2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

On the Design and Implementation of a Security Architecture for Software Defined Networks

Kallol Krishna Karmakar*, Vijay Varadharajan†, Udaya Tupakula†
Advanced Cyber Security Research Centre
Faculty of Science, Macquarie University, Sydney, Australia
kallol.karmakar@students.mq.edu.au*,[vijay.varadharajan,udaya.tupakula]@mq.edu.au†,

***Abstract*—In this paper, we propose techniques for securing Software Defined Networks(SDN). We describe the design of a security architecture that makes use of security applications on top of the SDN Controller to specify fine granular security policies based on domain wide knowledge of the domain and Security Agents to enforce these policies in the switches in the data plane. We have extended the Open Flow protocol to enable communication of the security policies between the security applications in the Controller to the agents in the switches. We have implemented the security architecture using POX Controller and demonstrated the operation of our architecture in a range of scenarios such as enforcing specific security policies for different traffic with different services, counteracting attacks such as Heartbleed and Shellshock as well as spoofing attacks, and protecting Content Management Systems(CMS) from data confidentiality attacks.**

***Index Terms*—Software Defined Networking (SDN) Security, OpenFlow, ACL, Source Spoofing, Policy Control.**

I. Introduction

Networking world has a new potential technology at its doorstep, called Software Defined Networks(SDN). As the name suggests SDN refers to software centric control over the data plane switches and networks. The basic idea is that the network is controlled centrally, which is achieved by decoupling the data plane from the control plane. To communicate with the data plane switches and network devices within its domain, a SDN Controller uses Open Flow protocol [1]. The OpenFlow protocol enable SDN Controllers to communicate their instructions to different switches and devices in the network data plane. The Open Flow switches have flow tables where the Controller specified instructions are maintained; these flow tables control the flow of packets between devices in the network. Recall in traditional networks, routers and switches have access control policies which are typically enforced using some form of Access Control Lists (ACLs); such switches and routers check the incoming packets against the access control rules and mechanisms to determine whether the request for transfer is to be granted or not. In the case of SDNs, the access policy specification happens at the SDN Controller and the enforcement of these policies occurs at the Open Flow switches using the flow tables.

Currently a lot of network attacks are happening all over the world. But one of the major security challenges is due to zero day vulnerabilities. A zero day vulnerability is a vulnerability for which there does not exist any known mitigation at the time the vulnerability is being exploited by attacks. That is, after the detection of the vulnerability, appropriate mitigations (e.g. software patches) may need to be developed and installed to counteract the attacks. Sometimes it is possible to detect anomalous behaviours using security systems, such as intrusion detection and intrusion prevention systems, which can also help towards identifying a zero day vulnerability; however with increasing complexity of systems and software, there are more and more zero day vulnerabilities. According to research from the High-Tech Bridge Security Research Lab [2], OS and application vendors are still taking an average of 11 days to release patches for critical security vulnerabilities while the low risk vulnerabilities take an average of 35 days. However, according to WhiteHat security research [3], it took an average of 193 days to apply the patches leased by the vendor to make sure the reported vulnerabilities are no longer exploitable. In addition to web application threats, a study by Symantec [4] reported that there were 54 zero-day vulnerabilities discovered in 2015, which has doubled from 2014. Therefore, it has become increasingly important not only to make the end hosts with their OS and applications more secure, but also to detect attacks using the signatures developed by automated tools such as [5] and the security tool vendors such as Snort[6] while the OS and application vendors are developing patches to the open vulnerabilities. In the case of SDN networks, the switches, the end hosts and most importantly the SDN Controller in a domain are vulnerable to zero day vulnerabilities. In particular, the Controller becomes a single vulnerable point of attack, as the whole domain is compromised if the Controller is compromised. A discussion of various security vulnerabilities and threats at different points in a SDN network can be found in Kreutz [7].

In this paper, we describe a security architecture for SDNs, which consists of Northbound security application as well as security components in the switches to securely manage a SDN domain. We have developed a security architecture consisting of secure Northbound applications for the SDN Controller as well as secure agents implemented in the Open Flow switches to enforce the security mechanisms. The first security component is a security application for the Controller, which enables the specification of fine granular access policies for the SDN domain. The enforcement of such policies using the Security Agents in the switches allows us to provide *on-demand security* for the user data in the network. The second

