TECHNICAL REPORT 2

APNIC Foundation's Information Society Innovation Fund (ISIF Asia) Project

SECURING SOFTWARE DEFINED NETWORK (SDN) ARCHITECTURES

Attack Detection on the Software Defined Networking Switches

Prof. Vijay Varadharajan, Dr Uday Tupakula, Dr Kallol Karmakar

Advanced Cyber Security Engineering Research Centre (ACSRC)

The University of Newcastle

Version 1.0

The authors would like to thank Information Society Innovation Fund (ISIF Asia) for supporting this Project.

Executive Summary

SDN is rapidly emerging as a disruptive technology, poised to change communication networks much the same way cloud computing is changing the "compute" world. It is altering the texture of modern networking, moving away from the current control protocols dominant in the TCP/IP Internet stack towards something more flexible and programmable. However, the SDN is vulnerable to attacks and requires security services and mechanisms. The first report focuses on the SDN infrastructure's data plane threats and attacks. In the SDN infrastructure, the switches are easier to target for the attacker, as they are located on the edge and are connected in close proximity. We have presented the various threats in the SDN data plane and considered a range of attacks against the OpenFlow switches. Furthermore, we described some advanced virtual switch-related attacks specific to cloud infrastructure.

SDN adopts a centralised framework to facilitate fine-grained application-driven network management. As its security is in its infancy, significant work is needed to deal with different attacks in the SDN data plane. This report will focus on developing a security architecture that can detect and prevent attacks on the SDN data plane. First, we will present the security requirements for the SDN infrastructure. These requirements are formulated based on the possible attack cases explained in the first report and the design flaws in the data plane devices. Secondly, we will present a Switch Security Management Architecture (SSMA) for SDN. We will explain its components and associated functions in detail. The SSMA design is modular and can be extended based on new requirements or suite various SDN Controllers. Finally, we will present how the security architecture can defend against some attacks on the SDN data plane.

CONTENTS

1. INTRODUCTION

Network management is a critical process which involves configuring and monitoring the network devices using cost effective methods. Traditionally, this has been a painstaking process involving manual configuration of individual devices across the network topology. Furthermore, enterprise network management is becoming an increasingly complicated process with rapid migration of services to hybrid cloud environments and users accessing these services with mobile devices.

Software Defined Networking (SDN) is an innovative architectural approach to modern computer networks where the control features of the infrastructure are abstracted from the network devices themselves and placed into a centralized location. This abstraction of the network allows for novel approaches to network management, including third-party applications, dynamic and adaptive configuration, and cloud-hosting. In particular, the applications developed for SDN Controller add value in ways that would have been difficult or impractical before. For instance, SDN facilitates rapid development of software solutions for a wide range of network issues in traditional network architectures such as static configuration, non-scalability and low efficiency. Hence, SDN is becoming immensely popular for network management among modern enterprise networks, cloud and network service providers.

Although SDN has several benefits, security in SDN is still in its infancy and different attacks are possible in such networks [1, 2, 3, 4, 5, 6, 7, 8, 9]. In particular, the SDN architecture has a larger attack surface than traditional networks due to reasons such as single point failure of the controller, exposed network-wide resources, and the possibility of malicious applications severely disrupting network operations. The devices in the data plane are also vulnerable to different attacks since they are connected to control plane and the data plane. Hence, attacks are possible from malicious applications in the control plane, exploiting the weak or lack of secure communication between the devices, and from end hosts that are connected to the switches. There is ongoing work

[10, 11, 12, 13, 14, 15, 16, 17] on enhancing security in SDN but still there is need for significant research for addressing several security issues and developing robust solutions to deal with the attacks. Hence, the aim of this report is to present security architectures and services/techniques for detecting attacks on the switches for enhancing security in SDN data plane.

In the Technical Report 1, we have presented an a comprehensive attacker model for the switches. We have also presented data-plane device architecture. In this report, we propose a Switch Security Management Architecture (SSMA) for SDN Controllers to detect attacks on switches.

The report presents a comprehensive set of security requirements based on security flaws. These requirements create the base for the security architecture. The SSMA is a modular security architecture and we want to develop it as a platform-independent and directly deployable service. The SSMA utilises fine granular management policies to maintain the switch's security status and communication integrity. It uses TPM-based attestation and dynamic switch state validation to measure the switches' security and trust status. In addition, it has symmetric key management capabilities. Moreover, the switches can detect an attack (equipped with IDS) and send reports of their states. Furthermore, it supports security-aware switch state tagging. Such features empower the programmable network infrastructure to detect attacks on the OpenFlow switch, evaluate their status dynamically, and restore them to a secure state on demand. The proposed architecture components also ensure secure and trustworthy routing of flows and installation of flow rules. Finally, we develop a prototype of SSA for ONOS SDN Controller and present some performance results. We will also present general discussion on some of the related issues that can be addressed as part of future research.

The report is organised as follows. Section 2 presents an overview of the SDN architecture and lays out the security requirements based on Report 1. In Section 3 we propose the Switch Security Management Architecture. The architecture can detect and mitigate the attack on the SDN data plane. Here, we first present a high-level overview of the SSMA and then comprehensively describe all its components and functions. In the future, we want to develop this security architecture as an application. The final report will contain details about it. Section 4 presents some hypothetical scenarios where SSMA can help tremendously. These scenarios will explain the SSMA components' strengths and how they can detect and prevent some attacks. Section 5 concludes the report.

2. SDN OVERVIEW AND SECURITY REQUIREMENTS

In this Section, we provide an overview of the SDN Architecture and discuss the attacker model.

2.1 SDN Overview:

The SDN is a emerging network architecture that has been deployed to enable more agile and cost-effective networks. It is an architectural approach that optimizes and simplifies network operations by more closely binding interactions such as provisioning, messaging, and alarming among applications, network services and devices, whether they be real or virtualized [18]. The key features of SDN are [18]:

- The separation of the network control plane (SDN Controller) from the data plane (Southbound Interface).
- A logically centralized controller communicating with the data plane over open and standardized interfaces and protocols (OpenFlow).
- The control applications (Northbound Interface/Programmable API interface) running on top of the Controller.

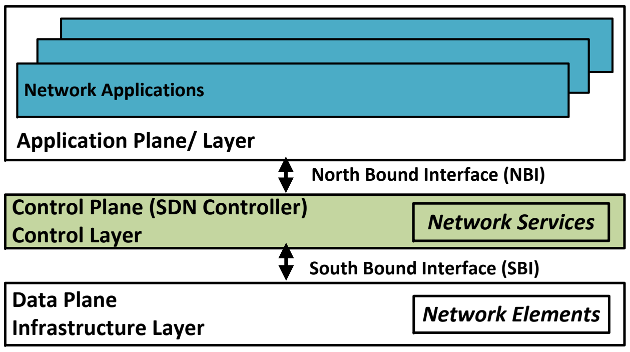

Figure 2.1: Software-Defined Network Architecture

Figure 2.1 shows the SDN architecture proposed by Open Networking Foundation [19]. The SDN architecture differs from legacy solutions by building networks in three abstractions/planes/layers.

Data Plane/Infrastructure Layer. The Data Plane or Infrastructure Layer acts as the foundation for an SDN architecture. This plane consists of both physical and virtual network devices such as switches, routers, wireless access points. These devices implement the OpenFlow protocol to maintain communication with the Control Plane and also implement standard methods for forwarding traffic using flow rules.

Control Plane/Layer. The Control Plane acts as the brain for the whole networking ecosystem. The Control Plane is decoupled from the underlying Data Plane infrastructure to provide a single centralised view of the entire network. The Control Plane utilises OpenFlow to communicate with the Data Plane devices. The SDN Controller is a logically separate entity responsible for receiving instructions from the application layer and enforcing them over the Data Plane devices. It is also responsible for gathering information about network devices, events and statistics and sharing this data with the network applications running in the Application Plane.

Application Plane/Layer. The Application Plane consists of network services, orchestration tools, and business applications that interact with the control layer. For instance, an application to monitor malicious activity in Data Plane devices. These applications leverage open interfaces to communicate with the Control Plane and the network state.

API interfaces. The application programming interfaces (APIs) are an alternate way to provide the abstraction necessary for SDN, along with a highly programmable infrastructure. APIs provide a channel by which instructions can be sent to a device to program it. In SDN, APIs are called "Northbound"or "Southbound", depending on where they function in the architecture. APIs that reside on a Controller and are used by applications to send instructions to the Controller are northbound because the communication takes place north of the Controller. Southbound APIs reside on network devices such as switches. These are used by the SDN Controller to provision the network, with the communication taking place south of the Controller.

2.2 Security Requirements

In report 1, we have already explained several threats to SDN infrastructure. Here, we will formulate the security requirements based on the previously discussed threats.

- (R1): In SDN, a vital component of the network is the network flows being generated from different users and devices. One of the major targets of an adversary is to capture these flows, reroute or forge malicious flows. Security requirements can vary for different types of flows in a network. There can be security constraints on different flow parameters such as the time of flow, the location of devices that are generating or receiving the flows, and delay and bandwidth requirements for the flows. For example, critical applications may have stringent requirements for the delivery of messages. Sensitive applications may require that the traffic is transferred through a secure channel involving secure switches. Hence, there is a need to ensure that these specific flow requirements are provisioned correctly in a multi-domain SDN environment. This requires the ability to handle flow specific security characteristics in both intra- and inter-domain communications in SDNs.

- (R2): In SDN architecture, OpenFlow switches/ forwarding devices/ dataplane devices are loosely connected to the Controller. Also, due to the SDN architecture, they are non-intelligent and might be connected to a malicious end host. Thus, the main target of a malicious end host are the OpenFlow devices. There is a strong possibility that these devices can be compromised, and an adversary can launch further attacks to the network Controller. Hence, OpenFlow switches or the SDN controller requires some security mechanism to secure and to check the security status of the switches.

- (R3): SDN can help deal with complexities in the current networks such as use of multiple technologies (wired, WLAN, virtualisation), heterogeneous devices (desktop, servers, laptops, mobile) and their mobility. However, we have already discussed how malicious end hosts are able to exploit the SDN operations and generate attacks. Hence, there is a need for techniques to enhance the security of these SDN operations.

- (R4): There is a need for effective detection of attacks. It is challenging to differentiate between the malicious and legitimate flow requests just by monitoring the network traffic. Attack detection can be made more effective by analysing the state of the end hosts.

