

“Integration of Intrusion Detection System with Software Defined Networks to Secure Cloud”

A

Project Report

*submitted in partial fulfillment of the
requirements for the award of the degree of*

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Name	Roll No.
Chaitanya Aggarwal	R110211011
Ishmeet Kaur	R110211018
Bhavya Kalra	R110211053

under the guidance of

Mr. G.L Prakash



Department of Computer Science & Engineering

Centre for Information Technology

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, UK

March – 2015

Acknowledgements

We wish to express our deep gratitude to our guide **Mr. G.L Prakash**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank to our respected Program Head of the Department, **Dr. Amit Aggarwal**, for his great support in doing our project in Area at CIT.

We are also grateful to **Dr. Manish Prateek**, Associate Dean and **Dr. Kamal Bansal**, Dean COES, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

Chaitanya Aggarwal
R110211011

Ishmeet Kaur
R110211018

Bhavya Kalra
R110211053



The innovation driven
E-School

CANDIDATES DECLARATION

We hereby certify that the project work entitled Integration of Intrusion Detection System with Software Defined Networks to secure Cloud in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Cloud Computing and Virtualization and submitted to the Department of Computer Science & Engineering at Center for Information Technology, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **January, 2015 to April, 2015** under the supervision of **Mr.G.L Prakash, Assistant Professor**. The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

Chaitanya Aggarwal(R110211011)

Ishmeet Kaur(R110211018)

Bhavya Kalra(R110211053)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

April, 2015

Mr. G.L Prakash

Project Guide

Dr. Amit Aggarwal

Program Head B.tech C.S CCVT

Center for Information Technology

University of Petroleum & Energy Studies

Dehradun 248 001 (Uttarakhand)

Abstract

Cloud computing platforms have been widely proposed and implemented due to the flexibility, scalability, high availability and efficiency. Based on compromised cloud resources, attackers may spam, disseminate malicious codes, crack passwords and security keys, compromise vulnerable VMs and then deploy DDoS attacks, deploy botnet command and control, etc. Security has been considered as one of the top concerns in clouds. Intrusion Detection and Prevention Systems (IDPS) have been widely deployed to enhance the cloud security.

An intrusion detection system (IDS) is designed to monitor all inbound and outbound network activity and identify any suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system. IDS come in a variety of flavors and approach the goal of detecting suspicious traffic in different ways. An Intrusion Prevention System (IPS) is a network security/threat prevention technology that examines network traffic flows to detect and prevent vulnerability exploits. Integrating these technologies by using Software-Defined Networking approaches to enhance the system security in clouds has been the current area of research in companies. In this project, we aim to establish a comprehensive IPS solution to reconfigure the cloud networking environment on-the-fly to counter malicious attacks which none of existing works have established.

A complete SDN based IPS solution called SDNIPS (Software Defined Networks Intrusion Prevention System) that is a full lifecycle solution including detection and prevention in the cloud is the main objective of the project. The solution proposes a new architecture which uses Snort-based IDS and Open vSwitch (OVS). Existing countermeasures usually provide add-on and customizable security models, and consider the cloud can afford the demanded resource. The performance of the entire system is evaluated by comparing the SDN based IPS solution with the traditional IPS approach from both mechanism analysis and evaluation. Finally, evaluations of SDNIPS demonstrate its feasibility and efficiency over traditional approaches.

Contents

Acknowledgements	i
Abstract	iv
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Motivation	2
1.3 Organization	3
2 SYSTEM ANALYSIS	5
2.1 Related Work	5
2.2 History	7
2.3 Existing System	10
3 BACKGROUND	18
3.1 Statement of Purpose	18
3.2 Main Objectives	20
3.3 Sub-Objectives	22
3.4 Assumptions	22
4 SYSTEM DESIGN	24
4.1 Proposed System	24
4.2 Design	26
5 ALGORITHM	29
6 IMPLEMENTATION	31
7 RESULT ANALYSIS	36
7.1 System Specifications	36
7.2 Output Screens	37

8 CONCLUSION AND FUTURE ENHANCEMENTS	42
---	-----------

Bibliography	44
---------------------	-----------

List of Figures

2.1	SDN Architecture	12
2.2	Basic SDN Flow Control	13
2.3	Open Flow Architecture	14
2.4	Process Flow Diagram	16
3.1	Underlying Topology of the project	19
4.1	Components of Snort	26
4.2	SDNIPS Flow Architecture	27
4.3	Proposed System Design	28
7.1	Mininet nat.py starting	37
7.2	Starting Snort and Pinging a device to test snort	38
7.3	Alert History Table	38
7.4	Rules Generated by Snort	39
7.5	Host1 pings Host2,Snort detects port 8080	40
7.6	Host1and Host2 communicate via telnet	41
7.7	Host1 and Host2 no longer able to communicate	41

Chapter 1

INTRODUCTION

1.1 Introduction

In traditional networks, in order to implement an Intrusion Detection and Prevention System (IDPS), one always needed to place the device in line with the network flow to have each packet analyzed as it is being transferred from source to destination through the network. If a network packet is found malicious, it is dropped from the network. The procedure of dropping packets is currently the only preventative countermeasure traditional IPS systems are capable of performing. However, the previous work implemented a system called Snort-Flow, which combined the IDS ability of snort with the network flow control capability of an Open-Flow switch [1]. The amalgamation would allow for more preventative measures. A detected intrusion could be operated by simply redirecting the traffic, blocking specific ports, or more. This is important as with these additional options, the changing network topology can not only prevent the current intrusion attempt detected by Snort, but also impact and hinder any follow-up malicious activity, which is currently outside the capabilities of packet-dropping IPS.