978-1-5090-4297-5/16 $31.00 © 2016 IEEE
DOI 10.1109/HPCC-SmartCity-DSS.2016.138

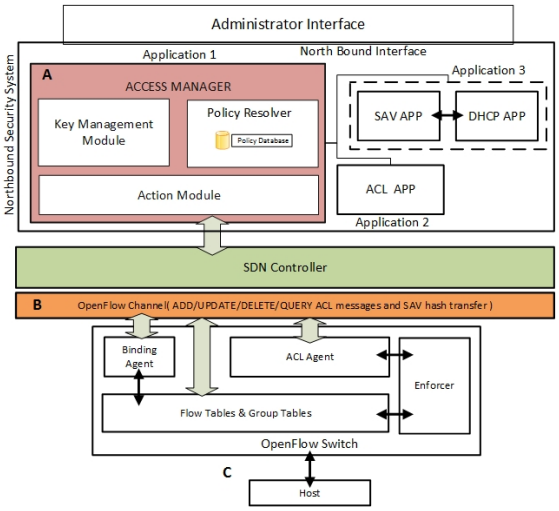


Fig. 1: Security Architecture

security component provides a capability to enforce the access control policies in the forwarding domain. With the help of this component, the administrator of the Controller is able to specify the access rules that will be enforced in the forwarding switches and devices in the SDN. Then we have developed security extensions for the Open Flow devices to enforce these access control policies. We also demonstrated, how the proposed security architecture can help to deal with unpatched vulnerabilities. Finally, we have used our security architecture to demonstrate a Secure Content Management System (SCMS) enabling different content confidentiality policies for different users. Here we have developed a mechanism to check the authenticity of the source address (e.g. IP or MAC) to counteract spoofing attacks in SDNs.

Hence, the overall focus of this paper is to develop techniques for securing SDN domains and to describe a security architecture comprising various security components and applications. Our main approach to designing the security architecture for SDNs is to develop secure Northbound applications for determining and specifying security policies, and to propose extensions to Open Flow protocols to allow communication of the security policies between the security applications in the Controller and the Security Agents in the switches.

The paper is structured as follows. Section II describes our security architecture for SDNs. Section III considers the different implementation scenarios of the proposed security architecture for the enforcement of policy control in the forwarding domain, for counteracting well known zero day vulnerabilities, and for ensuring content management security and preventing source spoofing. Related works are discussed in Section IV. Finally, Section V concludes the paper with a discussion and some potential further work.

II. System Architecture

This section gives an overview of our proposed security architecture for SDNs. Figure 1 shows our security architecture with its three sub-components: A) Northbound Security System, B) Extensions to the Open Flow Protocols, and C) Security Agents in the Open Flow Switches. The Northbound Security System in our security architecture consists of 3 applications; Application 1 coordinates Applications 2 & 3 (as shown in Figure 1). Application 1 has an Access Manager component which guides all the other applications as well as the main functionality of the Application 1 itself. The administrator interface to the Northbound Security System is used for configuring security. Application 1 manages the SDN domain by enabling the specification of fine granular access policies. It also helps to provide confidentiality of the host data which we will describe later. Application 2 is used in the enforcement of access control policies in the Open Flow switches and devices. We will describe how Application 2 is used to counteract certain security attacks as well as helping to secure the Content Management System (CMS). Application 3 is used to validate the source IP/ MAC address of the end hosts connected to the Open Flow switches. The Source Address Validation (SAV) module uses the information from the DHCP in the validation process. For these applications to work properly, it is necessary to extend the Open Flow protocol specifications (denoted as B in Figure 1) as well as specify additional messages for communicating security policies to the Security Agents in the switches. Security Agents are software modules implemented in the Open Flow switches for enforcing security functions as specified by the policies in the SDN Controller. They are denoted in Figure 1 as C. Let us now consider each of these security components in detail.

A. Northbound Security System

In this section, we describe the major component of our security architecture, the Northbound Security System with its three Security Applications.

1) Security Application 1: Security Application 1 is the core application that is used for maintaining the SDN domain wide knowledge about the various switches/devices, services and policies associated with them. This information is used in determining what policies need to be enforced at the various agents implemented in the switches and the associated security related decisions. Furthermore, Security Applications 2 and 3 interact with the Security Application 1 for enforcing the security policies in the switches and for counteracting the attacks respectively.

Application 1 is the policy based guidance module which controls the SDN domain according to the policies specified by the security administrator(s). Here, we first describe the functioning of Application 1 and then discuss the components enable this application to work properly.

The heart of the Northbound Security System is the Access Manager component. The security administrator specifies the access policies to be enforced by the switches when it receives different types of packets for transfer to different end devices and users. We have specified the access policies using a language based approach, implemented using templates and stored in a policy database (MySQL database). The policy

P_ID	USER	SRC_IP	SRC_MAC	DST_IP	DST_MAC	SERVICES	SEC_PROF	SWITCH_SEQ	PERM
3	Student	*	*	*	*	*	enc	*	P
5	Alice	172.56.16.02	48:2C:6A:1E:59:2F	*	*	*	enc	*	P
6	*	172.56.16.02	48:2C:6A:1E:59:2F	*	*	*	enc	*	P
7	*	*	*	*	*	20,21,22,23	enc	*	P
8	*	*	*	172.56.16.06	56:2D:7F:2E:50:FF	80	enc	*	P
9	*	172.56.16.02	48:2C:6A:1E:59:2F	172.56.16.08	60:FF:2F:D2:60:CC	20,21,22,23	enc	SW1;SW3;SW4	P
10	*	172.56.16.04	48:2C:6A:1E:60:FF	172.56.16.06	56:2D:7F:2E:50:FF	80	enc	SW1;SW5;SW4	P
11	Alice	172.56.16.02	48:2C:6A:1E:59:2F	172.56.16.08	60:FF:2F:D2:60:CC	20,21,22,23	enc	SW1;SW3;SW4	P
12	Alice	*	*	172.56.16.08	60:FF:2F:D2:60:CC	20,21,22,23	enc	SW1;SW2;SW4	P