- (R5): Intercommunication links between different OpenFlow switches are not secure and protected. An adversary can tap/spoof/reroute the connection between the switching fabric. There is a need for a security mechanism to monitor and prevent such malicious behaviour in SDN domain.

- (R6): The SDN Controller maintains a secure channel with each OpenFlow switch. To secure this channel SDN architecture uses TLS. Most of the Controllers (commercial/open-source) disable TLS for performance issues. On the other hand, an adversary in an SDN network can easily scan SDN domains for valuable information about the Controller secure channel (i.e., IP, ports, etc.). Once an adversary discovers the secure channel details, it can masquerade that to connect to the Controller. Thus, SDN requires a mechanism to securely connect/monitor/maintain the OpenFlow switches.

- (R7): The SDN architecture consists of a huge number of internal and external operations. Internal operations consist of controlling SDN core modules while external operations control the OpenFlow switches. There is a need to secure normal SDN operations, as the attackers can exploit the weaknesses in the SDN operations to generate various attacks. For example, currently, Controllers do not validate the flow requests before establishing the routes to enable communication between the end hosts. Another example is when the attacks, such as the spread of worms in traditional networks, can also happen in SDNs. A malicious host scans for random addresses to find vulnerable machines and spreads the attacks, establishing routes to destination hosts.

- (R8): The main foundation of the Internet lies in inter-domain communication. For SDN, inter-domain communication is a new area. Hence, it is vulnerable to both legacy inter-domain attacks and new SDN specific inter-domain attacks. For instance, topology poisoning attacks on the OpenFlow SDN edge routers can provide malicious adversary access to different domain traffic. Hence it makes all the domain user and the domain SDN Controller susceptible to attacks. Thus, there is a clear need for a security mechanism to secure the inter-domain communication in SDN.

- (R9): As the Controller has the visibility of its network domain topology and devices, there is a potential to develop secure northbound applications making use of the information available at the Controller for achieving end-to-end security within a domain.

- (R10): In a multi-domain situation, the end hosts are involved in communication with hosts connected to different networks. Hence, there is a need to develop techniques for achieving end-to-end security over multiple domains. However, as the Controller has visibility only over its domain, achieving secure inter-domain communications requires secure co-operation between the Controllers in different domains.

- (R11): There is a need for efficiency when it comes to detecting and preventing attacks. If all the traffic has to be forwarded to the Controller for monitoring then this will incur significant delays to communication and also cause congestion of the Control Plane. Such monitoring can incur very high overheads for monitoring the communication between two virtual machines that are hosted on the same server. In practice, virtual machines hosted on the same physical server make use of virtual links for communication and the traffic is never placed on the physical network medium. If such traffic needs to be monitored by the SDN Controller, then the flow has to traverse through the physical network which significantly downgrades the advantages provided by the virtualisation technology.

- (R12): In SDN, the Controller core provides a platform for the third-party network applications to run. The third-party developers develop these network applications to access and use different network infrastructure resources, such as OpenFlow switches creating report about network use dynamically. These network applications with the help of the core modules can access network resources. Thus, different applications (third-party or same platform) accessing the network resources without any authentication or authorisation possesses a direct threat to the SDN Controller. For instance, when a company develops a network switch performance monitoring application, logically the application is supposed to monitor the health of any particular vendor switches. However, if it turns out to be a malicious one or the application was bugged with malware, then once deployed it will start to physically tamper the switch OS files and flow rules causing it to behave differently.

- (R13): The core SDN Controller modules can have loop-holes or bugs. The adversaries can exploit these bugs to create new SDN specific vulnerabilities. The action can lead to compromising some vital software modules in SDN core, such as flow-rule service module, network topology service module, etc. The network applications require a sand-boxing and standards so that the failure or compromise of one application does not affect the others or the whole network Controller operation.

- (R14): We need to ensure that the devices and mechanisms used for monitoring the end hosts are secure in the first place. If the attacker can access these monitoring devices, s/he can disable them or alter their configuration so that they cannot detect the attacks.

- (R15): The security policies that can be enforced at the devices depends on the ability of the device. For example, layer 2 devices such as access point and switches can only enforce security policies at the MAC layer, whereas layer 3 devices can enforce security policies at the MAC/IP layer. Hence the Security Architecture should be able to enforce security policies based on the abilities available in the switches.

3. SWITCH SECURITY MANAGEMENT ARCHITECTURE (SSMA)

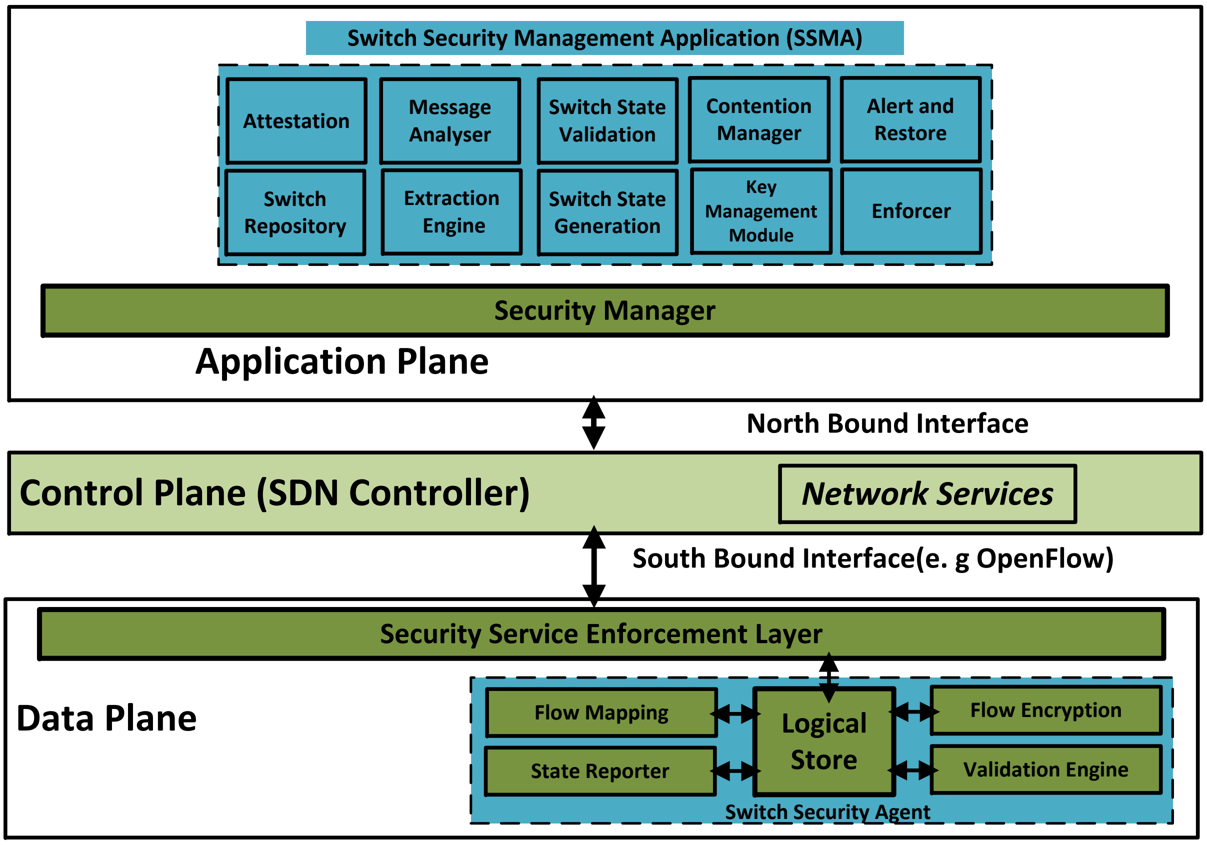

Figure 3.1: SDN Security Architecture

First, we will present Switch Security Management Architecture (SSMA) for intra-domain interactions. After that, we will demonstrate a high-level overview of the architecture and then describe each component in detail. Finally, we will explain different scenarios considering switch attack detection and secure communication route management.

Figure 3.1 shows the SSMA architecture for securing communication in an SDN switch communication. The proposed architecture can either form part of the SDN Controller or can run as a Security Application on top of the SDN Controller. We have designed and developed the SSMA as an Application running on top of an SDN Controller for flexibility reasons. SSMA is implemented in the northbound interface of the Controller; however the policies are enforced at the data plane layer. As SSMA is designed in a modular fashion, the components of SSMA can be implemented on a single host or distributed over multiple hosts. We have also developed OpenFlow switch agents to enforce the SSMA policies. Before describing the different modules of SSMA, we present a high-level overview of the overall architecture.

We assume that an SDN Controller can control each network infrastructure or an AS domain. Each Controller has its SSMA application with a Security Manager. The SSMA application has twelve main components: Security Manager (SM), a Switch Repository (SR), an Attestation Module (AM), Message Analyser (MA), a Switch State Validation module (SSV), Contention Manager (CM), Enforcer (E), Key management modules (KM), Extraction Engine (EE), Switch State Generation module (SSG), an Alert Restore (AR) module and Switch Security Agents (SSA). Each Controller maintains and updates a Topology Repository. The Topology Repository contains the network topology. The Security Manager in SSMA updates the Switch Repository. An SR is a collection of three repositories: i) Switch internal information; ii) Switch external information iii) Secure Route Policy repository. These repositories contain information and policy expressions that are used for switch state evaluation and secure trustworthy routing respectively. Here the policies are expressed using a simple language-based template described in Section 3.1.8 below.

