SDSecurity: A Software Defined Security Experimental Framework

Ala' Darabseh[1], Mahmoud Al-Ayyoub[1], Yaser Jararweh[1], Elhadj Benkhelifa[2], Mladen Vouk[3], and Andy Rindos[4],
[1] Jordan University of Science and Technology, Irbid, Jordan
[2]Mobile Fusion Applied Research Centre, Staffordshire University, Stafford. UK
[3] North Carolina State University, Raleigh, North Carolina, USA.
[4] IBM Corporation, Research Triangle Park, North Carolina, USA.

Abstract—**The emerging Software Defined Systems (SDSys) is a recent paradigm, which has been introduced to reduce the overhead in the control and management operations of complex computing systems. The main concept behind this technology is around isolating the data plane from the control plane. Traditional security mechanisms are facing more challenges in providing sufficient levels of protection and efficiency. SDSys for security has been proposed to address these challenges. Software Defined Security (SDSec) provides a flexible and centralized security solution by abstracting the security mechanisms from the hardware layer to a software layer. In this paper we present a novel experimental framework to provide a novel virtualized testbed environment for SDSec systems. This work builds on the Mininet simulator, where its core components, the host, switch and the controller, are customized to build the proposed experimental simulation framework for SDSec. To the best of the authors' knowledge, this is the first experimental framework and simulator for SDSec solutions. The developed simulator, will not only support the development and testing of SDSecurity solutions, it will also serve as an experimentation tool for researchers and for benchmarking purposes. The developed simulator could also be used as an educational tool to train students and novice researchers.**

I. INTRODUCTION

Virtualization has been widely adopted across different sectors following the rapidly increasing adoption of Cloud Computing technology and services due to its advantages over traditional computing provision. Software Defined Systems (SDSys) is a very recent paradigm, proposed to address control and management challenges known in traditional platforms by hiding their complexities from the end users. This is achieved by isolating the data plane from the control plane. SDSys technology has grown very rapidly to encompasses a number of disciplines such as Networking (SDN), Data Centres (SDD), Storage (SDStor), etc. However, SDSys, as a technology, is still not widely adopted due to some issues causing some concerns among IT enterprise managers which hinders its spread. One of the major issues is network security.

Traditional security mechanisms are not suitable for the new SDSys architectures, which therefore, require new security mechanisms. The SDNCentral website [1] discussed the main points to be covered when building and designing a security solution for SDN such as how the control layer must be kept protected and secure to guarantee the availability of the controller, and how to build trust between the controller, the applications and the devices.

Software Defined Security (SDSec) is a new technology emerging under the SDSys paradigm. It is an example of a Network Function Virtualization (NFV). The new technology provides a new way to design, deploy and manage security mechanisms by separating the forwarding and processing plane from the security control plane, in a similar way as SDN abstracts the forwarding plane from the control and management plane. Such separation provides a scalable distributed security solution, which virtualizes the security functions but remains manageable as a single logical system [1]. SDSec was proposed as a solution to help secure virtualized environment infrastructures, including virtual network, virtual storage and even virtual servers from different threats whether they are traditional such as intrusion detection and denial of service attacks or specific to virtualized environments such as insider threats [2], [3]. In SDSec the functions of network devices, like intrusion detection, firewalling and others, are extracted from the hardware appliances to a software layer.

It is important to differentiate between the software defined concept and another related concept which is "software deployed." In the former concept, the APIs and software are used to control and manage the resources and devices. On the other hand, the software deployed concept means that the functionality of the service is deployed in a computer hardware object. Using a software to manage and control the resources is not a new concept. The essential difference, which was brought by software definedness, is the ability of the control layer to control all the underlying resources regardless of their vendor variations by physically isolating them from the hardware resources in the data layer [4]. The concept of abstraction in SDSys is similar to the idea of Object Oriented (OO) paradigm, where the implementation is separated from the interface representing the data layer and the control layer, respectively, in SDSys. The reason behind this separation is to simplify the modification process, so that any change in the implementation will not affect the interface and vice versa [4].

Transferring the concepts derived from SDSys into real workable system is not a simple task. Implementing new concepts or solutions directly in operational systems before testing them in simulated environments are considered inefficient, costly and risky; especially, when dealing with security related solutions. Though, apart from SDN, there exist no simulation environments for most SDSys, including SDSec. The Mininet [5] simulator is the most used simulator by SDN research community due to its simplicity and usability.