Fig. 3: Policy Database

specification incorporates rules based on different parameters such as user and device/host identities, TCP/IP header information as well as services running on the destination hosts. This enables the security administrator to specify fine granular access control policies on packet flows in the switches in the data plane.

Let us now consider how the system works in practice. When a host's packet comes to a Open Flow switch, first the switch checks its flow table to determine how to route the packet. For a new flow, no flow entries exist in the switch flow table. Thus, the OpenFlow switch does not know how to process the packets in the flow. In this case, the packet is forwarded to the Controller. At this point, the Access Manager in our Security Application 1 intercepts and strips the packets in the message to extract different attributes such as source IP address, destination IP address, any MAC addresses and port numbers. It then passes these attributes to the Policy Resolver module in the Security Application 1. The Policy Resolver module then checks these parameters against the Policy Templates stored in the MySQL policy database. It then informs the Access Manager, which then instructs the Action Module to set the rules in the forwarding devices according to the access policy. The Action Module then sends the appropriate instruction (Flow_Mod() instruction) to the respective switches. We have developed the application is such a way that it only requires minor changes to the Action Module to implement this application with different SDN Controllers. We also provide on demand confidentiality and integrity services for the user data in the forwarding plane. If a user/host requests that its information should be protected for confidentiality and/or integrity, then the Action Module activates the Key Management Module which generates appropriate symmetric keys. The Action Module then distributes the symmetric keys to the switches that are connected to the source host and the destination host. From then on, the flows are encrypted and integrity checksum provided at the switch that is connected to the source host using the key provided by the Controller. At the receiving end, the same key is used by the switch that is connected to the destination host to decrypt the packets and validate the integrity checksums.

Let us now briefly consider the functionalities of the different components of the Security Application 1:

Access Manager:

Access Manager is the heart of the architecture and has some major responsibilities, for instance, analyses packet_in messages, maintenance of policies in the Policy Resolver Module, interactions with the Key Management Module (e.g. requesting generation of keys), and setting up the instructions for flows to be transmitted to the Action Module as well as exchanging information with Applications 2 and 3. Figure 2 shows the work flow of the Access Manager.

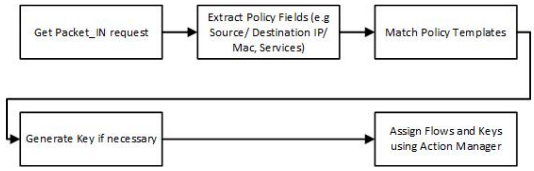


Fig. 2: Access Manager Work Flow

Policy Resolver Module:

Policy Resolver Module is responsible for maintaining the MySQL database where policies are stored. This Module fetches the particular Policy Templates and sends the report back to the Access Manager.

Policy Database:

The Policy Database stores the Policy Templates and the associated attributes. Figure 3 shows the view of the policy database. **P_ID** is the policy database id number. **User** is the respective user name. **SRC_IP, SRC_MAC** are the Source IP and MAC addresses respectively; similarly **DST_IP** and **DST_MAC** are the destination IP and MAC addresses. **Service Field** denotes the different types of user traffic. **SEC_PROF** denotes the security profile, which indicates the security services requested by a user such as confidentiality, integrity, authentication services. **Switch_SEQ** is the actual flow path a packet is supposed to take as dictated by the administrator from source to destination.

Action Module:

The Action Module acts as a message passing bridge between the Controller and the forwarding device/switch. It captures the Packet_IN request and sends it to the Access Manager. It also sends the Flow_mod() and keys to the respective forwarding devices as instructed by the Access Manager. It also passes the information from Application 2 and Application 3 to the Controller. These informations are passed to the respective Security Agents in the Open Flow switches. We have specified a set of special messages to pass the information to the Security Agents in the switches.

2) Security Application 2: The Security Application 2 makes use of information stored in Application 1 for determining specific access control mechanisms that need to be applied at

the switches for each flow. For example, the Access Control Lists are determined based on the services running on the end hosts as well as the need to drop certain attack traffic. Let us look at some examples.