As the name implies, the Security Manager manages every single operation of the security architecture. The security architecture operates in two phases, boot-time and run-time. Each phase uses different modules to achieve its respective goals. SM utilises the Attestation, switch state validation/ generation modules during the system boot-time or switch boot-time. At the same time, KM helps to set up the keys and secure channels for secure communication. Admins can set up secure policies during boot time as well. During the run-time, Extraction Engine is used to extract the incoming network traffic against the relevant security policies for that specific traffic. The MA can inspect the captured traffic. The Security Manager examines the extracted traffic and determines the security policies for the OpenFlow switches. This information is conveyed to the Enforcement

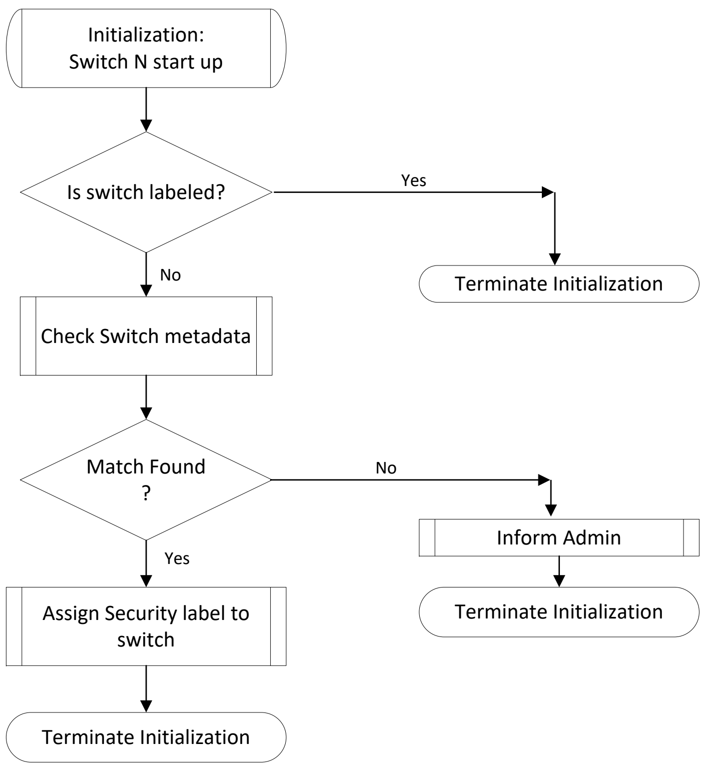

Figure 3.2: Switch labelling process by PM

Layer with the help of the Enforcer module. The Switch Security Agents then enforce the security services and mechanisms over data plane.

Section 3.1 describes the software modules in our Security Architecture. Section 3.1.8 describes the security policy specifications and Section 3.1 expresses the software modules of the Security Architecture. Note security policy specifications involve the design of a language. Hence, it is first described separately in Section 3.1.8 whereas

3.1 Components of SSMA

In this section, we explain different components of SDN Authorisation Policy based Security Architecture. Each component performs different functions, which we have explained in detail in this section.

Security Manager (SM)

Security Manager is the heart of the whole security architecture. We consider this as a layer on top of the SDN Controller. This approach provides flexibility in orchestrating the controller functions and services. The administrator makes use of the SM interface for accessing and configuring different components of the security architecture. It is responsible for all decision-making and instruction execution. The tasks performed by the PM are:

A. Switch initialisation;

B. Switch State generation and verification;

C. Switch attestation;

D. *packet_IN* request capture and analysis;

E. Send request to the Switch Store module;

F. State verification of the switch;

G. Security and Trust level assessment of the switch;

H. Check the Contention Manager for conflicts;

I. Key distribution and allocation;

J. Send Flow_mod() instructions;

SM tasks are divided into two phases. Phase one is *initialisation (A-C)*, which focuses on initialising switches, attesting them and adding security and trust labels, and the next phase is *processing (B- J)* consisting of processing of flows.

Phase 1: Initialization. When the OpenFlow switch starts up, it sends a "Hello"message to the Controller. The Controller, upon receiving this message, forwards it to the SM. The SM activates the attestation module to attest the OpenFlow switches. The SSG collects the switch information and generates the switch state. The SM forwards this to the Message Analyser to extract related information from the Hello message and assign a label accordingly with the help of Switch Store. The switch labels information is also updated to the Controller. The Controller maintains a complete view of all the active OpenFlow switches with switch security and trust labels with in the network. Figure 3.2 shows the switch labelling process.

Phase 2: Processing. The next part is focused on the processing of flows. SM receives the *packet_IN* request from the Controller and it uses Extraction Engine and Message

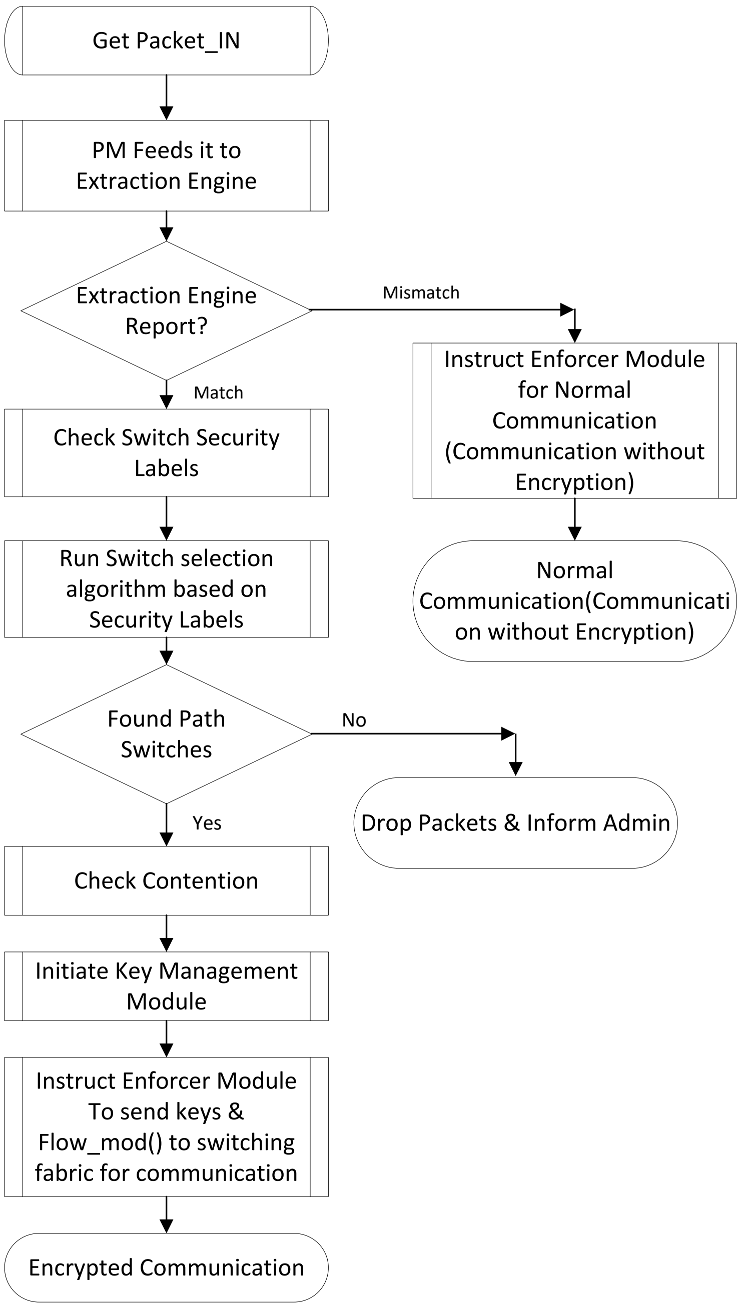
Get Packet_IN
PM Feeds it to Extraction Engine
Extraction Engine Report?
Mismatch
Match
Instruct Enforcer Module for Normal Communication (Communication without Encryption)
Normal Communication(Communication without Encryption)
Check Switch Security Labels
Run Switch selection algorithm based on Security Labels
Found Path Switches
No
Yes
Drop Packets & Inform Admin
Check Contention
Initiate Key Management Module
Instruct Enforcer Module To send keys & Flow_mod() to switching fabric for communication
Encrypted Communication

Figure 3.3: Policy Manager processing flow diagram

Table 3.1: Switch Label selection algorithm

Step 1	Get OpenFlow Switch **Label and ID**.
Step 2	Set Source Switch label as the **Reference Security Label**.
Step 3	Check Path Switches against the Reference Security Label.
Step 4	Select if **Path Switch i >= Reference Switch Label**
Step 5	Repeat step 2, 3 and 4 until destination switch found.
Step 6	Pass switch ID to next module.

Analyser to extract and analyse the message. During the flow attribute extraction phase, the Extraction Engine will extract the important fields necessary to match with the policy expressions policy terms (stored in Switch Store), such as **Source IP, Destination IP, Device MAC and Services etc**. The extracted information is used to query the Switch Store to determine the relevant Policy Expression. Now the SM becomes aware of the match/mismatch cases. The SM checks for the secure OpenFlow switch labels. It uses the switch security/trust label search algorithm described in Table 3.1. At first, it will take the security/trust level of the Source OpenFlow switch as the reference security/trust label. To establish a path between a source to a destination, it will only choose those OpenFlow Switches whose Security Label is equal and above that reference security label. The SM also listens to the Contention Manager before taking action. Once it gets the switch paths, SM then instructs the Key Management module to generate keys for source and destination switches. Finally, SM instructs the Enforcer to update the Controller and Switch Security Agents for setting the flow tables of appropriate switches. Figure 3.3 shows the flow diagram of SM. If no secure switches are found, it informs the administrator after dropping the packets.

3.1.1 Extraction Engine

Extraction Engine performs three fundamental functions. They are i) extracting the flow attributes, ii) querying the Switch Store, and iii) sending comparison reports to the SM. Firstly, the Extraction Engine helps SM to analyse *packet_IN* requests. First, it extracts the necessary flow attributes by processing the header information. Then, it stores them in a temporary data structure. Next, it runs a query for matching attributes in the Switch Store. Finally, it sends a report to SM for matching Policy Expressions and actions.

3.1.2 Message Analyser

This component is used for analysing, validating and storing (in the switch store) the communication between the Controller and the switches. In addition, it is used for extracting messages related to the configuration of the switches.

OpenFlow supports different types of messages [20] between the Controller and the Switches, such as Hello, Echo request, and Error. The message analyser is used for extracting the messages related to the flow rule configuration in the switches such as flow_mod, group_mod, table_mod from the Controller. Then the messages are validated to prevent conflicts in the flow rules enforced in the switches. If a new flow rule conflicts with the existing flow rules in the switch, then the flow rule with the highest priority is configured in the switch. One of the challenges is that the messages sent by the Controller can be executed in arbitrary order by the switch. Hence, we use Barrierreq and Barrier-Res to achieve synchronisation between the controller's flow rule configuration messages and the switch executed on the switch.

In this SSMA architecture, the Message Analyser also provided important key fragments of a message. For instance, header information of the OpenFlow message. Other components of the SSMA, such as SSG, can also use this information.

3.1.3 Contention Manager

The SSMA architecture deals with two types of conflicts: i) Security policy contention and ii) Flow rule conflicts.

The SM uses Contention Manager to check conflict and validate policies before enforcing them in the switching fabric. The Policy Expressions suggested by the Extraction Engine are forwarded to the SM. The SM queries CM for possible conflicts with already enforced policies. The CM checks the Security Architecture log to determine the policies/flows that are currently enforced in the switching fabric. Finally, CM checks for conflict between previously installed flows in OpenFlow switches to the new flow installation request by the SM.