We will first be implementing the basics of SnortFlow, similar as described in previous work using Snort and an OpenFlow Controller [1]. We will then implement some improvements to that initial design by adding in some additional features. An alert receiver

and parser will be implemented in the rule server. A database will be used to store data related to the action pool, historical events, and current rules. Both of them are used by the controller to determine the flow of traffic. We will also consider synchronization of rules between the controller and sensors. Once complete, we expect to have a functional IPS system that can be deployed to a larger network and be able to control each network segment using OpenFlow switches in order to take preventative actions against network intrusions. We will also evaluate how well our final implementation does in terms of performance and prevention rate. By the end of the project, we aim to complete the project by building out the environment, implementing the IPS Rule Server to include the Alert Receiver, the Alert Parser, the database, and the Rule Generator, implement the IPS on the controller. We also added an event handler in the basic layer 2 learning switch default example code so that the learning switch can process events created in the IPS server.

1.2 Motivation

Cloud Computing has made the consumers both excited and nervous as the reduction in capital cost comes with the expense of loss of direct control thereby increasing risks in the entire system if not properly secured [2].

Intrusion Prevention is one of the main challenges of Internet security today. Most of Internet security is hosted on the network's edge [3]. However, the majority of attacks occur internally. In this scenario, not only an Intrusion Detection System, but also a Prevention System, which takes actions in order to block the attack, is required. Future Network is based on the pluralist approach, by allowing distinct logical networks in the same physical space. In order to obtain this level of abstraction, the virtualization paradigm is introduced. In a virtualized environment, resources, such as memory, CPU and bandwidth, are shared between multiple virtual machines. In this context, network virtualization is achieved, through the usage of virtual machines as network elements.

The architecture proposed in this project provides an Intrusion detection as well as Prevention System for future networks based on Software Defined Network and its capability to integrate with the Snort as well as the database to store the upcoming packets in form of alerts and classify them in terms of the signatures based in IDS of Snort and then finally reach to a response of either denial of the packet by dropping it or forwarding it from the controller itself. SDNs plane separation provides more programmability to control the virtual switches, and therefore, creates a managing environment for an Intrusion Preventing System. The traditional networks incapability to detect and prevent the upcoming malicious packet in term of threat in a cloud architecture was the major source of inspiration to combine the upcoming technologies with the existing Intrusion and Detection capability of SNORT and the storage capability of the database MySQL server to prevent further attacks thereby securing the entire system. The project has been divided into four sub tasks to achieve the aim of detection and prevention of the threats in cloud environment through Software Defined Networking.

1.3 Organization

The entire report is organized into nine chapters providing an in-depth view of the entire project accompanied by the proposed design and the analysis of the work. The nine chapters have been summarized below.

The first chapter provides an introduction to our project i.e. it states what we are going to perform and what are we expecting from our project. It also mentions what motivated us to take up this particular project. This is followed by the amendments we have made to the already existing model. The second chapter entails the literature study conducted by us which helped us realize the opportunity to contribute to the existing systems. Also it includes the history of software defined networks and snort, the intrusion detection and prevention software.

The third chapter provides a detailed explanation of the problem for which we are trying to engineer a solution. Further it explains how we comprehend our problem, by dividing it into several main and sub objectives. These objectives involve segregating our project into various phases. Assumptions taken into account during the project are also mentioned in this chapter. Fourth Chapter explains the complete architecture of the system that we are proposing in order to engineer a solution to the problem stated in the previous section. Design of the solution is further explained in detail, which includes various components of our system along with the interactions taking place amongst those components.

The fifth chapter contains pseudo codes developed for algorithms that are being used in our project. The next chapter provides a detailed procedure involved in implementing the various components of our project. The first part deals with the information about the setup environment. The second part deals with building test environment followed by implementing the IPS rule server.

The seventh chapter gives a detailed analysis of the results being generated from various topologies being implemented in the network. Also graphs for various topologies depicting the efficiency of the proposed system have been incorporated. The chapter titled conclusion and future enhancements, describes the inferences that can be made on the basis of various results that have been achieved and the objectives that have been completed. The last chapter named references provides a list of various References that have been made throughout the report.

Chapter 2

SYSTEM ANALYSIS

2.1 Related Work

Software Defined Networking is an outlook to computer networking which allows network administrators to manage network services through abstraction of lower-level functionality. It is achieved by decoupling the Controller that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the selected destination (the data plane). The inventors and vendors of these systems claim that this simplifies networking.

The security and integrity of software-defined networking remains unproven and is particularly infected by perpetual attacks on control plane communications and the vulnerabilities in controllers. Security and dependability of SDN still is a field almost unexplored, presenting many challenges and opportunities. There are only a few closely related works [4] where the essential idea is to provide a security kernel (e.g., by extending a controller like NOX) capable of ensuring prioritized flow rule installation on switches. Applications are classified in two types, one for security related applications and another for all remaining applications. The first type represents specialized programs used to ensure security control policies in the network, such as to guarantee or restrict specific accesses to the network or take actions to control malicious data traffic.