Consider, for instance, an end host which is running services related to a Content Management System (CMS) such as Wordpress and Drupal. Users often exploit the access control vulnerabilities [8] in these applications to access or change data related to other users. There has been some recent work [9] which secure these content management applications by implementing additional access control lists at the proxy server. However this approach is limited to securing the application at the server end. We have extended this approach by enforcing access control at the switches to secure such content management applications. We have also enforced the access control on the switches at the user end. Hence our security extension can achieve dropping the unauthorised user access request at the source end, which is much more efficient. We assume that there is an automated signature generation mechanism such as [5], [6] that can be used to detect attacks targeting the open vulnerabilities in the applications and OS. Our main aim is dynamic application of these signatures using our SDN application. Consider another example scenario, where the destination host is running an application with SSL that is vulnerable to the Heartbleed attack [10]. Once again, we have applied access control at the switches, both at the source and the destination, to drop the attack traffic. This is achieved by checking the payload of the network traffic. The Access Control List is modified dynamically using the Security Application 2 that is running at the Northbound Security System. We have introduced new Open Flow messages to perform update, delete, add and query operations. The Enforcer Module in the Security Agent (denoted as C in Figure 1) enforces the Access Control List in the data plane. It matches the payload of a flow with the rules in the ACL Agent, in this case signature for Heartbleed attack; if a match occurs, then the rule is enforced. The default case is to drop the matched attack traffic packets.

3) Security Application 3: The Security Application 3 is concerned with spoofing attacks, as spoofing of host IP and MAC addresses are quite common. When a host machine boots up, it is assigned an IP address using a Dynamic Host Control Protocol (DHCP) application running in the Controller. A time stamp is associated with each DHCP assignment. The Source Address Validation (SAV) Module works in parallel with the DHCP application. It intercepts the DHCP messages and creates a binding table type data structure [11]. It consists of an Anchor (linked to physical or link layer property of the host, e.g. Port ID/ MAC address), IP Address of the host, DHCP State and Lifetime. After an IP assignment, a hash of the parameters Anchor and Address is created by the Source Address Validation (SAV) Module. Then to authenticate the validity of the connected host, the SAV Module intercepts the Packet_IN messages and calculates the hash of these parameters, and then checks to see if it matches with the previously stored hash. Such a mechanism enables the detection of spoofing attacks.

B. Extension to OpenFlow Messages

In this section, we address how the Security Applications 2 and 3 communicate with the Security Agent in the switches using the Open Flow channel (shown as B in Figure 1). There are two types of messages(simple string format) for the two applications. The Security Application 2 communicates with the ACL agent in each of the OpenFlow switches via new OpenFlow messages. The general syntax of them are:
$< \ user \ >< \ operation \ >< \ URI \ >>< \ Action(Optional) >$

- ADD RULE: To add an URI information to an ACL agent in the Open Flow switch, the security administrator passes a string "$USER + URI$".
- DELETE RULE: To delete any rule, the information "$USER - URI$"is sent. Here '-'indicates that the URI rule under a specific user must be omitted.
- UPDATE RULE: To update a rule, the '$>$'symbol in the operation block of the syntax is used. For example, "$USER > URI$"updates the URI information for a specific user.
- QUERY RULE: A Security administrator can query a rule for a specific user using "$USER?$".
- RULE EXCEPTION: To enforce host IP and MAC based Access Control Lists, we require an exception. This is achieved by sending the IP or the MAC address as a string through the Northbound Security System. It is also possible to keep track of the payload size, e.g. for the Heartbleed attack.

To transfer the hash function created by the Security Application 3, another Open Flow message has been introduced. This is referred to as OFPRAW_OFPT_HASH. Its purpose is to transfer the created hash to the binding agent in a Open Flow switch.

C. Security Agents in Open Flow Switches

Security Agents are software modules in Open Flow switches that work in coordination with the Northbound Security System (shown as C in Figure 1). The main task of the Security Agents is to enforce the security mechanisms and check newly added specific Open Flow messages from the Controller. Let us consider now the operation of the Security Agents with the Security Applications 2 and 3.

Security Application 2: For Security Application 2, we have developed two modules in the Open Flow switches. They are the ACL Agent and the Enforcer.

ACL Agent: Figure 1 shows the basic block diagram of our modified Open Flow switch. ACL Agent has two main tasks. It maintains the communication with the Controller Northbound using the newly defined Open Flow messages. The ACL Agent maintains a table to store the URI information sent by the Security Application 2. The URI information needs to be checked in the payload. In some cases, we might want to drop a particular flow from a specific IP address. In this case, this particular IP address is transferred to the ACL agent from the Security Application 2. The ACL agent can differentiate between IP and URI.

Pseudocode for *ACL Rule* Enforcement

```
1. Get packet and acl_rule
2. Split acl_rule into acl_user and acl_URI
3. Construct regex string for acl_user and acl_URI
4. Extract user and payload from packet
5. if((regexec of user and acl_user)==true &&(
regexec of payload and acl_URI)==true)
6.        return packet_id
7. else
8.        return "No Match"
9. end
```

Enforcer: The Enforcer Module enforces the ACL rules in the Open Flow switches. This module fetches the URI rules for a particular USER from the Agent data structure. Then it tries to match the payload for that particular information. If it finds a match, it drops that particular host's flow.