Each OpenFlow switch stores the flow rules for specific communication. Sometimes communication glitches or switch OS malfunctions can cause flow rule conflicts. SSMA can resolve such conflicts as well. The CM with the help of Switch Store, can cross-validate the flow rules stored in each OpenFlow switches. This helps SSMA to resolve the flow rule conflicts.

3.1.4 Key Management Module

The Key management modules help in two aspects. First, it helps to create an encrypted TLS channel between the Controller and the OpenFlow switch. Secondly, it helps to generate symmetric keys for the specific switch for secure flow routing. This symmetric key encryption ensures the confidentiality of the flow content. Here, the SM will request Key Management module to issue symmetric keys that can be used by the source and destination OpenFlow switches for communication. The OpenFlow switch connected to the source host uses the symmetric key for encrypting the traffic. The OpenFlow switch connected to the destination host uses the same key for decrypting the traffic and forwarding it to the destination host. We present the schemes here in detail.

Secure Flow Installation

SDN Controller uses OpenFlow protocol to send instructions to the switches. Th OpenFlow protocol uses Transport Layer Security (TLS) to secure the OpenFlow communication [21], [22], [23]. However, due to performance issues, TLS is disabled by most of the Controllers available in the market [23]. Therefore, we have extended OpenFlow by properly utilizing TLS. In our case, TLS helps to install the flow rules into OpenFlow switches securely. In this section, we present the TLS extension and explain how it is used to install flow rules into the OpenFlow switches securely. It has five major phases. We have illustrated the protocol in Figure 3.4 and Table 3.2.

Phase1: Setup Security Capabilities

In this phase, both the Controller and switch agree on the cipher suite they will be using during the handshake process. At first switch issues a *Client Hello* message, which

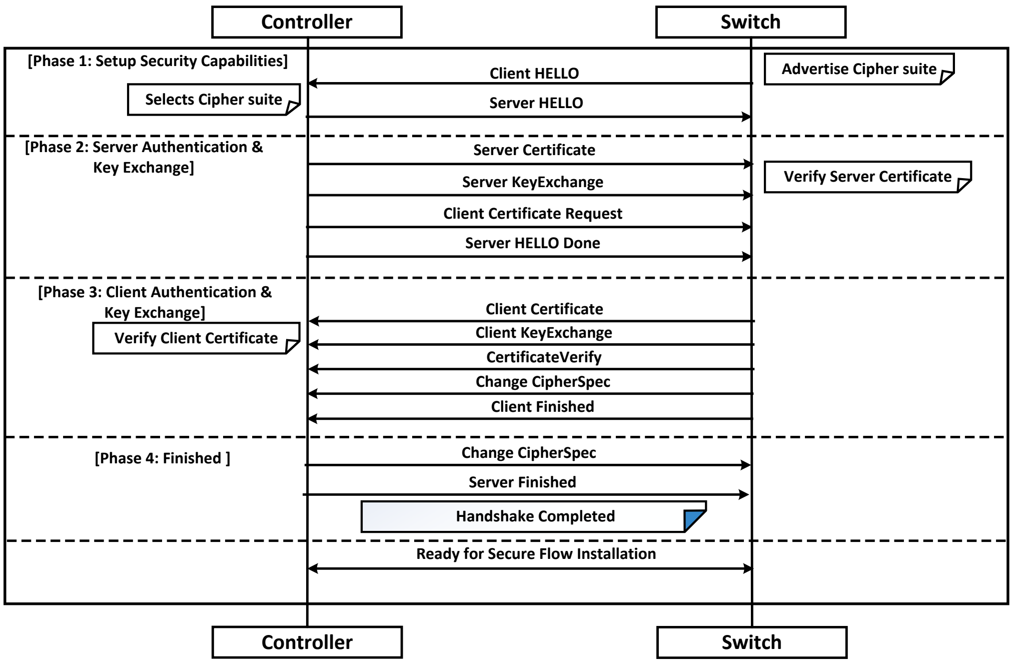

Figure 3.4: TLS for secure flow installation

contains a nonce (a random number), available cipher suits, session ID, and available compression method. In response, Controller sends a *Server Hello* message containing, a server nonce (replays the switch selected nonce), previous session ID, selected cipher suite and compression method for the handshake process. It is represented as STEP 1 in Table3.2.

Phase2: Server Authentication and Key Exchange

In this phase, Controller generates it's public/private keys and responds to the OpenFlow switch by sending its certificate. The Controller creates a hash of the information and signs its own private key. The Controller certificate contains the Controller public key. Hence, the switch receives the public key of the Controller. The OpenFlow switch verifies the Controller certificate. The exchange happens during the key exchange phase. The type of key and size depends on the previously negotiated encryption method. The Controller also issues a request to send the OpenFlow switch certificate. The message contains a new nonce generated by the Controller. Finally, the Controller finishes the phase by sending a *Server Hello Done* message. It is represented as STEP 2 and 3 in Table3.2.

Phase3: Switch Authentication and Key Exchange

Verification of switch certificate is optional, and, in most cases, it was overlooked while implementing TLS in OpenFlow [24]. In this phase, switch responds with its certificate to the Controller. The OpenFlow switch also sends *certificate verify* message to ensure that the switch is legitimate and has its private key. This message is signed by the private key of the switch. It is represented as STEP 4 in Table 3.2.

The Controller generates a premaster secret and encrypts it with the switches public key. Finally, it creates a hash of it and signs it with its private key. It is represented as STEP 5 in Table 3.2.

Phase 4: Finished

Both parties use the premaster secret and nonces to calculate the master key (a symmetric key). It is represented as STEP 6 in Table 3.2.

Phase 5: Secure Flow Installation

Switch acknowledges the Controller about the reception of the premaster secret and the Controller responds by acknowledging the reception. It is represented as STEP 7 and 8 in Table 3.2. Thus, the Controller uses the master key to install the flow rules securely.

Notations:

- Version information of switch and Controller is represented as V_{SW_i} and V_C respectively.
- Cipher suite advertisement by the switch is presented as $Suite_{SW_i}$.
- Controller has a pair of public and private key $(K_{PU(C)}/K_{PR(C)})$.
- Cipher suite selected by the Controller is $Suite_C$.
- Nonce generated by the switch n_{SW_i}.
- Nonce generated by the switch n_C, n'_C.
- KM_{C-SW_i} is the master key for any particular Controller (C) to switch (SW_i) communication.
- P_{MS} is pre-master secret.

- Controllers certificate is represented by $Cert_C$.
- Switches certificate is represented by $Cert_{SW_i}$.
- Hash function is represented as $h(.)$.
- A function to calculate the master key is represented as $f(.)$, which is a combination of MD5 and SHA.
- A certificate request is represented as req.
- An acknowledgement is represented as ack.

Table 3.2: Modified TLS for Secure flow installation

STEP 1: $SW_i \rightarrow C :< SW_i, V_{SW_i}, Suite_{SW_i}, n_{SW_i} >$

STEP 2: $C \rightarrow SW_i :< V_C, Suite_C, n_{SW_i}, Cert_C > [h(V_C, Suite_C, n_{SW_i})]_{K_{PR(C)}}$

STEP 3: $C \rightarrow SW_i :< SW_i, n_C, req > [h(SW_i, n_C, req)]_{K_{PR(C)}}$

STEP 4: $SW_i \rightarrow C :< C, n_C, Cert_{SW_i} > [h(C, n_C)]_{K_{PR(SW_i)}}$

STEP 5: $C \rightarrow SW_i :< SW_i, n'_C, [P_{MS}]_{K_{PU(SW_i)}} > [h(SW_i, n'_C, [P_{MS}]_{K_{PU(SW_i)}})]_{K_{PR(C)}}$

STEP 6: Both parties calculate $f(n_{SW_i}, n_C, n'_C, P_{MS})$ to get KM_{C-SW_i}

STEP 7: $SW_i \rightarrow C :< C, n'_C, ack > [h(C, n'_C, ack)]_{KM_{C-SW_i}}$

STEP 8: $C \rightarrow SW_i :< SW_i, n'_C + 1, ack' > [h(SW_i, n'_C + 1, ack')]_{KM_{C-SW_i}}$

Securing Flow Information

The flow information is stored in the packet payload. In the current SDN network, by default, none of the Controller's provides an on-demand confidentiality service for user flows. Our SSMA provides confidentiality service to the users or devices if required. In our policy language, we have a special term that signifies whether the communicating entities require confidentiality services. Based on this, the following a symmetric key-based scheme. Here, we will explain the scheme. The SSMA architecture provides confidentiality service to the communicating entities, in this case, the hosts. We have policy

expressions that specify which communication entities would be provided with confidentiality services. The Controller generates symmetric keys for each communicating entity according to the policy expressions. The key convention used for the policy expression is $K_{S(PolicyExpressionID)}$. The switches use these symmetric keys to encrypt the flow payload during communication. The switch's agents help in the payload's encryption and decryption process. The distribution or sharing the symmetric key with the OpenFlow switches present within a path other than the source and destination switch is entirely dependent upon the Controller and the policy expressions present in SSMA. In exceptional cases, the intermediary OpenFlow switches can request these symmetric keys for any particular flow. We have introduced new messages in the OpenFlow protocol for this.
The scheme also focuses on verifying the switch which generates the flow. Broadly, every intermediary switch in a routing path can justify which OpenFlow switch is sending a particular flow. This improves the authenticity of the routing path and OpenFlow switches can judge forged routing requests and flows.
We will first present the procedure and then present an example.

3.1.5 Procedure Assumptions:

The following procedure assumptions are made:

- Each OpenFlow switch has a pair of public/private keys.
- Each OpenFlow switch during the bootup time receives a certificate from the Controller.