Flow rules generated by security applications have priority over the others. The security kernel is responsible for ensuring this behavior. FRESCO is an extension of this work that makes it easy to create and deploy security services in software-defined networks. However, none of these works fosters or enforces the security of SDN itself, the goal we are pursuing.

Presently Open Flow is a new network technology and an open standard for Software Defined Networking (SDN) in which the control plane and data plane of network equipment is separated. In this project we propose a concept of an open flow switch that contains Intrusion Detection System in it and a controller that contains Intrusion Prevention System. Security approaches based on Software Defined Networks have been considered as the trend for future virtual networking security solutions in a cloud virtual networking environment. Opensafe is a system utilizing both OpenFlow and Snort technology but they focused on the area of how to route traffic to monitoring appliances, rather than attempting to provide a comprehensive detection and prevention solution[5]. In the paper titled Openflow random host mutation: Transparent moving target defence using software defined networking, the authors proposed a mechanism called OpenFlow Random Host Mutation (OFRHM)[6] in which the OpenFlow controller frequently assigns each host a random virtual IP that is translated to/from the real IP of the host. This mechanism can effectively defend against stealthy scanning, worm propagation, and other scanning-based attack, but does not work when the attackers know the internal address of victims. In a recent work titled Network intrusion detection and countermeasure selection in virtual network systems, [7] IEEE Transactions on Dependable and Secure Computing (TDSC), the authors presented an SDN-based IDS/IPS solution to deploy attack graph to dynamically generate appropriate countermeasures to enable the IDS/IPS in the cloud environment. SnortFlow, is another recent work focusing on the design and preliminary evaluation of OpenFlow, a communications protocol that gives access to the forwarding plane of a network switch or router over the network, enabled

IPS in the cloud environment. Moreover, the work entitled SDNIPS: Enabling Software-Defined Networking Based Intrusion Prevention System in Clouds[8] proposes a new IDPS architecture based on Snortbased IDS and Open vSwitch (OVS). It also compares the SDN based IPS solution with the traditional IPS approach from both mechanism analysis and evaluation. An SDN-based IPS Development Framework in Cloud Networking Environment[9] presents an IPS development framework to help user easily design and implement their defensive systems in cloud system by SDN technology. This framework enables SDN approaches to enhance the system security and performance. Design of Event-Based Intrusion Detection System on OpenFlow Network presents a design of an event-based Intrusion Detection System (IDS) architecture on Openflow network for better network security[10].

However, to our best knowledge, none of them address all the issues addressed below:

1. How to establish an efficient SDN-based IPS solution in the cloud virtual networking environment;
2. How to design the SDN-based IDS/IPS networking architecture that provides a dynamic defensive mechanism for clouds;
3. Expanding the previous designs by implementing scalability and additional preventative functionality.

Therefore, motivated by the issue above, we propose a project involving integration of a python based IDS/IPS based on SDN network management, i.e., Open vSwitch (OVS) to a SDN controller. This project proposes a new design of IDS/IPS based on SDN network management, i.e., Open vSwitch (OVS).

2.2 History

In 1995 [9], the technology software-defined networking (SDN) came into existence shortly after Sun Microsystems released Java. One of the first and most notable SDN

projects was AT&T's GeoPlex. AT&T Labs Geoplex project members Michah Lerner, George Vanecek, Nino Vidovic, Dado Vrsalovic leveraged the network APIs and dynamic aspects of the Java language as a means to implement middleware networks. AT&T wanted a "soft switch" that could reconfigure physical switches in the network and load them with new services from an OSS. However, when provisioning services GeoPlex could not reach deeply into the physical devices to perform reconfiguration. The operating systems running on networked devices in the physical network therefore became a barrier to early SDN-like service delivery.

In 1998, Mark Medovich, a senior scientist of Sun Microsystems and Javasoft, left Sun to launch a Silicon Valley soft switch startup WebSprocket. Medovich designed a new network operating system, and an object oriented structured runtime model that could be modified by a networked compiler and class loader in real time. In July 2000, WebSprocket released VMFoundry, the Java to bare metal structured runtime compiler, and VMServer, a networked device compiler application server. Custom networked devices were preloaded with images created by VMFoundry then deployed on the network and connected to VMServer via UDP or TCP services plane, which could proactively or reactively load or extended network protocol methods and classes on the target system. WebSprocket's version of SDN, therefore was not confined to a set of limited actions managed by an SDN controller. Rather, WebSprocket's "control plane" contained code that could change, override, extend, or enhance Network protocols on operating networked systems.

In Summer of 2000, Ericsson's advanced network research engineers saw an immediate need and visited WebSprocket to design and architect features of a next generation soft switch thus taking first steps to build the world's first commercial soft switch. Sometime during 2000, the Gartner Group recognized the emergence of programmable networks as the next big thing for the Internet and introduced the "Supranet", the fusion of the

physical and the digital (virtual) worlds as "internet of things". and by October 2000 the Gartner Group selected WebSprocket as one of the top emerging technologies in the world.