In this work, we endeavour to extend the features and functionalities of the Mininet simulator to enable the simulation of SDSec solutions. This framework is called SDSecurity and, to the best of the authors' knowledge, it is the first experimental framework and simulator for SDSec solutions. The developed simulator, will not only support the development and testing of SDSec solutions, it will also serve as an experimentation tool for researchers and for benchmarking purposes. The developed simulator could also be used as an educational tool to train students and novice researchers.

The rest of this paper is structured as follows. In section II we explain the idea of SDSec in more details including its architecture and features and discuss some real SDSec systems. After that, a brief introduction about Mininet is given in section III. Whereas, our experimental SDSecurity framework will be explained in section IV. Finally, we conclude this paper and present our future plans in section V.

II. Software Defined Security (SDSec)

Traditional security mechanisms are considered unsuitable to deal with virtualized environments. The design of traditional security devices is unable to protect the components of virtualized environments, due to its dependency on physical network devices, which cannot see the significant security activities inside virtualized environments. The changes brought by virtualization, which range from new virtual network topology and the threats related to the hypervisors to eliminating the roles in virtualized management, demonstrate the need of virtualized security. Such virtualization would reduce the complexity and the cost of security operations. It also facilitates the deployment of security policies that are superior to the classical ones by making them more accurate, seamless and context-aware. In addition, by virtualizing the security, the data center can automate all the security activities such as firewalls configuration.

A. SDSec Architecture

Similar to the SDN architecture, the architecture of SDSec separates the data plane from the control plane. The general view of this architecture is organized into three main layers: the physical layer, the control layer and the application layer.

- The Physical layer: Inside this layer, all the hardware devices are located, which may include database arrays, switches, routers, or any other asset. This layer is also called the data layer and its role is limited to following the policies created by the control layer, which resides on top of this layer.
- The Control layer (Middleware layer): All the control and management operations are abstracted form the devices located in the physical layer and set inside this layer. This layer is considered the brain of any SDSys since it handles all the core control operations.
- The Application layer: It is the only one visible to the user. All applications are implemented inside this layer. Several applications can be created on the top of the control layer to help the users interact with the underling devices and data.

The three layers can communicate with each other by a set of APIs, southbound and northbound. The former is used by the physical layer to interact with the control layer. The most famous example of such APIs is the OpenFlow protocol, which is used in SDN systems. As for the latter, it is used by the user's applications to interact with the control layer. To the best of our knowledge, there is no standard northbound APIs till now.

B. Features of SDSec

There are some features and attributes that distinguish the SDSec approach from traditional security approaches, and eliminates traditional security bottlenecks that prevent the system from expansion and exploitation of virtual resources. The main ones are listed below with a focus on Catbird, a software defined security solution, as a case study [6], [7].

1) Abstraction: As the SDN abstracts the control and management from hardware appliances, the SDSec does the same thing by abstracting the security policies from the hardware layer and run it in an independent software layer. Catbird uses the idea of policy envelops, which cover the hardware assets. In such a way, the deployment of a new policy control is not affected by the assets or VMs location, which simplifies the network.
2) Automation: Creation of a new VM or device in the system and putting it in a specific trust zone is done automatically by Catbird without the need for manual intervention. On the other hand, the process of detecting any violation or vulnerability is done automatically when an event occurs and then the appropriate alerts are fired to solve the problem. Such flexibility does not exist in traditional security approaches, which are working, mostly, manually. The transition from manual to automatic process increases the speed of the system and enhances its efficiency.
3) Elasticity: Since Catbird is considered entirely software-based and unrestricted to hardware, it is easy to scale it up and adapt it to new changes.
4) Concurrency control: In SDSec systems security techniques and controls, like intrusion detection, firewalling, violation monitoring, etc., are working together as a single comprehensive system. Such a grouping improves the system security and accuracy, and, at the same time, reduces the cost. In fact, the abstraction feature of SDSec provides the ability to apply different aspects of security regardless of the underlying language of the assets appliances; thus, providing a higher level of control.
5) Visibility: By virtualizing the security, Catbird increases the ability to discover the problems and abnormal actions and activities.
6) Portability: The independent property of SDSec facilitates the deployment process of Catbird even if the devices move from one location to another.