Procedure

In this procedure, the verification process starts from the source switch and finishes at the destination switch. The OpenFlow switch in which the flow-generating host is connected is called the source switch. The switch in which the flow-receiving device/host is connected is called the destination switch. In between the source and destination switch, all the other devices within a particular path/route is known as intermediary switches.
All the switches are equipped with similar components/agents, as discussed 3.2. The

agents can sign, generate, verify and store keys in the local caches. The users cannot participate in this procedure. After getting the flow from the host, each OpenFlow source switch checks the switch flow table. If there is no flow rule, it follows the normal SDN operation with SSMA. We consider that the flow rule is present and the communicating parties request confidentiality service in the policy expression. Then the source switch will modify the header information and send it to the intermediary switches. The modified header consists of some extra information apart from the usual IP header information: 1) time (the time this flow packet was created); 2) source and destination switch information, and 3) all of them are signed with the private key of the source switch. It also sends the hash of the source, destination, time of source packet generation and symmetric key for the specific policy expression, signed with the public key of the source switch. This information is piggybacked with all the packets generated by the intermediary switches. When the intermediary switch receives it, it uses the source switch's public key to verify the flow packet's authenticity. Note that the payload information is encrypted with the policy expression key.

Before sending it to the next intermediary switch, the current intermediary switch constructs a similar type of flow, including his information (destination, time, and signs with his private key). The process is repeated until it reaches the destination switch. The process with two intermediary switches can be mathematically represented as:

$$\begin{aligned}
S \rightarrow I1 :&< H, S, I1, D, t_s > [h(H, S, I1, D, t_s)]_{K_{PR(S)}}.[(S, D, t_s), X].[PL]_{K_{S(PE-i)}} \\
I1 \rightarrow I2 :&< H', I1, I2, D, t_{I1} > [h(H', I1, I2, D, t_{I1})]_{K_{PR(I1)}}.[(S, D, t_s), X].[PL]_{K_{S(PE-i)}} \\
I2 \rightarrow D :&< H'', I2, D, t_{I2} > [h(H'', I2, D, t_{I2})]_{K_{PR(I2)}}.[(S, D, t_s), X].[PL]_{K_{PU(PE-i)}} \\
&where, \\
&\quad X = [h(S, D, t_s, K_{S(PE-i)})]_{K_{PR(S)}}
\end{aligned} \tag{3.1}$$

Notations:

- H, H', H'' is the header information.

- t_a represents the packet construction time by the a switch.
- PL represents the payload.
- $h(.)$ represents the hash function.

3.1.6 Scenario

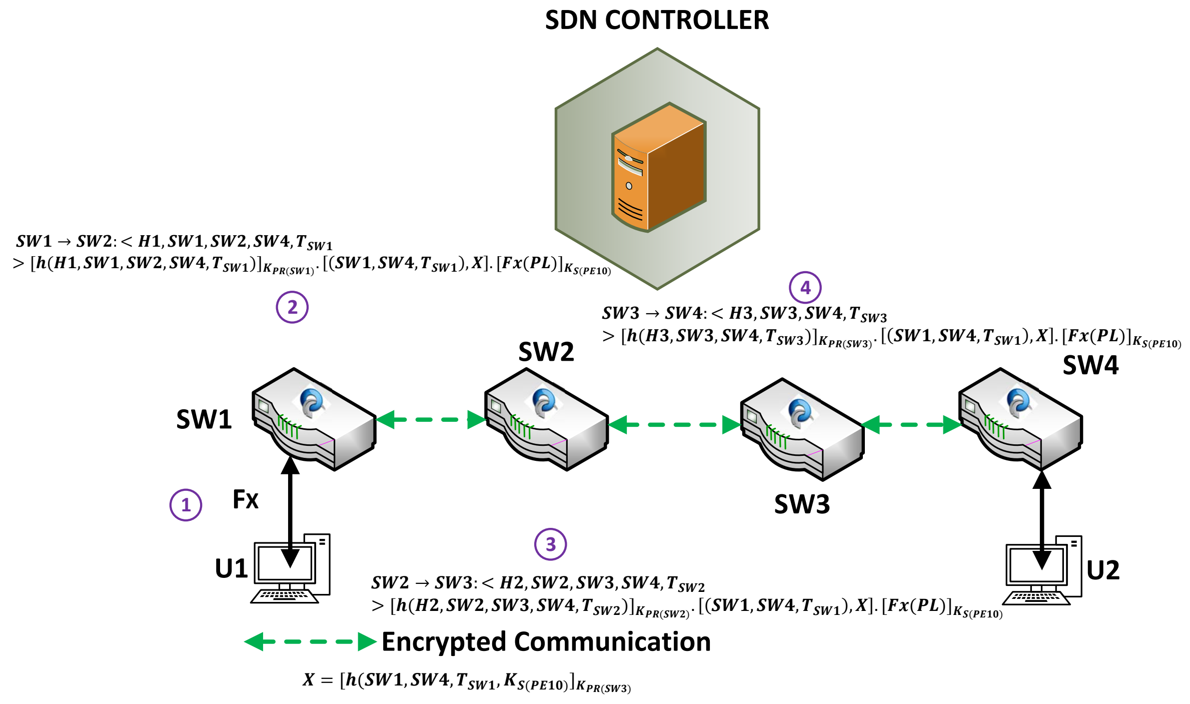
SDN CONTROLLER
$SW1 \rightarrow SW2 :< H1, SW1, SW2, SW4, T_{SW1}$
$> [h(H1, SW1, SW2, SW4, T_{SW1})]_{K_{PR(SW1)}}. [(SW1, SW4, T_{SW1}), X]. [Fx(PL)]_{K_{S(PE10)}}$
2
4
$SW3 \rightarrow SW4 :< H3, SW3, SW4, T_{SW3}$
$> [h(H3, SW3, SW4, T_{SW3})]_{K_{PR(SW3)}}. [(SW1, SW4, T_{SW1}), X]. [Fx(PL)]_{K_{S(PE10)}}$
SW2
SW4
SW1
1
Fx
SW3
3
U1
U2
$SW2 \rightarrow SW3 :< H2, SW2, SW3, SW4, T_{SW2}$
$> [h(H2, SW2, SW3, SW4, T_{SW2})]_{K_{PR(SW2)}}. [(SW1, SW4, T_{SW1}), X]. [Fx(PL)]_{K_{S(PE10)}}$
Encrypted Communication
$X = [h(SW1, SW4, T_{SW1}, K_{S(PE10)}]_{K_{PR(SW3)}}$

Figure 3.5: A scenario explaining end-to-end authentication procedure

In this scenario, we assume that $U1$ generates the flow F_X towards $U2$. Here, the source and destination switches are $SW1$ and $SW4$, respectively. $SW2$ and $SW3$ are the intermediary switches. Using Equation 3.1 we get:

(1) The $U1$ generates the flow F_X.

(2) The $SW1$, makes a packet for next intermediary switch $SW2$.

$$\begin{aligned} SW1 \rightarrow SW2 :< H1, SW1, SW2, SW4, T_{SW1} > \\ [h(H1, SW1, SW2, SW4, T_{SW1})]_{K_{PR(SW1)}}.[(SW1, SW4, T_{SW1}, X)] \\ .[Fx(PL)]_{K_{S(PE10)}} \end{aligned} \tag{3.2}$$

(3) The $SW2$ checks the authenticity of the sender, makes a new packet and sends it to the next intermediary $SW3$.

$$\begin{aligned} SW2 \rightarrow SW3 :&< H2, SW2, SW3, SW4, T_{SW2} > \\ & [h(H2, SW2, SW3, SW4, T_{SW2})]_{K_{PR(SW2)}}.[(SW1, SW4, T_{SW1}, X)] \\ & .[Fx(PL)]_{K_{S(PE10)}} \end{aligned} \tag{3.3}$$

(4) Similar process is adapted by $SW3$

$$\begin{aligned} SW3 \rightarrow SW4 :&< H3, SW3, SW4, T_{SW3} > \\ & [h(H3, SW3, SW4, T_{SW3})]_{K_{PR(SW3)}}.[(SW1, SW4, T_{SW1}, X)] \\ & .[Fx(PL)]_{K_{S(PE10)}} \end{aligned} \tag{3.4}$$

The whole process is illustrated in Figure 3.5. Throughout the process, the flow payload $Fx(PL)$ was encrypted with the $K_{S(PE10)}$ policy key.

3.1.7 Attestation

This module uses the TPM attestation between the Switches and the Controller to ensure that the switches are in a trusted state during boot time. In the attestation process, all hardware and software components in the trusted platform are measured using hash values at the time of boot and measurements are stored securely to prevent modification. When a third party presents an attestation request, the trusted platform responds with an attestation report. This response includes the measured hash values, and a set of expected hash values that are supported in the form of measurement certificates issued by trusted certifiers. The basic idea is that a match between the measured and expected values usually indicates that the components are in a known and trusted state.

In our approach, the attestation module on the SSMA requests an attestation report from the physical and virtual switches and validates the report from the trusted switches. Below is a sample policy for using the TPM attestation for validating the virtual switches implemented in the hypervisor.

Listing 3.1: TPM Attestation label

```
<?xml version="1.0" encoding="UTF−8" ?>
<vtpm_policy id="XXXX">
<vm id="yyyy" name="" location="vm image file location" server= "IP address">
<attestation method="TPM 1.2">
< verification  method="AIK">
<algorithm type="sha1">[AIK fingerprint]</algorithm>
</ verification>
< verification  method="x509">
<algorithm type="x509">[x509 certificate]</algorithm>
</ verification>
</attestation>
<authentication>
< verification  method="x509">
<algorithm type="x509">[x509 certificate]</algorithm>
</ verification>
</authentication>
</vm>
</vtpm_policy>
```

The Controller will send flow configuration messages to the switches only if there is a match between the measured and expected values for the switch. If there is any variation in the reported values compared to the expected value then the switch is considered to be compromised and an alert is raised to the administrator. However this process only ensures that the switches are in trusted state during boot time. There is possibility for the attackers to alter the flow rules in the switches during run time. Hence there is also a need to validate the flow rules enforced in the switches during run time for detecting the attacks.

3.1.8 Switch Store

The switch store preserves the switches' internal and external state information and secure routing policies. In this section, we will explain each state component and secure routing policy in detail.

Switch Internal State:

The switch store is used for maintaining fine granular details on the switch's internal functionality or features and storing all the communication between the Controller and switches. For instance, it is used for storing information such as vendor information, hardware information, memory, the number of interfaces, the maximum number of flow rules, timeout for the flow rules, the location of the switch in the network topology and the flow rules configured by the Controller. This type of information is considered as switch internal information. Such information helps to generate a trusted report on the expected state of the switch.
When a new switch is deployed in the network, the switch details, such as the number of flow tables and flow rules supported by the switch and buffer capacity for storing the messages or packets, can be manually entered into the database by the administrator. We can also use the FeatureReq message to determine the switch's hardware or software resources and functionality. After a secure channel (TLS) is established between the switch and Controller, the SSMA will send a FeatureReq to the switch over the transport channel. The feature request consists of just an OpenFlow header message, with the FeatureReq value set in the type field. The switch will then respond to this request with its specification and functionalities. This basic sequence does not change across OF versions.