In April and May 2001, Anjaneya Prasad Calyam,[11] a graduate student at the Ohio State University and researcher at OARnet, ran the first SDN test and developed the first practical SDN use case for Internet. The telecom market deflated in 2001 and Ericsson's soft switch development program came to an end, thus stalling the only known commercial SDN soft switch R andD effort at that time. Software-defined networking (SDN) was continued with work done in 2003 by Bob Burke and Zac Carman developing the Content Delivery Control Network patent application that eventually was issued as two US patents. The Open Networking Foundation was founded in 2011 [12] to promote SDN and OpenFlow .At the 2014 Interop and Tech Field Day, software-defined networking was demonstrated by Avaya using Shortest path bridging and OpenStack as an automated campus, extending automation from the data center to the end device, removing manual provisioning from service delivery.

Originally released in 1998 by Sourcefire founder and CTO Martin Roesch[12], Snort is a free, open source network intrusion detection and prevention system capable of performing real-time traffic analysis and packet logging on IP networks. Initially called a lightweight intrusion detection technology, Snort has evolved into a mature, feature-rich IPS technology that has become the de facto standard in intrusion detection and prevention. With over 4 million downloads and nearly 400,000 registered users, it is the most widely deployed intrusion prevention technology in the world. Snort's open source network-based intrusion detection system (NIDS) has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching, and content matching. These basic services have many purposes including application-aware triggered quality of service, to de-prioritize

bulk traffic when latency-sensitive applications are in use.

Combining the above two technologies we get a Software Defined Networking Intrusion Prevention System. This has been used in this project to implement an IDPS i.e Intrusion Detection and Prevention solution to the software defined networks. We aim to propose a new IDPS architecture based on Snort based IDS and OpenvSwitch (OVS). It also classifies the requests via the action pool and the dates of the events along with traffic analysis based on Ip addresses as well as the Mac addresses which better than the traditional IPS approach from both mechanism analysis and evaluation.

2.3 Existing System

Cloud computing refers the use of computing resources like hardware and software which can be delivered as a service over a network. It is a solution for providing on-demand access to computing infrastructure. End users can visit cloud based applications by web browser, lightweight desktop, mobile devices at a remote location while users data, information and computing resources are stored in cloud infrastructures. It has been widely deployed to day because the demanding resource provisioning capabilities. However, security has been one of the top concerns in cloud community while cloud resource abuse and malicious insiders are considered as top threats. Some attacks, such as spam, cracking passwords, performing malicious code and compromising vulnerable virtual machines can happen in a high possibility in current cloud system. Based on compromised cloud resources, attackers may spam, disseminate malicious codes, crack passwords and security keys, compromise vulnerable VMs and then deploy DDoS attacks, deploy botnet command and control, etc [13]. Existing countermeasures usually provide an add-on and customizable security models, and consider the cloud can afford the demanded resource .Establishing the Intrusion Detection and Prevention System (IDPS) is a good way to protect the cloud system with both detection and prevention capability. There have

been various algorithms developed with IDS configured. Traditionally, the IDS can be configured and enabled to be an IPS.

Operating and maintaining a computer network is an arduous task. To express the required high-level network policies, network operators need to configure each individual network device separately - from a heterogeneous collection of switches, routers, middle boxes, etc. - using vendor specific and low-level commands. In addition to configuration complexity, networks are dynamic, and operators have little or no mechanisms to automatically respond to network events. It is therefore difficult to enforce the required policies in such a continually changing environment. Traditional networks are managed through low-level and vendor-specific configurations of individual network components, which is a very complicated and error-prone process. And nowadays computer networks are becoming increasingly complex and difficult to manage. This increases the need for a general management paradigm that provides common management abstractions, hides the details of the physical infrastructure, and enables flexible network management. Making the network programmable leads to such a general paradigm, as programmability simplifies network management and enables network innovations.

Software Defined Networking (SDN) has been proposed to enable programmable networks. In SDN, the network is considered to have two components:

1. control plane which determines how to handle and forward data traffic, and
2. data plane which handles and forwards data traffic toward its destination.

The logical centralization of the control logic in a software module that runs in a standard server —or the network operating system offers several benefits. First, it is simpler and less error-prone to modify network policies through software, than via low-level device configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain the high-level policies in place. Third,

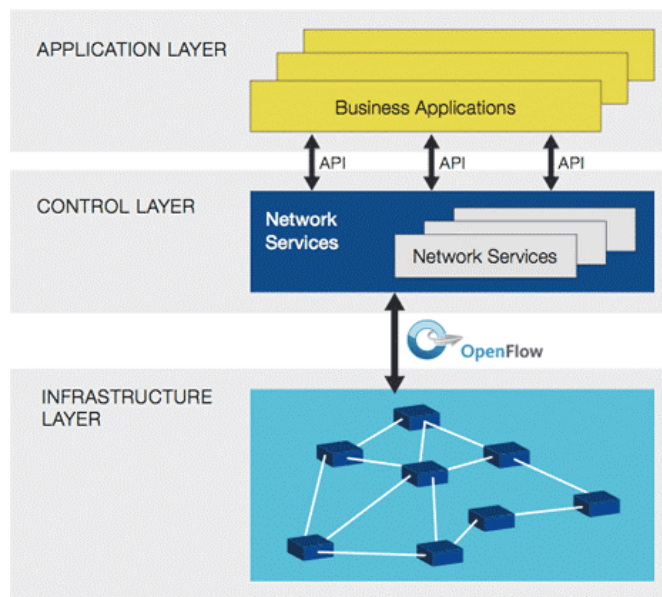


Figure 2.1: SDN Architecture

the centralization of the control logic in a controller with global knowledge of the network state simplifies the development of more sophisticated network functions.