C. Existing SDSec solutions

The idea behind the SDSec concept appeared at the Cloud Security Alliance (CSA)[1] as they sought to find a new ap-

[1]https://cloudsecurityalliance.org/

proach for security with lower costs [8]. To transfer their vision into reality they launched the Software Defined Perimeter (SDP) project as new security architecture in order to keep secure systems against network attacks [9]. SDP was designed to complement SDN in order to reduce the attacks on the network applications by disconnecting them until the users and devices are authenticated.

Other examples include Catbird [6], vShield [10], [11], OneControl [12], vArmour [13], etc. Catbird is a full software-based security solution for hybrid and private cloud. It protects the virtual resources automatically, monitors the system security to verify real time security control and enforces the necessary alerts when needed. The latest release of Catbird, which is formally called vSecurity, is the first multi firewalls and hypervisors integration solution. Catbird is established to support the hypervisors of Microsoft and VMware as well as VMware vCloud networking and security applications firewall and Cisco Virtual Security Gateway (VSG).

vShield, part of the vCloud suite, covers the bottlenecks of the physical security approach, and provides a single comprehensive virtual security framework. It improves security management and reduces the complexity associated with virtualized security solutions especially for the companies that need to move to virtualized and cloud environments. Also, it allows customers to build various policy-based groups and establish logical boundaries between them.

Trying to eliminate the need of manual reconfiguration and response actions when an event or any change occurs in the network, NetCitadel introduced OneControl [12]. It provides users with an abstracted user interface. This interface facilitates the network management operations to configure the hardware devices as network routers, switch, or network firewall. OneControl works like a central point [14] controller, which receives any change or event in the network and decides if this event requires any policy change. The overall system workflow is shown in Figure 1. As shown in this figure, scripts or daemons are used by OneControl to determine if the new event requires policy change. The new policy rules are pushed to specific devices by the controller to take effect and generate a response. It allows the system managers to programmatically detect and respond to various events by a set of REST APIs.

Another security company launched its own SDSec solution, called vArmour, for SDN-based and cloud data center systems to fully exploit the benefits of virtualization environments [13]. vArmour addresses the scalability, flexibility, and cost issues facing traditional security techniques in virtualization environments. It provides a dynamic and secure protection for various organizations assets that work with a new paradigms like cloud computing, mobile applications and virtualization systems. vArmour protects distributed data, which are located across several servers in an efficient manner to allow the enterprises to adapt with the new business changes in real-time.

III. Mininet: A Network Emulator

Mininet is an OpenFlow-based SDN simulator which gives researchers an efficient way to evaluate their SDN frameworks by studying their behaviors and measuring their performances. Mininet is an open source simulator written in the python

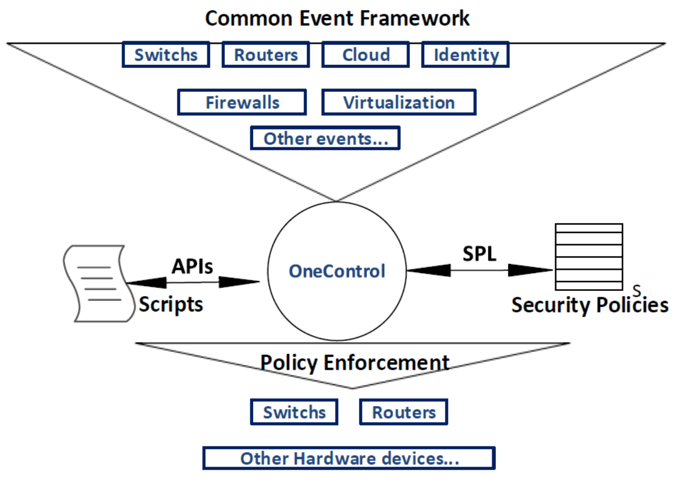

Fig. 1. The workflow and policy enforcement in OneControl SDSec solution

programming language and is built over the Ubuntu Linux distribution. The elements of Mininet are organized into three main components: the host, which sends and receives the packets, the switch, which stores all the required rules to forward the packets to its destinations, and a central controller, which handles the functionality of control and management operations in the network.

Mininet supports different types of virtualized hosts, switches and controllers. Furthermore, it provides two essential tests, ping and iperf, to check the reachability and the network bandwidth, respectively. The minimum topology supported by Mininet consists of two hosts (h1 and h2), one switch (s1) and a central controller (c0). However, the users are given the flexibility to extend this topology and build their own customized topologies to test the performance of their algorithms/solutions. One of the drawbacks of Mininet is its inability to handle large scale networks [15]. MaxiNet is an extension of Mininet to emulate a large scale environment by building a distributed emulation environment [15].