Switch External State:

As the Controller has visibility of the network topology and the communication behaviour of the switch, the Switch's External State captures the nearest neighbour connection points and the malicious interactions with the neighbour. Such information help evaluate the trust level and the state of the switch.

Security Policy Specifications

A key component of the security architecture is the specification of security policies that are to be enforced on the SDN infrastructure. The specified security policies are stored in the Switch Store. We have adopted a simple language-based approach to specify the security policies. We have chosen the policy-based routing syntax defined in RFC1102 [25] as the basis for our security policy specifications. Policies specify a switching path or paths packets must follow in the network and the conditions under which the packets follow these paths. In our language, we have Policy Expressions specifying various attributes associated with the flow and the entities in the SDN infrastructure. These include the following:

- Flow Attributes: Flow ID, sequence of packets associated with the flow, type of packets, security profile indicating the set of security services associated with the packets in the flow.
- Autonomous System Domain Attributes: AS identities such as source AS and destination AS identities (*AS Domain ID*), sub-net address space (SRC_{SUB} for source and DST_{SUB} for destination), identities of entry (SRC_{ENT}) and exit (DST_{EXT}) gateway/switch to AS, AS type (e.g. Commercial domain, Government domain) (SRC_{Type} and DST_{Type}) and security label associated with the AS (SRC_{SL} for source and DST_{SL} for destination).
- Switch Attributes: Identities of the switches, security, trust label and the state of the switches.
- Host Attributes: Identities of hosts - source host IP & MAC (SRC_{IP} & SRC_{MAC}) and destination host IP & MAC (DST_{IP} & DST_{MAC}).
- Flow and Domain constraints: Flow constraints (*FlowCons*) and Domain constraints (*DomCons*) associated with a flow such as thresholds, attack signatures etc.

- Services - Services for which the Policy Expression applies.
- Time Validity - The period for which the Policy Expression is valid.
- State - Presents the state of the switches.
- Path (AS_{SEQ}) - In the case of intra-domain, it indicates a specific sequence of switches, whereas with inter-domain communications, it indicates the sequence of Autonomous Systems traversed by a flow.

The flow constraints are conditions that apply to a specific flow or a set of flows. For instance, a constraint may specify that the flow of packets of a specific type (e.g., video) should only go through a set of switches that can provide a certain bandwidth, or from a security point of view, or a trust reference state, a constraint could be that a flow should only go through AS domains that are at a particular security level. Domain constraints apply to all flows within a domain. They are used to specify domain-wide policies. For instance, a domain-wide security policy may specify that all flows should be protected for integrity as part of the security profile. These constraints are used as part of the actions associated with the Policy Expressions. Packet (PKT_{ATT}) and Time attributes (T^{PE_i}) can be integrated into the constraints. Here, flow attributes indicate attributes associated with the sequence of packets in the flow, such as the type of the incoming packets based on port numbers, thresholds, security services associated with the packets and attack signatures. Time attributes represent the duration time for which a particular Policy Expression is valid.

Alternatively, it is also possible to enforce specific paths by explicitly specifying the set of switches through which a flow must go through or a specific set of AS domains that should be traversed.

The policy language has wild cards in its syntax enabling the specification of policies that can apply to sets or groups of entities and services. When a Policy Expression is satisfied, then the associated action is performed, which could be as simple as allowing or denying the request. Hence, using these policy terms, one can specify different sets of Policy

Expressions to deal with a range of scenarios in intra-domain and inter-domain communication in a distributed SDN. An action can also have some attributes. For instance, the destination exit switch (DST_{EXT}) attribute associated with an action indicates the exit switch through which a flow should pass once a policy is satisfied. Hence, we can specify conditional, constraint, state-dependent, and obligation policies.
Each Policy Expression has an action associated with it. The *FlowCons* and *DomCons* impose certain conditions associated with the actions. For instance, if there is a flow constraint which requires that this specific flow of packets of a specific type should only go through a set of switches (e.g., that can provide a certain bandwidth), then the action requires that appropriate flow rules be dynamically configured into a set of switches enabling a path that satisfies this constraint. Similarly, from a security point of view, if there is a flow constraint that requires only AS domains that are at a particular security label be traversed by the traffic in question, then the action will require the selection of a path with appropriate AS domains. An action can also have some attributes. For instance, Destination Exit Switch (DST_{EXT}) attribute associated with an action indicates that the exit switch through which the traffic should exit for the policy to be satisfied.
In particular, the language can be used to specify policies that take into account the context associated with the resources and the devices. For instance, it can be used to specify protection policies that take into account the attributes of the devices through which the flow can occur or be displayed. For certain confidential information, the paths through which the packets are transferred and the devices/switches which can process them must have certain security attributes. Such policy expressions can be specified using simple Boolean algebra on the security labels. The policy engine evaluates the Boolean expression to determine whether the condition on the security labels is satisfied or not. The language can also be used to specify release policies associated with the end points through which the traffic can be released, requiring certain security attributes. These types of policies, namely protection and release policies, are common in the context of content based security, and are significant when it comes to the provision of SDN services. Using the policy terms mentioned above, a simplified Policy Expression template could

be as follows:

$PE_i^{AS_k} = < FlowID, SourceAS, DestAS, SourceHostIP, DestHostIP, SourceMAC, DestMAC,$
$User, FlowCons, DomCons, Services, Sec - Profile, State, Path >:< Actions >$

where i is the Policy Expression number and k is the AS ID. This is a generic Policy Expression for both intra and inter-domain. In the following sections, we will explain the use cases for both intra and inter-domain. For simplicity, we have omitted the AS ID notation in intra-domain expressions. Also, in intra-domain, *path* indicates a set of switches within the domain while in inter-domain *path* refers to a set of AS domains.

In general, we have a number of Policy Expressions stored in the SDN Policy Repository. Such a template enables us to specify a range of policies for different users (and hosts), from different locations, accessing different services using different devices following different paths. Later we will illustrate the use of such policy expressions in both intra- and inter-domain environments when we discuss the different scenarios in **??** and **??**.

3.1.9 Switch State Generation

This component is used for generating a trusted report on the expected state of the switches. The flow rules that are currently enforced in the switch depends on the available resources and configuration of the switches. One of the challenges in generating the expected state of the switches is that the switches support different functionality and the state of the switch can vary on different factors such as resources available at the switches. For instance, switches have very limited memory and can only accommodate a few rules in their flow tables. Therefore, when the flow table is full and new flow rules have to be inserted into the flow table, some of the existing rules must be removed from the switch based on different algorithms such as First-In-First-Out (FIFO) and least used. As already discussed, all the information relating to the available resources and configuration of the switch (internal and external) are maintained in the switch store component. Hence, the Switch State Generation component can generate the expected state of the switch by querying the switch store component. For example, consider that N is the maximum number of rules that can be stored in the switch memory and FIFO is used for storing the flow rules. In this case, the switch is expected to have rules that were configured from the last N flow_mod messages. Since SSMA has the information related to each switch, it can query the switch store repository to determine the previous N configuration messages from the Controller and generate a trusted report on the expected state of the switches.

3.1.10 Switch State Validation

SSV validates the flow rules that are active and enforced in the OpenFlow switches. Hence this module helps to detect flow-related attacks. The component requests the switches to report the flow rules currently enforced in the switches. When the switch responds with the switch_state_ report, it compares this report with the trusted state report generated by the Switch State Generation component. If there is no variation in the currently enforced flow rules compared to expected flow rules, then the switch is considered normal. On the other hand, if there is any variation in the presently enforced flow rules compared to expected flow rules, the switch is considered under attack.

3.1.11 Enforcer

An enforcement layer is added in each OpenFlow switch for the proper functioning of the SSMA. The Enforcer module acts as a bridge between Controller and OpenFlow switches Enforcement layer. First, it helps SM intercept *packet_IN* requests. Then, after intercepting the *packet_IN* request from the OpenFlow switches, it forwards them to the Extraction Engine. Finally, SM starts to process the header information with the help of the Extraction Engine.

This module also helps SM to enforce flow rules in OpenFlow switches by issuing Flow_mod() messages. The KM uses this module to distribute the keys for source/destination switches. The keys are only distributed among the switches participating in a particular communication.

3.1.12 Alert and Restore

This component generates an alert to the security administrator when attacks are detected on the switches. This component also helps in generating alerts during system-level malfunctioning. For instance, switch state validation failure due to physical disruption and attestation failures due to TPM to hardware compatibility etc. The administrator can conduct an offline analysis to determine the event's seriousness and decide to either isolate the victim switch from the network or restore the state of the switch with the expected state report generated by the Switch State Generation module.

3.2 Switch Agents:

As shown in Figure 3.1, Flow Mapping (FM), Logical Store (LS), Validation Engine (VE) and Flow Encryption (FE) are components of the Switch Agent. The FM is used for detecting the applications running in the end host and fine granular mapping of the flows to specific applications. The LS component captures the end host-specific knowledge. For instance, LS has information such as the operating system, applications and resources allocated to the end host. The LS also enables auditing of end host transactions. The VE is used for detecting and preventing the attacks by validating the traffic and applications in the end hosts. FE is used for securing the communication between the end hosts. Now we present a detailed discussion of these components.

Flow Mapping

In the current networks, it is not an easy task for the victims to determine malicious networks or end host that is generating the attack traffic. One of the main challenges in the current networks is that the attackers can generate attacks with the spoofed source address. Furthermore, the usage of private addressing and techniques such as port address translation and network address translation are used to cope with the shortage of IP addresses make it extremely challenging to determine the malicious end host that is generating the attack traffic.

The FM receives the flows originating from the end host. For each new flow request, the FM extracts the source port information from the packet headers and makes use of the end host state reports to determine the specific application associated with the port and stores them in LS. When attacks are detected at later stages, SSMA can query the Switch Store to detect the malicious application and selectively isolate the malicious application. The application report can be generated by placing the FM component in the end host or VMM or network. Let us consider these options in detail. The FM is placed in the end host, and VMM has access to the end hosts' internal state. Hence they can detect all the applications running in the end host. The FMs placed in the network do not have access to the internal state of the end hosts. So they detect the applications by analysing the incoming and outgoing traffic from the end host. Hence they can detect only the applications with active flows in the end host.