The SDN architecture is:

- **Directly programmable:** Network control is directly programmable because it is decoupled from forwarding functions.
- **Agile:** Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- **Centrally managed:** Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- **Programmatically configured:** SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.

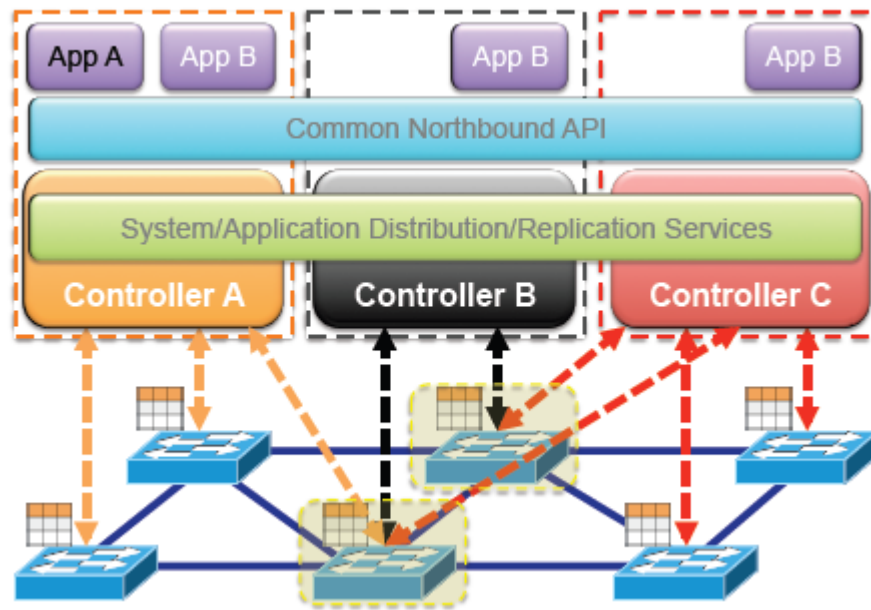


Figure 2.2: Basic SDN Flow Control

- Open standards-based and vendor-neutral: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

OpenFlow [14] is the most representative protocol implementing the SDN concept. SDN-enabled devices could be manipulated by OpenFlow. Open Flow is such a protocol that gives the management layer access to switches and routers. With the separation of the control plane from the data plane that lays the ground to the Software Defined Networking paradigm, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller, even though practically physically distributed.

It defines standard control interfaces, implement pre-programmed control policy, such as packet-forwarding rules in OpenFlow switches, inserts rules to flow tables and then handle data packets delivery. The programmable network interface enables us to define and configure network in an extremely flexible and efficient way. Thus, the emerging

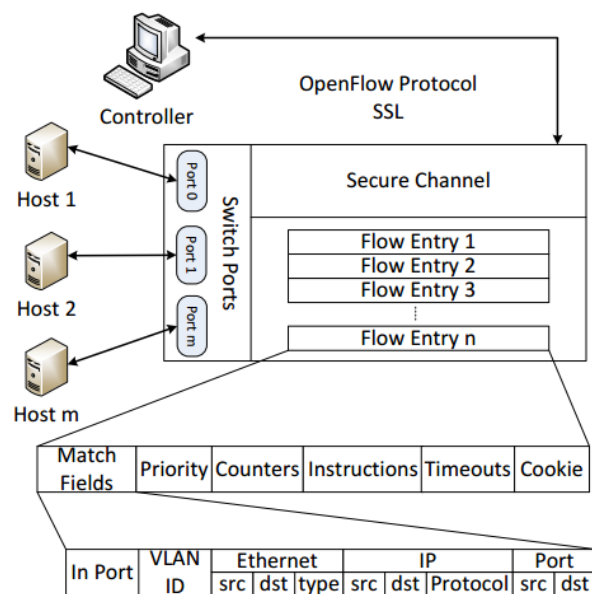


Figure 2.3: Open Flow Architecture

of SDN introduces an innovative approach to protect network with both flexibility and compatibility, which is a great fit for cloud environment.

An IDS is a security tool that allow us to monitor our network events searching attempts to compromise the security of our systems[15]. Its possible matching predefined rules emulating the behavior of an attack and its possible to deny the package or simply alert us to an email or sending messages to log. Basically we can find two types of IDS:

- HIDS: Host based IDS, monitors the activity of a single machine, searching abnormal behaviors.
- NID: Network IDS, capture and analyse network packages to search attack patterns.

Generally an IDS can be located in each network segment, for example front of the firewall or back of the firewall or also can be implemented in the same firewall if we have a small network traffic, with this way we can analyse all input and output traffic.