The growing interest in Software Defined Systems (SDSys) in general and especially SDSec is the motivation behind our proposal to build an experimental framework for the SDSec system with the ambition to help advance SDSec research similar to what Mininet did for SDN. Moreover, the simplicity and flexibility of the Mininet help us to embody this ambition and transform it into reality by customizing the existing elements in Mininet. This gives researchers the chance to test their new systems before transferring them into real systems.

IV. SDSecurity: A Software Defined Security Emulator

In this section we explain in details our Software Defined Security experimental framework, SDSecurity. The proposed SDSecurity experimental platform is build upon Mininet. Starting with the environment and characteristics of Mininet, we build SDSecurity by customizing and extending the elements of Mininet to facilitate building a virtualized environment to emulate the different forms of SDSec policies and test their performance under different scenarios.

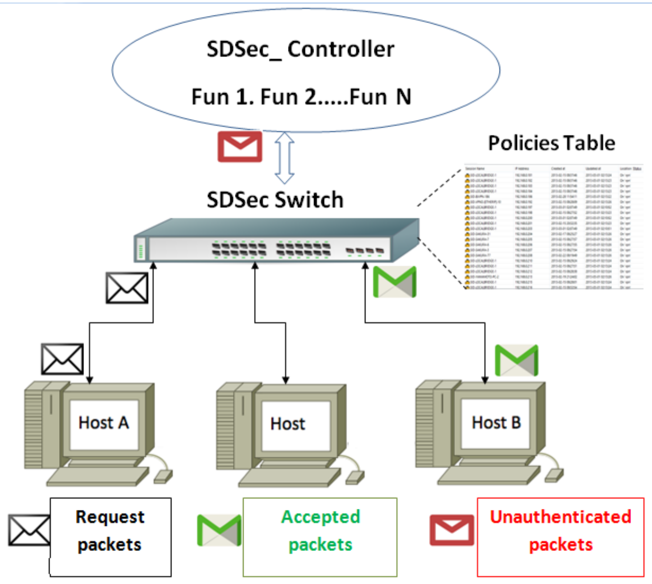

Fig. 2. The main components of SDSecurity and their integration to protect the network.

The idea of SDSecurity is to capture the basic idea of SDSec architecture, which is to abstract the data plane from the control plane. All the security mechanisms are abstracted from the security devices and set inside the software defined security controller inside the control layer in the SDSecurity. In such a way, when the host sends data traffic to another host in the network, the authentication process and all the security controls occur at the control layer not at the device level (physical layer). To explore how this happens, we show next the main elements of the SDSecurity and how these elements interact with each other to provide an experimental framework for testing the efficiency and effectiveness of SDSec systems and determine at which level they are protecting the network.

A. *Elements of SDSecurity*

SDSecurity is divided into three main customized components: the host, the switch and the controller. Figure2 presents the general view of the SDSecurity framework and its elements.

SDSec_Host. Mininet gives the users the ability to customize the host based on their requirements. By this flexibility we extend the basic “Host” element in Mininet to become SDSec_Host. It is meant to be a simple host which may send or receive data traffic through the network. This host is a virtualized host like other Mininet hosts with some extra parameters, which are required for SDSec purposes such as “Trust”, “zone_ID” (which specifies the trust zone for the host), “Per” (which defines the permissions for this host: read, write, both or none), “resources_consumption” (which presents the total amount of exploited resources) and the “scope” parameter to determine the allowable range for each host to communicate with other hosts. In traditional network systems, all the security techniques are kept inside the hosts, and when any new packet arrives, the host checks it to detect threats. On the other hand, in SDSec all of these techniques are transferred to the SDSec controller to get the benefits of SDSec. The benefits are centered around providing a single point of view and the flexibility to buy low cost commodity hardware.

SDSec_Switch. Mininet supports different types of switches: UserSwitch, KernelSwitch and OVSSwitch. For our experimental framework, we choose to extend the first one due to its simplicity. The SDSec_Switch inherits all the functions and parameters of the UserSwitch and implements extra functionality related to the SDSec emulation. Inside this switch a “Policies_Table” is created to store all the IPs of the authenticated SDSec_Host(s) and the related access policies for each one. If the IP of the sender is authenticated and the policies are achieved then the switch accepts its request and forwards it; otherwise, the switch consults the controller to take the proper decision of accepting or rejecting this request.

