDI

block packets associated with intrusion from reaching to original server. This action plan is initiated by sending a message to modify a flow in a switch near the IDS security module. Here the packets are blocked near the victim web-server.
If SDI determines that the attack is not that critical, SAVI/SDI will create a honey-pot (a copy of the original server) and direct the attack to the honey pot. The location of honey pot will be determined via a decision from Whale as well as the SDI security module manager. This decision aims to minimize the honeypot cost in the system.
In case that the SDI does not find the attack interesting enough to explore with Honey Pot, it will contact Whale to map and locate the compromised IP address and its nearest Open-Flow switch. Then SDI will directly block the IP address at its closest openflow switch (front end). At the end SDI will try to remove malware from the specified device.
In the case that the security module manager decides to study the attack, SDI will determine a suitable point in the network to instantiate a Honey Pot server.

V. EVALUATION

In this section, we discuss the testing and verification of the proposed methods. Parameters of interest include attack detection and response time, resource utilization and the transitional time from attack detection to mitigation.
Here we have initiated a DDOS attack from a number of VMs to a web-page. The number of attackers for our experiment has been set to 4. At this moment, SNORT [9] which is an open software has been employed as an IDS. Snort is a free open source software for network security. It implements intrusion detection and prevention by means of Deep Packet Inspection (DPI).
All of the experiments have been developed and evaluated on SAVI Testbed, which as mentioned, is an implementation of SDI. The IDS sits on a VM that has been created via SAVI SDI. Here, the system was examined under a set of attack prevention methods and the corresponding system performance was extracted. In the case that there are no security modules in place, as the number of attackers increase, the webpage loading time increases as well. To fully study the attack, the resources such as CPU usage, memory usage, and bandwidth utilization on a VM were monitored. It was quite surprising to observe that with the number of attackers that we studied, the only depleted resource was the incoming/outgoing bandwidth of the web-server. The CPU usage rose only by nearly 2% and memory had a similar increase, while the bandwidth of the traffic had a huge boost as the flood of traffic came in.
To assure that the system is properly responding to attack scenarios, two mitigation have been tested: Front end blockage, and Honey-pot. In Fig.2 the bandwidth resources on the web-server has been monitored. As soon as an attack is initiated, a spike in the bandwidth utilization is visible. This attack is followed by an automatic detection from IDS and a report to SDI security manager to block the attackers. Hence the bandwidth utilization will go back to normal after the attack. To do a sanity check on Honey Pot system, the

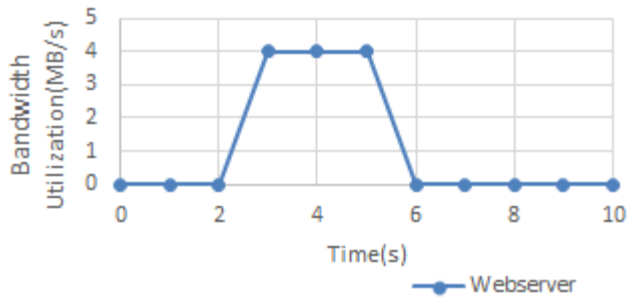

Fig. 2. Web-server bandwidth utilization under attack and prevention

bandwidth utilization has been looked at for the real web-server as well as the Honey Pot server for a same period of time. In Fig.3 we can observe that the traffic has been shifted from the Real Server to the Honey Pot after the attack is detected. The next important factor for our purpose

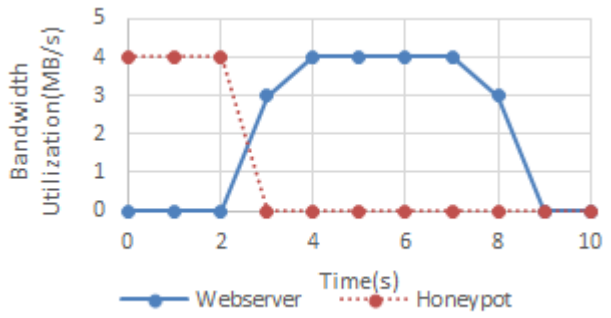

Fig. 3. Attack has been shifted from Webserver to Honeypot

is a response time from the initiation of attack to time of their blockage. Our purpose is to demonstrate the reasonable performance of the system. This end-to-end number varies according to the number of rules that are in place for the IDS module. But for our test purposes, the detection and re-routing numbers are demonstrated in Table I.

Table I. Timing for detection and different mitigations.

Time to detect	on Average 2.5 seconds
Honey Pot transfer	$<$ 4 seconds
Block the IP	$<$ 2 seconds
Detection + Block	$<$ 4 seconds
Detection +Honey Pot	$<$ 7 seconds

VI. CONCLUSION

In this paper we introduced an architecture for providing security services to application using Virtualized Security Functions in a Software Defined Structure. We described how SDI can facilitate scalable and efficient deployment of security functions and how it can be utilized by security functions to take action when a security threat is detected. We demonstrated the effectiveness of proposed security solution by developing a fully working Deep Packet Inspection system on the SAVI Testbed which a prototype of a Sofwtare Defined Infrastructure and has been deployed in seven Canadian universities. Providing measurements, we argued that our architecture is not only well suited for short term detection and quick reaction to security threats, but also it utilizes SDI features to block attackers from entering the network. For future work, we will look at ways to further utilize SDI features in demonstrating a large scale security system that could provide security for all applications running on a Software Defined Infrastructure.

VII. REFERENCES

[1] Joon-Myung Kang, H. Bannazadeh, and A Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 664–667.

[2] Porras, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, New York, NY, USA, 2012, HotSDN '12, pp. 121–126, ACM.

[3] Popa, "Cloudpolice: Taking access control out of the network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, New York, NY, USA, 2010, Hotnets-IX, pp. 7:1–7:6, ACM.

[4] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[5] G. Piccinelli, C. Zirpins, and W. Lamersdorf, "The fresco framework: an overview," in *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, Jan 2003, pp. 120–124.

[6] T. Lin, Joon-Myung Kang, H. Bannazadeh, and A Leon-Garcia, "Enabling sdn applications on software-defined infrastructure," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–7.

[7] Hadi Bannazadeh Joon-Myung Kang, Thomas Lin and Alberto Leon-Garcia, "Software-defined infrastructure and the savi testbed," *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2014)*, 2014.

[8] B. Al-Duwairi and G. Manimaran, "Just-google: A search engine-based defense against botnet-based ddos attacks," in *Communications, 2009. ICC '09. IEEE International Conference on*, June 2009, pp. 1–5.

[9] "Snort," http://www.snort.org, Accessed: 2014-09-30.