The design choice of placing the FM, in the end host can lead to the compromise of FM since the attacker who has exploited the vulnerability in the end host can also access the FM and tamper the reports. Hence the reports generated by the FM placed in the end host cannot be trusted. However, the attackers who have compromised the end hosts do not have access to the FMs placed in the VMM and network. Hence the FMs placed in the VMM and network are more secure and the reports generated them can be trusted. So we have opted for the placement of FM in VMM and network (physical switches).

The FM makes use of LibVMI API interface [26] to obtain a list of the processes (`Pro_List`) running in the VM. The method is a standard way to fetch the application details for any OS running in the VM. It is similar to the report generated by the task manager in the Windows or top in Linux. The main difference is that the report is generated by a component residing within the VMM instead of being inside the VM. Hence, we take this report to be more trusted.

3.2.1 Logical Store (LS)

This component is used to store specific information related to the end hosts to detect attacks. The information that can be stored in this component depends on the resources

available at the switches. For instance, LS component is used for storing the application reports generated by the FM, attack signatures, whitelists, blacklists, behaviour profiles of the end hosts and storing all the incoming and outgoing traffic from the end hosts to enable auditing of flows.
We have captured specific information for different services (such as DNS, Web server and database server) and client machines with Windows and Linux OS that are commonly used in the SDNs. For example, there are 63 default processes running in the case of clean state installation of Apache web server and MySQL running on a Linux server with Ubuntu 3.5.0-23 kernel. The process apache2, which is located on the path (/usr/sbin/apache2) and runs under root UID is responsible for handling the client requests. Similarly, a Windows XP machine with default configuration will check for updates at 3:00 am everyday. If Sophos anti-virus is installed in the end hosts, the process alupdate.exe is dynamically invoked every 10 minutes to check for updates of attack signatures from the remote Sophos server. Such information can be used for specification of fine-tuned security policies for each end hosts. For example, the process-level information captured from a clean state installation of the Apache web server can be used as a whitelist for validating the state of end hosts during run time.

3.2.2 Validation Engine (VE)

This component is used for detecting attacks by validating the traffic and applications in the end host. As LS has specific information for each end host, the is able to specify finely-tuned security policies for each end host; e.g. signatures selected depending on the OS and applications running in the end host. Different techniques such as state validation of the end hosts, input/output traffic validation using whitelists and blacklists, signature-based and anomaly-based traffic filtering, are used to detect attacks. State Validation (SV) and Traffic Validation (TV) are the two important sub-components of the VE. The SV subcomponent is used for validating the state reports generated by the FM component and detecting the attacks. Similarly, the TV subcomponent is used for validating the traffic generated by the end hosts and detecting the attacks. Let us consider these components in detail:

State Validation: Let us consider how state validation can help to detect and isolate malicious VM's from targeting attacks on other devices (including switches, controller and other end hosts) in SDN networks. For each new flow request, SV first invokes utilities within guest OS such as `ps` (for UNIX like VM) or `tasklist.exe` or `ps.exe` (for Windows VM) to generate the process list VM_report. If the attacker has compromised the VM, then the `VM_Report` may not provide the actual list of applications or processes running in the VM. For example, the attacker could have disabled the security-critical processes in the VM or installed new malicious processes which are not listed in the `VM_Report`. Now SV makes use of the `FM_Report` to detect if the state of VM is suspicious. SV first checks for the presence of security-critical processes in the `FM_Report`. If the `FM_Report` does not include security-critical processes, then the VM is considered to be compromised. If the security-critical processes are found to be running, then it compares the `VM_Report` with `FM_Report`. If there is any variation of processes listed in these reports, then the VM is considered to be compromised.

If a VM fails to pass the SV security check, it is assumed to be compromised. Hence the flow request is dropped, the VM is isolated from other hosts in the network and an alert is generated to the SA. If the VM passes the SV security check, then the VM traffic is validated with the TV subcomponent.

Traffic Validation: This component is used for validating the traffic from all the end hosts (physical and virtual) and detecting the attacks. It makes use of source address validation, signature and anomaly-based techniques for detecting the attacks. Validating the source address prevents end hosts from injecting malicious messages to destabilise the network or generating attack traffic on any other hosts with spoofed identity. Verifying the source address ensures that the SDN Controller can only receive the new flow requests from a valid end host in the domain. Hence the switches and the SDN controller are protected from malicious flow request floods with a spoofed source address. However, the switches and the SDN Controller are still vulnerable to the malicious flow requests with a valid source address. The signature and anomaly-based techniques help to deal with

the attacks that are generated with the correct source address.
We have developed software modules for implementing DPI-based validation in all the OpenFlow switches. The software modules are used to analyse the payload and apply ACL to drop the malicious packets. ACL rule matching is done using Regular Expressions. GNU C regex cite is used for this. Also, Open vSwitch is constructed mostly using C, which makes GNU C regex easily adaptable to the environment. To do the comparison between the ACL rule and payload information, rules are stored in LS and then a regular expression string is constructed with each ACL rule. Finally, created regex strings are compared to the payload information. TV accepts the ACL rules sent by the SSMA and stores them in an array. When the packet flow happens, SSMA pull up the ACL rules and creates regex strings for them. These regexes are matched against the packet payload for similarity. If a match is found, TV drops the flow and raises an alert to the SSMA. The anomaly detection uses static thresholds in the initial stages to prevent flooding attacks from the malicious end host. The threshold for each host that is connected to the switch is determined by the number of hosts connected to the switch and the threshold for each switch is determined by the number of switches connected to the controller. Let 'CC' be the total capacity of Controller that is connected to 'X' number of switches. Similarly, let 'CS' be the capacity of the switch and 'Y' is the number of hosts connected to each switch. Then threshold 'TS' for each switch is determined by $TS = CC/X$ and threshold for each end host is determined by $Th = TS/Y$. For example, consider the case of a single controller that is connected to 10 switches which are further connected to 10 end hosts. If the Controller is capable of processing 1000 requests per second then each switch is restricted to sending $1000/10 = 100$ requests per second and each end host is restricted to sending $100/10 = 10$ requests per second. Also, note that these threshold policies can vary depending on the number of running instances of the Controller. For instance, the controller services can be hosted on multiple machines during peak hours. Hence the thresholds will be high for such cases. Traffic logs captured at LS are then used for training the TV to capture the behaviour of the network. We use machine learning techniques for capturing the behaviour of the traffic in the network and detecting the attacks. We

have used RF classifier since it does not require extensive training [27] and can fit with large databases very well. It is also not very sensitive to input parameters like SVM. The trained classifier is stored as a baseline detector for matching traffic behaviour. The trained profiles are stored in form of a decision model in LS for each monitored machine. In testing time, the TV uses these pre-compiled profiles for validating the behaviour of the end hosts and detecting the attacks.

3.2.3 Flow Encryption (FE)

This component is used for securing the communication between the end hosts. If the flow security policy mandates secure communication between the end hosts, the Key Management module in SSMA will generate symmetric keys for securing the flows and distribute the key to the related SA.

Many of the critical national infrastructures that were historically implemented as physically separate systems are increasingly using the Internet to minimise operating costs. Often critical infrastructures have some legacy systems in the networks which do not support any security functionality. Hence security administrators can make use of this component for securing the communication between the hosts. In this case, if the new flow is destined to the services or devices that do not have any security functionality, then the SSMA can enforce policy to encrypt the flow at the SA that is connected to the source host and decrypt the flow at the SA that is connected to the destination host.

3.2.4 State Reporter

The state reporter reports the state of the OpenFlow switches to SSMA. First it collects all the OpenFlow switch's internal and external state information. The other modules of the Switch Agents helps in this process. After that, it binds them into an information package and prepares an OpenFlow message. We are using OpenFlow experimenter message to send our custom state report to the SSMA. Finally, the switch agent sends the report.

3.3 Operation:

In this section, we will explain the operation of SSMA. The operation is divided into two phases: i) the boot phase and ii) the runtime phase.
We assume that the hardware platform is trusted and the operating system on which the SDN Controller is installed is trusted. During the SDN Controller boot time, the controller starts up and loads all the necessary core application modules and looks for the OpenFlow switches. The OpenFlow switches boot up and look for the SDN Controllers in a specific IP address.
The TPM measures all the hardware and software components of the switch during boot time using hashes and stores them securely. The core root of trust module ensures that the switch is in a trusted state during boot time. For instance, if there were any unauthorised changes to the switch software, it would prevent the startup of the switch. When the switch startup is successful, it sends OFTP_Hello message to the Controller which is subsequently forwarded to the SSMA.
The SSMA initiates the TPM attestation process to validate the boot time state of the switches. The Attestation module helps in this process. The SSMA permits the Controller's configuration of flow rules on the switches only if the TPM attestation of the switch is successful. Furthermore, SSMA validates the flow rule configuration messages for conflicts and stores all the communication between the Controller and switches. If there are any conflicts with the flowrule configuration messages, then the flowrules with higher priority are configured in the switch. The Contention Manager looks for such conflicts.
If the TPM attestation of the switch is not successful, then SSMA raises an alert to the SDN administrator. The Alert and Restore module helps to raise such an alert. This process ensures that only the switches that are found to be trusted during boot time are used for forwarding packets in SDN network. In addition, the State Generation Module generates the security state of the switch based on the internal and external state (as explained previously). All these help to assess the security and trust level or the status

of the OpenFlow switch. However, as discussed earlier, the attacker can use different techniques to maliciously insert or delete or alter the flow rules in the switches during runtime. Let us discuss how SSMA can detect such malicious flow rules in the switches during runtime.

As shown in Figure 3.1, the logical architecture of SSA consists of different modular components such as Attestation, Message Analyser, Switch Store, Switch State Generation, Switch State Validation, and Alert and Restore for detecting the attacks on the switches. The attestation module is used for validating the boot time state of the switches. The Message Analyser is used for validating the communications between the Controller and the switches and storing them in Switch Repository. The Switch Repository is used for maintaining the information related to different features or functionality of the switches and storing all the communications between the Controller and switch. Switch State Generation is used for generating trusted report on the expected state of the switches by querying the Switch Repository database. Switch State Validation is used for detecting the attacks by querying the switches for the switch_state_report and comparing them with the trusted report. Alert and Restore component is used for raising alerts to the SDN administrators when attacks are detected on switches and restoring the flowrules in switches. The Key Management Module is used for managing the symmetric keys for confidential communication. Here the enforcer helps to enforce the rules in the data plane layer.