SDSec_Controller. This controller inherits the features of the “Controller” element in the Mininet which is the super class Controller for the all OpenFlow controllers. Mininet provides users the ability to choose the best controller implementation to control their topologies whether it is implemented locally inside the Mininet VM or externally by linking the topology to a remote controller. In our SDSecurity emulator, we use the basic controllers for Mininet like the super class Controller itself and POX controller [16] to implement our security techniques by customizing the controllers for our purposes. This controller is entirely software-based and it is considered to be the engine for any SDSec systems, since all security checks are generated inside it. The check_sum, intrusion_detection, and other security controls can be implemented here. For test purposes, we build a few security controls and define a set of access policies (policies are named P1-P9). When the controller receives a packet from an IP address, it tests this packet by applying the implemented security controls and the access policies to decide its status (authenticated or not) and checks if it achieves the predefined policies. After that, it pushes these changes to the SDSec_Switch to store the changes inside the “Policies_Table”. In our experiments, we use a simple Denial of Service (DoS) attack scenario as a case study to show how SDSecurity can be used. Other scenarios/attacks can be easily added and tested in the framework. Integrating all of these controls inside this controller will increase the security level, since several security controls can be applied to the same traffic to guarantee a high level of security and at the same time reduce the total cost to install and configure these controls and policies in every host.

Figure 2 shows the main components for SDSecurity and presents how it works when two hosts (Host A and Host B) need to communicate with each other. At first, Host A sends a request packet to the switch to which it is connected. After that, the switch checks the IP address of Host A, and if it is authenticated and the packet is in accordance with the policies, then it forwards the packet to Host B. Otherwise, it sends the packet to the controller or rejects it. The controller is responsible for applying different security mechanisms to discover if this sender is authenticated or not. Finally, the controller informs the switch with the results to take the proper action.

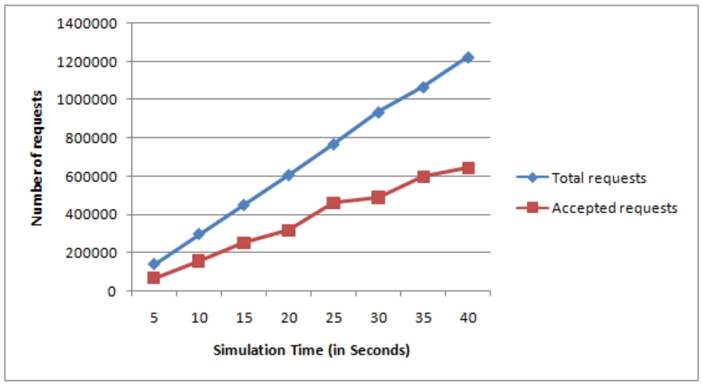

Fig. 3. The total requests compared to the accepted requests

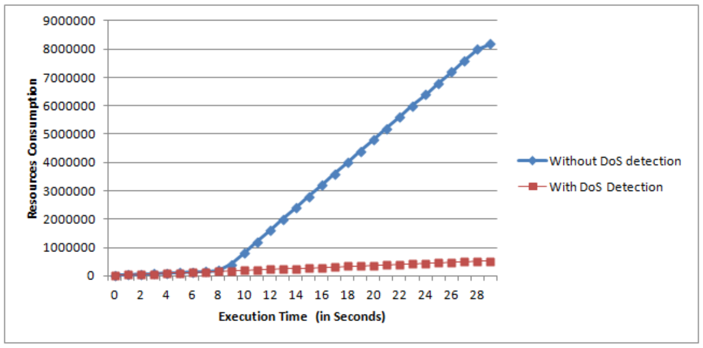

Fig. 4. Resources consumption for the victim Host

B. Experiments and Results

After discussing the main ideas in SDSecurity, we now discuss the implementation details. We also present a scenario in which a DoS attack is simulated and show how using SDSec concepts implemented in our SDSecurity framework can positively affect the system's performance under such attacks.

For the work environment, we use a Lenovo Z510 laptop with Intel Core i7-4702MQ CPU running at 2.20GHz, 8GB of RAM, 1TB of hard disk and Ubuntu 14.04.1 LTS 64-bit Operating System. On top of the Ubuntu Linux OS, we install the Oracle Virtual Box as a Virtual Machine Manager (VMM). Inside the Virtual Box, we add Mininet 2.2.0rc1 VM and link it with the Ubuntu shell. Next, we install the POX controller and customize it to handle SDSec functionality. Finally, we use the python programming language to build our code.