Security Application 3: For Security Application 3, we have developed the Binding Agent in the Open Flow switch. The main task of the Binding Agent is to store the hash of (Anchor + Address) and check the hash against the generated new hash to make the decision about spoofing.

Regular Expression: ACL rule matching is done using Regular Expressions. GNU C regex[12] is used for this, which is simple and easy to use. Also, Open vSwitch is constructed mostly using C, which makes GNU C regex easily adaptable to the environment. To do the comparison between ACL rule and payload information, rules are stored in a local database first and then a regular expression string is constructed with each ACL rule. Finally, created regex strings are compared to the Payload information. ACL agent in the Open vSwitch accepts the ACL rules sent by the Administrator via Security Application 2. ACL agent stores the ACL rule in an Array. When the packet transaction happens, ACL agent pulls up the ACL rules and create regex strings for them. These regex strings are matched against the packet payload for similarity. If a match is found, ACL agent informs the enforcer, which drops the flow. Pseudo code shows the matching process for the GNU C regex.

Similar type of approach is used for spoofing check. Binding Agent stores the Address Hashes in database array with ID numbers. Now, when a packet comes in, its address authenticity is checked against the stored hashes. Thus, only the legitimate address flows are stored in the flow table, there by dropping others.

III. Implementation

In this section, we describe the implementation scenarios and performance of the modified Open vSwitch module. First, in section A we represent a scenario where Security Application 1 controls the forwarding domain according to the administrator policy. Secondly, we have illustrated access control scenario for CMS using Security Application 2 is section B. Third section (C), represents how Security Application 2 can help to secure unpatched systems. Section D, shows how Security Application 3 can help in case of spoofing attacks in SDN environment. Finally, in section E we represent performances for modified Open vSwitch.

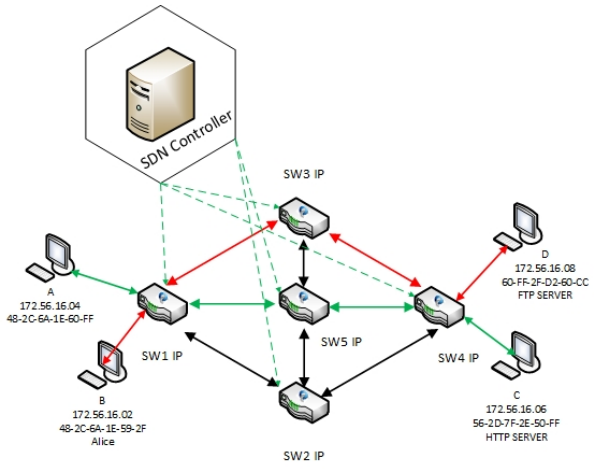
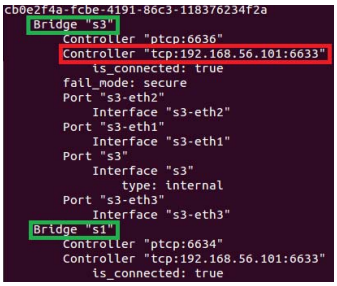


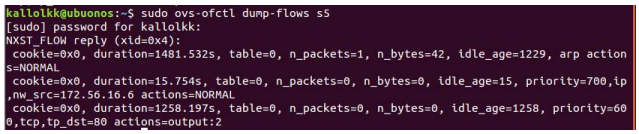
Fig. 4: Experimental Setup for Controlling Forwarding Domain in SDN

A. Controlling Forwarding Domain:

The first scenario we have implemented demonstrates how our security architecture can securely route the traffic from different users for different services based on security administrator defined policies in a SDN domain. We implemented a switch matrix of five OpenFlow switches in Open vSwitch. Figure 5a shows the Open vSwitch bridges(Shown in green square) for each switch. The bridges are connected to a remote SDN controller(Shown is red square). We used POX as the SDN Controller, which had secure links as shown by the dotted green lines in Figure 4. We have created Port connections in Open vSwitch and attached four Virtual Machines (Hosts) running in Oracle VM Box. Two hosts A and B are connected to Switch 1. The IP (DHCP App assigned) and MAC addresses are shown in Figure 4. We require that any HTTP request from host machine A is to be forwarded through switches SW1→SW5→SW4 (shown with continuous green lines). The FTP traffic and request from the Host B are to be forwarded through the switches SW1→SW3→SW4 (shown with continuous red lines). To achieve this, we created two policies in the Policy database, shown as Policy ID (PID 9, PID 10) in Figure 3. Now during the first run, when the Packet_IN request from A goes to the OpenFlow switch SW1. The switch checks its flow tables. As there is no flow entry, the switch forwards the request to the Controller. The Controller passes the request to the Access Manager. The Access Manager checks the policy database for the appropriate policy. As the Controller has a topological view of the whole domain, the Access Manager has also access to this view. The Access Manager queries the Policy Resolver for related policy. The Policy Resolver responds with the policy which states that the traffic should be protected for confidentiality by encrypting the traffic. Now the Access Manager requests the Key Management module