The SSMA queries the switches in the data plane to report the flow rules in their flow tables. The switches generate a switch_state_report by extracting the flow rules configured in the flow table and update the SSMA's Switch Repository. Note that the component that is generating the switch_state_report and dealing with the switch state report is assumed to be trusted. Hence, switch_state_report includes all unauthorised changes to the previously configured rules such as deletion or modification of the rules and any new rules inserted by the attacker.

Since the SSMA has the information related to the functionality of the switch and all the authorised flow rules configuration messages from the Controller to the switches, it is able to generate a report on the expected state of the switch by querying its database. Since we assume the Controller is trusted, the report on the expected state of the switch generated by the SSMA can be trusted. Now SSMA compares its report stored in Switch Repository on the expected state of the switch with the switch_state_report. If there are any additional rules in the switch_state_report that were not configured by the SDN Controller, or if there is mismatch with any of the rules previously configured by the Controller, then the attack on the switch is detected. Now the SSMA generates an alert using Alert and restore module to the SDN administrator.

4. EXAMPLE SCENARIOS

This section will explain two hypothetical scenarios where SSMA can be of tremendous benefit. The SSMA components and features of the security architecture can resolve the following security issues. In the next report, we will present the main prototype implementation, which can help achieve success for the following scenarios.

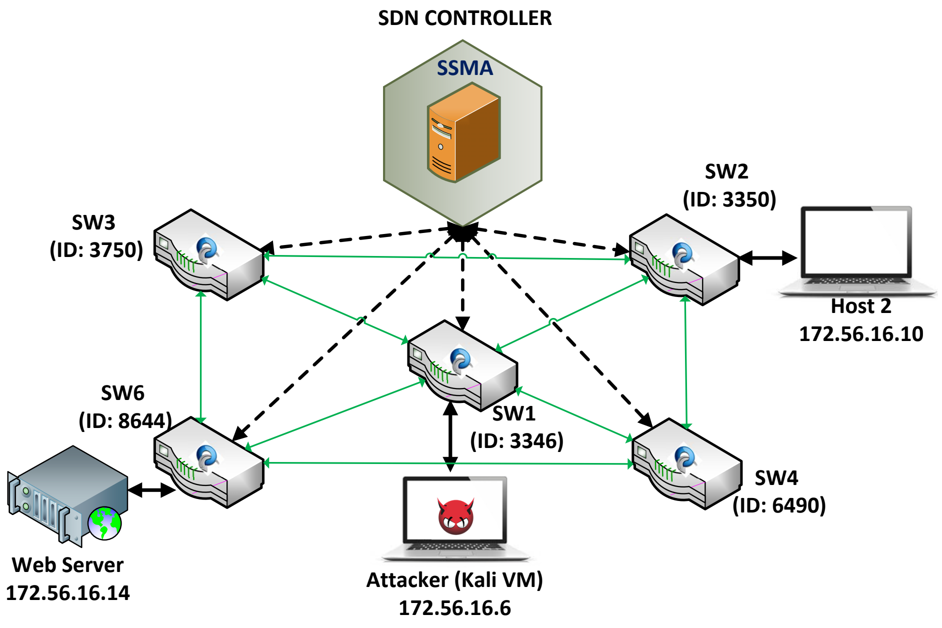

Figure 4.1: Network Setup

4.1 Attack Detection

This scenario is more specific to malicious adversaries attacking the OpenFlow switch. Their main target is to take control of the switch and manipulate the flow rules in flow tables. Such actions can lead to fake communication, leakage of privacy, the spread of malware and total disruption of the SDN infrastructure. We will now explain the attack and present how SSMA can help detect the attack from a conceptual viewpoint.

We are using a network setup presented in Figure 4.1. Let us consider that an attacker has initiated an attack on the switch SW1 to alter some of the flow rules, as shown in Figure 4.1. In this case, the attacker is using the Kali Linux machine to generate a ma-

licious flow_mod message to insert an unauthorised flow rule in SW1.
By default, ONOS does not provide any Controller core security. It depends on JVM for core application security. First, an adversary spoofs the IP of a legitimate host (HOST 1: Kali VM: IP 172.56.16.6) and uses that IP to inject the Malware code into the SDN network. The adversary can use the Metasploit framework of Kali Linux to deploy the Malware code via the network interface. After injecting the script via the network interface, it disables the JVM security features.
As a consequence, the adversary gains access to the ONOS Core. Once the adversary gains access to the core, he can intercept any method calls and change the values of methods, for example, the adversary intercepts the flow_mod() method calls, changes the value and sends them to OpenFlow switches. This type of malicious flows can help connect malicious hosts to the SDN environment.
Now we will present how the SSMA can detect this attack from a conceptual viewpoint

- Any external attack will be detected by the switch agent.
- During boot time, SSMA attests every OpenFlow switch and generates a secure and trustworthy state. This state reflects the internal and external state of the switch. When the attacker attacks and injects modified flow rules, the SSMA can check and verify the state. Hence, the SSMA can detect the attack.
- Ever if the OpenFlow switch is compromised and the attacker tries to re-route the flows from one destination to another, the policies in the SSMA will stop it from happening.
- Any security and path violation will be reported to the SSMA and alert module will make the network admins aware of them. The actions can be taken in demand.

4.2 Secure and Trust Aware Packet Routing

For this scenario, we will use the same network setup as Figure 4.1. Assume that the SW1 OpenFlow switch is compromised. The adversary intends to remain dormant in the network and intercept any packet flow towards the Webserver. The Controller in its default state does not have any capabilities to detect the security and trust status of the OpenFlow switches in the data plane. Also, the SDN data plane lacks the capabilities to maintain the privacy of the flow-communication. Hence, the SDN infrastructure lacks a secure and trustworthy routing mechanism.
We will present how SSMA can conceptually help achieve the above requirements.

- With SSMA, each OpenFlow switch is continuously assessed for its security and trust state. Each of them has been tagged with the security and trust label. The SSMA will have dedicated on-demand security service policies which will ensure secure and trustworthy routing.

- The Key Management Module in the SSMA will ensure confidentiality service for the dedicated flow-communication.

- The SSMA will also be able to ensure unauthorised interception of the communication.

- All operations will be logged and reported to the network admin.

5. CONCLUSION

The first report presented various threat vectors for SDN infrastructure. In this report we proposed a Switch Security Management Architecture. We want to use this switch security architecture in the form of a northbound application. We call it SSMA. The SSMA for SDN Controllers help to detect attacks on switches. Here we presented the operation of SSMA and discussed in detail the SSMA components. We have explained how SSMA validates the switch state at boot-time using remote attestation. Also, we have presented runtime state validation of the switches using trusted logs for detecting the attacks. The SSMA maintains a communications log between all the internal components and the switches. Also, the SSMA queries the switches to report the current flow rules in the switches. Such features help SSMA to verify the state of the potential malicious OpenFlow switch. Any switch abnormalities will the noticed by the SSMA and will generate an alert to the security administrator. Finally, we have presented how the proposed security architecture' component can help mitigate security attacks in SDN dataplane. In the next report, we will present how we have implemented the security architecture and its components. The next report will also present the performance of the SSMA.

BIBLIOGRAPHY

[1] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *2013 IEEE SDN For Future Networks and Services (SDN4FNS).* IEEE, 2013, pp. 1–7.

[4] S. W. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *ACM SIGCOMM workshop on Hot topics in software defined networking.* SIGCOMM, 2013, pp. 165–166.

[5] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 6, pp. 3514–3530, 2017.

[6] S. Lee *et al.*, "Delta: A security assessment framework for software-defined networks," in *Proc. of NDSS*, vol. 17, 2017.

[7] M. C. Dacier *et al.*, "Security challenges and opportunities of software-defined networking," *IEEE Security Privacy*, vol. 15, no. 2, pp. 96–100, March 2017.

[8] B. E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi, "Cross-app poisoning in software-

defined networking," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2018, pp. 648–663.

[9] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into sdn topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2019, pp. 1101–1114.

[10] W. Jiasi, W. Jian, L. Jia-Nan, and Z. Yue, "Secure software-defined networking based on blockchain," 2019.

[11] Q. Li, Y. Chen, P. P. C. Lee, M. Xu, and K. Ren, "Security policy violations in sdn data plane," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1715–1727, Aug. 2018.

[12] T. Park, Y. Kim, V. Yegneswaran, P. Porras, Z. Xu, K. Park, and S. Shin, "Dpx: Data-plane extensions for sdn security service instantiation," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 2019, pp. 415–437.

[13] S. Shin *et al.*, "Rosemary: A robust, secure, and high-performance network operating system," in *Proc. of the 2014 ACM SIGSAC conference on computer and communications security.* ACM, 2014, pp. 78–89.

[14] P. Porras et al., "A security enforcement kernel for openflow networks," in *Proc. of the 1st workshop on Hot topics in software defined networks.* ACM, 2012, pp. 121–126.

[15] H. Li et al., "Byzantine-resilient secure software-defined networks with multiple controllers," in *2014 IEEE International Conference on Communications (ICC).* IEEE, 2014, pp. 695–700.

[16] P. Chaignon, K. Lazri, J. François, T. Delmas, and O. Festor, "Oko: Extending open vswitch with stateful filters," in *Proceedings of the Symposium on SDN Research.* ACM, 2018, p. 13.

[17] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: protecting data and control plane resources under sdn-aimed dos attacks," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE.* IEEE, 2017, pp. 1–9.

[18] T. D. Nadeau and K. Gray, *SDN: software defined networks.* " O'Reilly Media, Inc.", 2013.

[19] O. N. Foundation, "Software-defined networking: The new norm for networks," https://www.opennetworking.org/images/stories/downloads/sdnresources/white-papers/wp-sdn-newnorm.pdf.

[20] *OpenFlow Switch Specification.*

[21] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Tech. Rep., 2008.

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

[23] K. Benton, L. J. Camp, and C. Small, "Openflow vulnerability assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 151–152.

[24] B. Agborubere and E. Sanchez-Velazquez, "Openflow communications and tls security in software-defined networks," in *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2017 IEEE International Conference on.* IEEE, 2017, pp. 560–566.

[25] D. Clark, "Policy routing in internet protocols. request for comment rfc-1102," *Network Information Center*, 1989.

[26] B. D. Payne, "Simplifying virtual machine introspection using libvmi," *Sandia report*, pp. 43–44, 2012.

[27] C. C. Aggarwal and C. Zhai, *Mining text data.* Springer Science & Business Media, 2012.