To do our experiments, we build our customized topology (SDSec_Topo). Inside this topology we use our customized SDSec hosts, switches and controller described previously to start this topology. Researchers can write their own topologies and scenarios to conduct different experiments and tests. To show how the SDSecuriy works, we build three tests: Test1, Test2 and Test3. For each test, we build a network with the following specifications: one Controller (c0), two Switches (S1 and S2) and 40 Hosts (h0-h39). We link the first 20 hosts with the first switch (s1). These hosts are responsible for sending packets to the remaining hosts (h20-h39), which are connected to the second switch (S2). The receiving hosts can be regular hosts or any other type such as storage hosts (built through the SDStorage framework [17]). The important thing is that the receiving hosts have some resources (storage, bandwidth, etc.) which the sending hosts are requesting. The following is a discussion of the considered tests.

Test1 shows the effects of enforcing the access policies on the packets. For test purposes we configure the parameter values of the hosts randomly and accordingly the access policies have been set. Figure 3 shows the variation between the total requests by the system and the actual accepted requests, which passed the access policies over different simulation times. As shown in the figure, the controller accepts the requests which satisfy the access policies and reject (drop) the remaining ones.

For Test2, we present the DoS attack detection control in the SDSecurity experimental framework to check how the system handles this type of attacks. We fix the simulation time at 30 seconds and generate a DoS attack around the tenth second. The DoS attack is initiated when the sender starts sending a lot of packets beyond the predefined threshold value, which is defined by the controller.

Test3 is not much different from Test2, except that we omit the part that is responsible for detecting the DoS attack from the SDSecurity experimental framework in order to show the effect of the DoS detection alone. For both Tests (Test2 and Test3), we set the values for the network hosts in a way that eliminates any packets being dropped due to the access policies.

Figure 4 shows the total resources consumption for the "victim" host only with applying the DoS attack detection (Test2) and without applying it (Test3). As we notice, the resources consumption for the victim will dramatically increase if the DoS attack detection is omitted, whereas, applying the DoS attack detection prevents this by blocking the attacker, which allows the system to continue functioning.

In figure 5 the variation between the total resources consumption and the victim consumption is presented. As shown in the figure we notice that most of the resources are reserved by the victim host when the DoS attack detection is not applied. On the other hand, by applying it we observe that the victim resources consumption is much less than the total consumption. In addition, it also shows the percentage of the useful/real resource utilization compared with the total consumption.

Figure 6 shows the variation between the total requests by the system, the accepted requests and the requests received by the victim host in both scenarios (with and without applying DoS attack detection). As mentioned previously, all the access policies are satisfied to show the effect of DoS attack detection only. As can be seen, if the detection is not applied then all requests are accepted and most of these requests are targeting the victim host. On the other hand, by using the SDSecurity DoS attack detection we notice that the requests that target the victim are much less than the total accepted requests. In fact, when the controller detects a DoS attack it blocks the attacker and eliminates its previous requests.

V. Conclusion

In this paper, an experimental framework (called SDSecurity) for Software Defined Security (SDSec) systems is

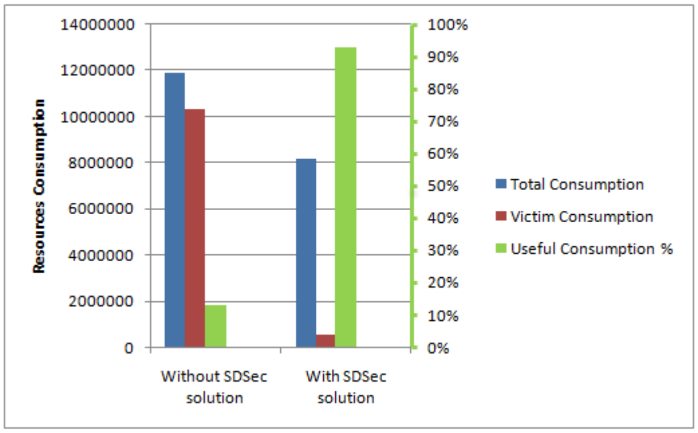

Fig. 5. The overall resource consumption for the system compared with the victim host resource consumption