(a)



(b)

Fig. 5: Open vSwitch Configuration of the Implementation (a) Switch configuration and (b) Switch five flow dump

to generate symmetric keys for the flow. Then the Access Manager instructs the Controller via the Action Module to send the Keys and Flow_mod() messages to the respective OpenFlow switches, in this case, SW1, SW5 and SW4. Now for the subsequent packets coming from the same source will be forwarded through the same path of switches, as there will be flow entries regarding this in each of the OpenFlow switches.

Figure 5b shows the installed flows in SW5. SW1 & SW4 switches will also have similar type of flows installed in them by the application running over the controller. The first flow is used for the network discovery(ARP). Second flow, is for the response from the web server running in IP 172.56.16.6, which tells to the controller that any IP packet from the Web-server should be handled normally. Third flow, is dedicated to HTTP packets and indicates that all HTTP traffic should take an output path Port 2(in Open vSwitch).

Similar sequence of events occurs for the second policy. Now the Access Manager will request the Controller to install Flow_mod() in the switches SW1, SW3 and SW4. As a result, the host will be able to communicate with the FTP server. In this case, there is no ACL rules but the SAV application did check the authenticity of the connected virtual host.

B. Access Control for CMS:

In this scenario, we used the Open vSwitch (OVS) environment. We have used the similar setup of previous scenario: POX as the Controller and executed Security Application 1, Security Application 2(ACL App) and Security Application 3(DHCP, SAV). We created two instances of OVS switches (SW1 & SW2) as shown in Figure 6. We connected a WordPress VM with SW1(172.56.16.6) shown in red block in Figure 7. WordPress is a Content Management System (CMS) with some well known vulnerability issues as mentioned in [9]. We developed a rule based ACL for this WordPress VM. We

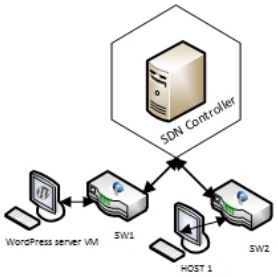


Fig. 6: Experimental Setup to Demonstrate SDN based Access Control for CMS

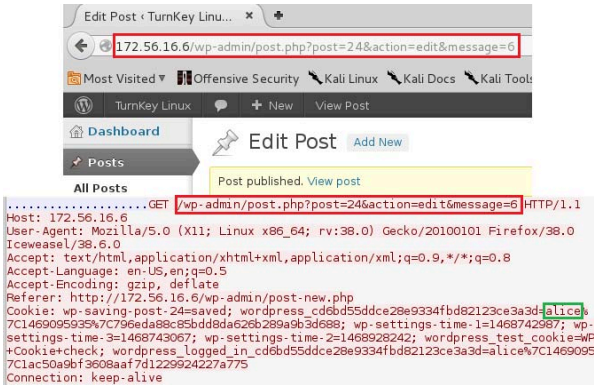


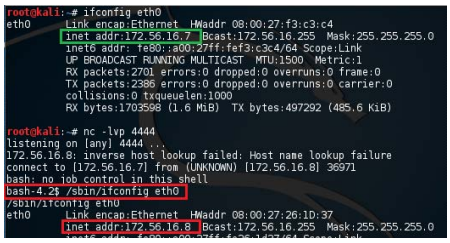
Fig. 7: Wordpress post by Alice

have the host machine connected to the switch SW2 denoted as HOST1(172.56.16.7). Using the Northbound ACL application, we implemented the ACL rules for specific user. Our approach was to give access to some of the users for that WordPress site. Table I shows the users and their roles.

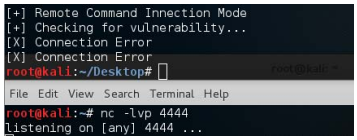
There could be several vulnerabilities in WordPress such as contents of one user can be exposed to others, files of one user can be downloaded by others and RSS feeds could be exposed to non members. Our security architecture is able to restrict such vulnerabilities. From Table I, normally it is possible for Joe to download and access Alice's content, since both of them has the same level of access to the site/page/post. In both SW1 and SW2 we have introduced an access control rule "$alice + /post.php?post = 24\&action = edit\&message = 6$". It says that access to this post as administrator will be limited to Alice only. When Joe tries to edit/steal something from this page, his flows were dropped. An example for a valid REST API wireshark trace is shown in Figure 7. For a valid user, if the post, action and message IDs match in the REST URI sting then only the flows will be permissible. For

TABLE I: Roles for user in WordPress VM

USER	ROLE
Alice	User
Bob	Adminstrator
Joe	User



(a)



(b)

Fig. 8: Reverse Shellshock attack (a) Successful approach and (b) Failed attempt/ lost connectivity

instance, Figure 7 shows a post(24), action (edit) and message (6), by user alice(shown in green block), which is permissible. Thus, alice is able to edit the post. The same action was tried by Joe and his flows were dropped. This example was tested using GNU C Regular Expressions as explained in section II-C ; similarly various other cases were tested.