Intrusion prevention systems (IPS), also known as intrusion detection and prevention systems (IDPS), are network security appliances that monitor network and/or system activities for malicious activity. The main functions of intrusion prevention systems are to identify malicious activity, log information about this activity, attempt to block/stop it, and report it

In order to implement an Intrusion Prevention System (IPS) today, you must place a device in line with the network flow to have each packet analyzed as it's flowing through the network. If a network is deemed malicious, it is dropped. This packet dropping is currently the only preventative countermeasure today's IPS systems are capable of performing. Previous work implemented a system called SnortFlow[16]. Snort is a NIDS, implements real time scanning of attack detection and port scanning detecting. Snort-Flow combined the IDS ability of snort with the network flow control capability of an OpenFlow switch. This combination would allow for more preventative measures to be taken against a detected intrusion, such as simply redirecting the traffic, blocking specific ports, or more. This is important as with these additional options, the changing network topology can not only prevent the current intrusion attempt detected by Snort, but also impact and hinder any follow-up malicious activity, which is currently outside the capabilities of packet-dropping IPS.

The basic architecture of snort:

- Packet capture module: Used to capture network traffic using libpcap library.
- Decoder: It ensures to form the data structures of the packages captured and identify the network protocol.
- Pre-processor: pre-processors are plugins developed generally in C and process the

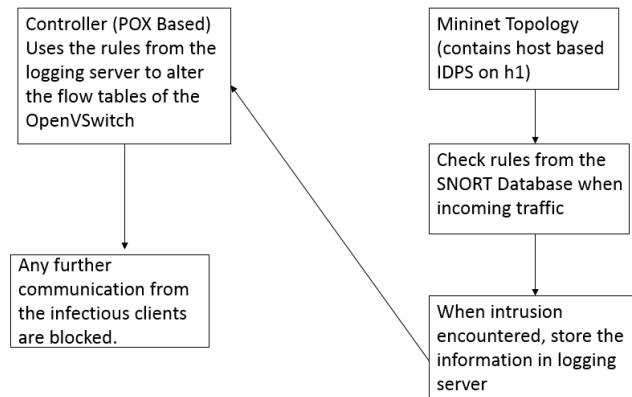


Figure 2.4: Process Flow Diagram

packets provided by the decoder and ensembles the packets received. This pre-processors are configured in `snort.conf` file configuration. Some pre-processor examples may be: `sfPortscan` `Frag3` `HTTP` `SSH` Detection engine: Analyze the packets based in our rules configured.

- Detection plugins: Used to modify the behavior of the detection engine.
- Output plugins: Defines how and where saves the packet alters and the packages generated.

We will first be implementing the basics of Snort Flow. We will then implement some improvements to that initial design by adding in some additional features. An alert receiver and parser will be implemented in the rule server after making the topology. A database will be used to store data related to the action pool, historical events, and current rules. Both of them are used by the controller to determine the flow of traffic. Once complete, we aim to have a fully functional IDPS system that can be deployed to a large network and be able to control each network segment using Open Flow switches in order to take preventative actions against network intrusions.

We also added an event handler in the basic layer 2 learning switch default example code so that the learning switch can process events created in the IPS server. After completing the task 3, we evaluate how the overall result functions. The result derives

from the performance of dropping packets and prevention rates.

Chapter 3

BACKGROUND

3.1 Statement of Purpose

In order to implement an Intrusion Detection System, we need to place a device in line with the network flow architecture. If it is found malicious, it is dropped. This packet dropping is currently the only preventative countermeasure today's IPS systems are capable of performing.

If we combine the IDS capability of Snort with the flow control of Open Flow Switch, it would prevent the detected intrusion in a better manner like the traffic can be redirected, specific ports can be blocked and the like [17]. This is important as with these additional options, the changing network topology can not only prevent the current intrusion attempt detected by Snort, but also impact and hinder any follow-up malicious activity, which is currently outside the capabilities of packet-dropping IPS. The basic topology used in the project is illustrated via the figure below. It includes a topology consisting of a set of hosts via mininet [16] on one virtual machine in the VMWare Workstation virtualization environment and a controller on a separate virtual machine in the same virtualization environment. The controller is also established utilizing the mininet topology and is accompanied by the POX controller as well as the MySQL server which stores all the log details and the rules through which any event is classified after passing