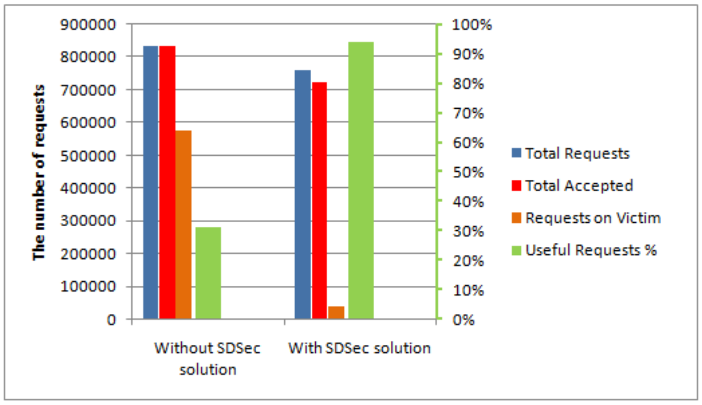

Fig. 6. The variation between the total requests, accepted requests and the victim requests

presented. First, the importance of SDSec is highlighted as it provides a centralized, programmable, flexible, simple and scalable solution to control and protect the systems with the rapid increase of threats and malicious attacks. After discussing existing SDSec solutions, the proposed SDSecurity experimental framework is discussed. It is built on the SDN Mininet simulator and its main components. We discussed the proposed framework, its components and environment. The developed emulator was tested to provide a proof of concept, and we explained how the users can build their customized topologies and scenarios to extend their testing or functionalities to suit their requirements. Future work will extend the SDSecurity to build a distributed controller to reduce its overhead and to improve its performance. We plan also to add more security controls inside each controller and create more tests to analyse its performance.

ACKNOWLEDGMENT

The authors would like to thank the Deanship of Research at the Jordan University of Science and Technology for funding this work, grant number 20150050. Also, they would like to thank IBM and IBM Cloud Academy for their support.

REFERENCES

[1] "Security challenges in sdn (software-defined networks)," https://www.sdncentral.com/security-challenges-sdn-software-defined-networks/ [Online; accessed Oct-2014].

[2] Q. Yaseen, Q. Althebyan, and Y. Jararweh, "Pep-side caching: An insider threat port," in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*. IEEE, 2013, pp. 137–144.

[3] A. Almodawar, M. Al-Ayyoub, and S. Mohammad, "Security-aware placement and migration algorithm in iaas interclouds," 2013.

[4] L. MacVittie, "Stop conflating software-defined with software-deployed," http://virtualization.sys-con.com/node/2929360 [Online; accessed Oct-2014], 2014.

[5] R. de Oliveira, A. Shinoda, C. Schweitzer, and L. Rodrigues Prete, "Using mininet for emulation and prototyping software-defined networks," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, June 2014, pp. 1–6.

[6] "Private cloud security, a catbird white paper," Catbird Networks, Inc, white paper, 2014.

[7] S. V. CA, "Catbird announces support for vmware nsx network virtualization platforml," http://www.catbird.com/company/catbird-vsecurity-with-vmware-nsx#.U-z9J9KSzHT [Online; accessed Oct-2014], 2013.

[8] M. Vizardl, "What software-defined security could mean for the channel," http://www.channelinsider.com/security/what-software-defined-security-could-mean-for-the-channel.html [Online; accessed Oct-2014], 2013.

[9] K. Walker, "Cloud security alliance announces software defined perimeter (sdp) initiative," https://cloudsecurityalliance.org/media/news/csa-announces-software-defined-perimeter-sdp-initiative/ [Online; accessed Oct-2014], 2013.

[10] "Vmware vshield virtualization-aware security for the cloud," VMware, Inc, white paper, 2010.

[11] "Vmware vcloud networking and security overview," VMware, Inc, white paper, 2013.

[12] "Netcitadels onecontrol platform the key to intelligent, adaptive network security," NetCitadel, Inc, white paper, 2012.

[13] "varmour inc," https://www.varmour.com/ [Online; accessed Oct-2014].

[14] D. Spalding, "Netcitadel and software defined security netcitadel unveils industry's first software defined security solution for centralized security intelligence in cloud, virtual and physical environments," http://goo.gl/p0jjBe [Online; accessed Oct-2014], 2013.

[15] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Networking Conference, 2014 IFIP*, June 2014, pp. 1–9.

[16] "Pox controller," http://sdnhub.org/tutorials/pox/ [Online; accessed Dec-2014].

[17] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdstorage: A software defined storage experimental framework," in *IEEE International Conference on Cloud Engineering (IC2E 2015)*, 2015.