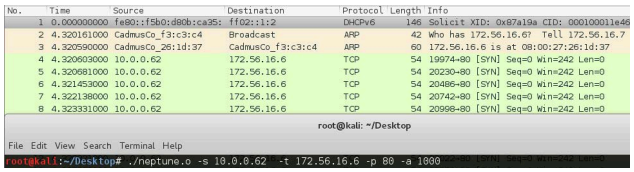
C. Defending Unpatched Systems:

In another experiment, we used the same syntax to stop shell attacks. We chose a Kali Linux VM as the host machine and tried to launch a Shellshock attack on the PentesterLab VM (instead of the WordPress VM in figure 6). The PentesterLab VM did not have updated *Bash shell* and so it was unable to withstand the Shellshock attack. We get the bash environment in the Reverse Shellshock attack. Figure 8a shows a successful Reverse Shellshock attack. We used a python script to initiate Reverse Shellshock attack from a kali VM running IP 172.56.16.7 (shown in Green block). The Vulnerable Pentester VM IP is 172.56.16.8 (shown in Red blocks). After the attack we are able to get the shell successfully as show in the figure. Now to stop this, we included a single rule in SW1 and SW2. The rule is "$shellshock - signature > drop$"; this rule says that for any packet coming from any user containing Shellshock signature will be dropped. Now, we ran the same script to compromise the PentesterLab VM, but this time the VM lost connectivity(shown in Figure 8b).

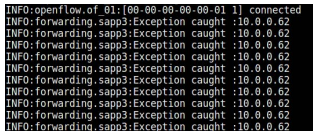
With the same setup, we replaced the PentesterLab Shellshock vulnerable VM with a HeartBleed vulnerable VM. We took a rule exception approach and enforced a rule to check the length of the Heartbeat messages. We took the default length of the payload + padding to be 16 bytes. After that we tried to launch an attack which was successfully dropped, though the normal heartbeat messages working properly.

D. Spoofing Check:

In the same environment as shown in figure 6 we tried to spoof the IP address of the PentesterLab VM(172.56.16.06)



(a)



(b)

Fig. 9: Spoofing attack (a) Successful approach and (b) Failed attempt/ lost connectivity

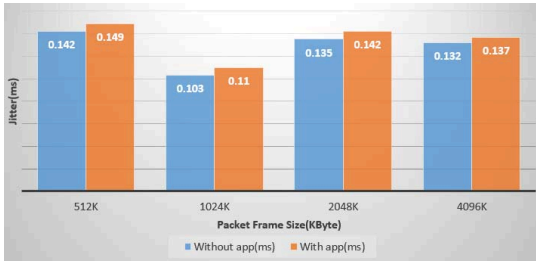


Fig. 10: Frame size vs Jitter

from Kali Linux(172.56.16.07). Successful spoofing attack wireshark trace is shown in figure 9a. Then we activate our security applications. After that, when the machine was turned on, the Kali VM was assigned an IP address by the DHCP running in the Controller. Also at the time of assignment, the SAV application created a hash of the Anchor (MAC in this case) and the assigned IP address. Then it sent it to the agent at each of the switches. After that, when we tried to spoof the IP address of the PentesterLab VM, from Kali VM, all the spoofing IP traffic is being blocked (shown in figure 9b). A small trick in the application code helped us to drop the spoofing attacks generated from Kali VM.

E. Results:

Figure 10 shows the performance of the OpenvSwitch. In this experiment, we have connected two VMs using OpenvSwitch. We have used Iperf [13] to calulate jitter between communication with and without our application. In the bar chart, blue bars show the OpenvSwitch performance without our applications running on the SDN controller. However, the controller was running the default applications, for instance, forwarding application and device discovery application. With slight increase in jitter Orange bars show the performance with our applications running in controller Northbound. For these experiments, the native machine hosting SDN controller is a Core i7 4790 HQ @ 3.60 GHz with 32 GB of RAM. The performance varies with the variation of the SDN controller native hardware.

IV. Related Works

Motivation of our work came from FlowWatcher [9], which implements Nginx reverse proxy to detect violation of the User Data Access (UDA) policy. FlowWatcher was implemented in a legacy network. It tracks the UDA violations by payload checking against a set of USER and URI rules enabling the network to defend from six CVE bugs. We have followed a similar approach and applied it to the SDN environment. We have extended the techniques to counteract a range of attacks in the SDN domain. Our security architecture is able to enforce fine granular access policies in the switches as well as defend against attacks such as source spoofing.