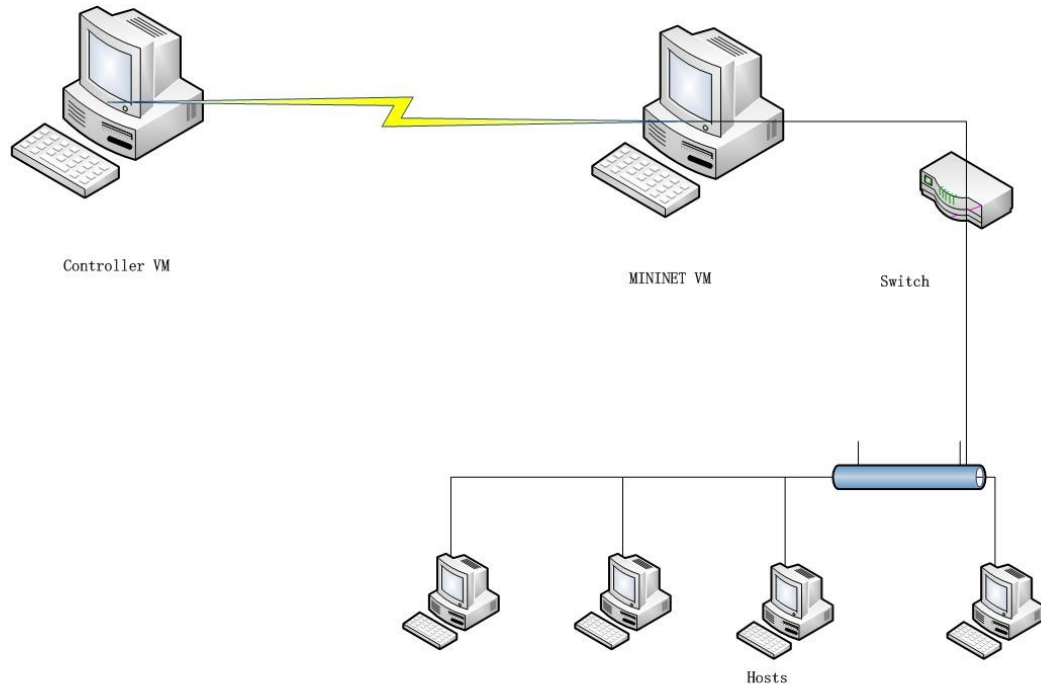


Figure 3.1: Underlying Topology of the project

through the snort server implemented in the other virtual machine. The basic underlying topology will then receive the requests which is analyzed by Snort through port number 6633 and the entire information gets into the database of action pool in the tables made by My SQL in the controller virtual machine. The rule server consists of alert receiver and parser which parses the requests via POX controller to further classify the requests and determine the flow of traffic.

In order to address the critical issue of cloud security, there are various SDN based solutions being worked upon. However, none of existing works established a comprehensive IPS solution to reconfigure the cloud networking environment on-the-fly to counter malicious attacks. In this project, we present an SDN-based IPS solution called SDNIPS (Software Defined Networking Intrusion Prevention and Detection System). Establishing the Intrusion Detection and Prevention System (IDPS) is a good way to protect the

cloud system with both detection and prevention capability.

The basic topology which consists of four hosts connected to a switch via the mininet and one controller which is the main point of prevention. The SNORT is installed on the mininet itself which is detecting the malicious packets by parsing the alerts based on the predefined signatures and the entry is made in the database along with the responses. The controller will then query the database and based on the responses in the database stored by the SNORT, the syslog server on the controller parses the alert history table to make a decision based on the traffic. The rule will be generated and updated via the script on the controller in the database by looking primarily at the the Classification of the alert generated by the sensor of SNORT. Classification includes categories such as A Network Trojan was detected and Attempted User Privilege Gain.

3.2 Main Objectives

In order to implement an Open flow based Intrusion Detection and Prevention System in Cloud environment, we have utilized SNORT and integrated in the SDN controller.

This project will be completed by four steps. An IDS Rule Server is needed, which plays the role of reading alerts and providing key data to controller. Considering the challenge of reconfiguration of network, a rule generator should be integrated into the controller so that it does some prevention actions as necessary. Then the Snort sensor in each node of network has traffic packets spanning to it for analysis, and will alert when detection occurs.

The various tasks are categorized in the project which are as follows:

Task 1: Build test lab

Build out the network and software installations that will be needed to implement the project [21]. There should be several networks controlled by a SDN switch, each with a

snort device monitoring the network. We are looking to utilize Mininet for this task.

Task 2: Implement IDS Rule Server

We have divided task 2 into several sub tasks to further explain each component of the IDS Rule Server. Task 2.1: Implement Alert Receiver - this component is responsible for receiving snort alerts via syslog listener, and then forwarding them to the alert parser.

Task 2.2: Implement Alert Parser - this component parses the syslog message, inserts alert data into the database, and finally passes the alerts to the rule generator.

Task 2.3: Implement Database - this component contain the tables of current rules, action pool, and historical alerts.

Task 2.4: Implement Rule Generator - this component is responsible for analyzing the alert, querying the historical alert table for similar previous alerts to correlate alerts from multiple snort detections, and then determining the action to take. Upon deciding on an action, it updates the rule table, and triggers an event to the controller.

Task 2.5: Implement basic controller - initially, this component will act as a layer 2 switch that is capable of receiving events from the rule generator. This task will not implement the IPS action; it will just build the framework for it.

Task 3: Implement IPS rule functionality

IPS functionality will be implemented in the controller. In Task 2.5, we built the basic framework and communication between the IPS Rule server and controller, and in this task, we will have the controller actually apply the rule determined by the rule generator for a given flow.

Task 3: Evaluate Implementation

Evaluate the impact to the overall network in terms of dropped packets and rate of detected packets from a varying number of hosts and increase in the level of security and reliability of the system.

3.3 Sub-Objectives

- Establish and configure a SDN
- Learn and implement various network topologies using SDN.
- Installation of host based Intrusion Detection System (SNORT) as Inline Mode on Mininet Simulator on the host Virtual Machine in VMWare Workstation.
- Directing the packet flow to pass into the network via the IDS host i.e Snort by the network topology in the virtual machine.
- Creating a plugin on python based RYU Controller on the controller virtual machine which enables the listening of data packets only from the IDS host, and redirects the packets into the network via the Data Plane and Snort and stores the data into action pool in MySQL server.
- Defining the rules in the rule generator of the My SQL database table to further analyze each event.
- Securing and Implementing an Intrusion Detection Prevention System System in SDN via the controller VM and by classifying the actions by the alert interpreter and alert parser and then generating the rules via rule generator.
- Reducing the controller related vulnerabilities by adding more features into the rule generator.
- Minimizing risk related to network configurations in VM as well as SNORT.