There have been recent works that have extended SDN Controllers with security such as FortNox, which provides a role-based authorization and security constraint enforcement for the NOX Open Flow Controller. There are some other works on SDN Controller based policy control which includes Nettle [14], Maple[15], and SNAC. Flow-based Management Language (FML)[16] by Hinrichs, Pyretic [17] & Frenetic[18] are the most popular SDN language based approach to controlling forwarding domain. However, none of them are able to enforce policy control at a fine granular level. None of these provide mechanisms to counteract source spoofing and ACL enforcement. Our security architecture is able to provide fine granular policy control together with the provision of services such as content confidentiality and integrity.

Our Source Address Validation approach is based on the Virtual Address Validation Edge (VAVE) proposed by Guang et al.[19]. VAVE is an upgrade of Source Address Validation Improvement (SAVI) [20]. SAVI was only used in legacy devices. VAVE is an application than runs over NOX to detect spoofing flow entries. The checking is done against a set of rules. On the other hand our approach is much simpler. We collect the source information from the devices and enforce the rule checking on the data plane. Hash checking makes it possible to validate the source address quickly.

There are a number of other firewall based techniques in the literature , for instance [21], [22], which deal with blocking the flow rules and resolving conflicts between flows. Our approach is somewhat different as we have filtered the flows based on source validity and authenticity of the payload.

V. Conclusion

In this paper, we have developed techniques for securing SDN domains. We have described the design of a security architecture for SDNs which consists of security applications running on top of the Controller to specify security policies and Security Agents implemented in the Open Flow switches to enforce the security policies, and security enhancement to Open Flow protocols between the Controller and the switches. Then we described the operation of the security architecture in a range of scenarios: We have shown how security architecture can be used to enforce different security services for different types of traffic such as on demand confidentiality or integrity. We have demonstrated how the security policy enforcement can be used to prevent two well known attacks Heartbleed and Shellshock for unpatched systems. These SDN App based techniques can be used to drop malicious flows. We showed that our architecture is also able to counteract spoofing attacks by validating the source addresses of the hosts. Furthermore, this is helpful to prevent the attacker from launching attacks on the SDN Controller. We have also shown how the security architecture and policy rules in the switches can be configured to protect Content Management Systems from data confidentiality attacks.

References

[1] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] Ht bridge (2014), high-tech bridge research: Web application security trends in 2013. https://www.technologyreview.com/web/21537/. Accessed 14 June 2014.

[3] J. Grossman, "How does your website security stack up against your peers?" *WhiteHat Security Website Statistics Report*, vol. No. 12, 2013.

[4] Symantec interbet security threat report 2016. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.

[5] S. Kaur and M. Singh, "Automatic attack signature generation systems: A review," *IEEE Security & Privacy*, vol. 11, no. 6, pp. 54–61, 2013.

[6] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, no. 1, 1999, pp. 229–238.

[7] D. Kreutz *et al.*, "Towards secure and dependable software-defined networks," 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. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491199

[8] (2014) Cve. [Online]. Available: cve.mitre.org

[9] D. Muthukumaran *et al.*, "Flowwatcher: Defending against data disclosure vulnerabilities in web applications," in *22nd ACM Conference on Computer and Communications Security (CCS)*, ACM. Denver, CO, USA: ACM, 10/2015 2015.

[10] B. Schneier, "Heartbleed," *Schneier On Security. Blog*, 2014.

[11] J. Bi *et al.*, "A source address validation architecture (sava) testbed and deployment experience," 2008.

[12] Gnu c regex. http://www.gnu.org/software/libc/manual/html_node/Regular-Expressions.html#Regular-Expressions.

[13] A. Tirumala *et al.*, "Iperf: The tcp/udp bandwidth measurement tool," *htt p://dast. nlanr. net/Projects*, 2005.

[14] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Practical Aspects of Declarative Languages*. Springer, 2011, pp. 235–249.

[15] A. Voellmy *et al.*, "Maple: Simplifying sdn programming using algorithmic policies," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 87–98.

[16] T. L. Hinrichs *et al.*, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 1–10.

[17] J. Reich *et al.*, "Modular sdn programming with pyretic," *Technical Reprot of USENIX*, 2013.

[18] N. Foster *et al.*, "Frenetic: A network programming language," in *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[19] G. Yao *et al.*, "Source address validation solution with openflow/nox architecture," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*. IEEE, 2011, pp. 7–12.

[20] J. Wu *et al.*, "Source address validation improvement (savi) framework," Tech. Rep., 2013.

[21] P. Porras *et al.*, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 121–126.

[22] T. Javid *et al.*, "A layer2 firewall for software defined network," in *Information Assurance and Cyber Security (CIACS), 2014 Conference on*. IEEE, 2014, pp. 39–42.