3.4 Assumptions

The project has been implemented keeping in mind the following assumptions:

-
- The entire topology is assumed to represent a real life scenario of a cloud architecture wherein the hosts are the separate virtual machines or dumb terminals being controlled by a single physical system.
 - The packets being sent as malicious by one host to other are assumed to be the packets which hamper the functioning of a normal architecture by destroying the security, either by leakage of information or by denial of service.
 - The packets are detected based on the signatures defined in the SNORT.

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

Cloud Cluster is the basic architecture which the project is aiming to secure via Software Defined Networking environment by aiming to implement a SDNIPS (Software Defined Networking Intrusion Prevention System)[5]. One or more cloud servers are present in a cloud cluster which is assumed to be replicated in the topology set up. In this project, our established system is based on VMWare Workstation that is an efficient parallel virtualization solution on top of the bare metal.

SDN decouples the underlying nodes from the main system controlling the entire topology via controller plane. Its security approaches are crucial to the virtual networking security solutions in a cloud networking environment.

Controller is a component in SDN providing control and a centralized view over the cloud virtual network. There are three major components in SDN while implementing an Intrusion Detection and Prevention System,

- **SDNIPS Daemon:** It is responsible for recognizing alerts and collecting the data from the SNORT daemon installed along with mininet on the switch through Open virtual Switch. It collects the data and stores it in the MySQL server.

- Alert Interpreter: It takes care of parsing the alert and targets the suspect traffic. There are several subsets of information analyzed from the alert like timestamp, source IP address, destination IP address, TCP port, etc.
- Rules Generator: The parsed and filtered information from the alert is passed to rule generator which generates the rules to be injected to the OpenFlow device in the controller itself to reconfigure the network.

Open-vSwitch (OVS) in SDN is a pure software implementation of the OpenFlow switch, which is usually implemented in the management domain of the cloud system. In user-space of OVS, there are two modules which are `ovsdbserver` and `ovs-switchd`. The module `ovsdb-server` is the log based database that holds switch-level configuration; while the module `ovs-switchd` is the core OVS component that supports multiple independent data-paths (bridges). The `ovs-switchd` module is able to communicate with `ovsdb-server` through management protocol, with controller through Open Flow protocol, and with kernel module through `netlink`. In the kernel space, kernel module handles packet switching, lookup and forwarding, tunnel encapsulation and decapsulation. Every Virtual Interface (VIF) on each VM has a corresponding virtual interface/port on OVS, and different virtual interface connecting to the same bridge can be regarded on the same switch.

Snort is the de facto open-source IDS/IPS solution and a packet analysis tool which has overall performance strength over other products. It has sniffer, packet logger, and data analysis tools. In its detection engine, rules form signatures to judge if the detected behavior is malicious. It excels at traffic analysis and packet logging on IP networks. The Snort architecture is illustrated below.

Through protocol analysis, content searching, and various pre-processors, Snort detects thousands of worms, vulnerability exploit attempts, port scans, and other suspicious behavior. It uses a flexible rule-based language to describe traffic that it should collect or pass, and a modular detection engine.

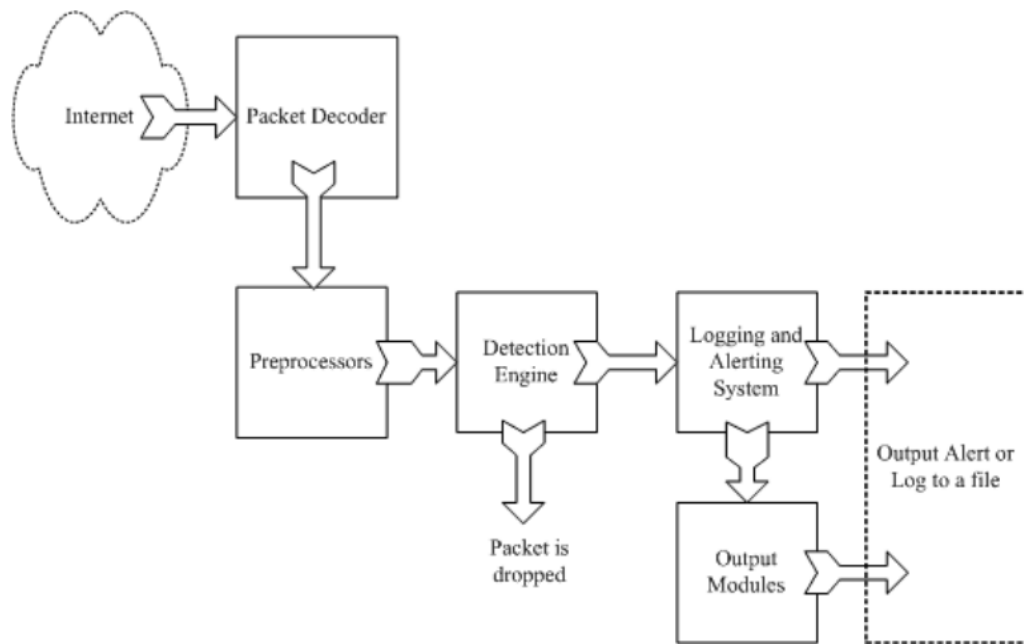


Figure 4.1: Components of Snort

4.2 Design

In this project we will be designing, implementing, and evaluating an openflow-based IPS system in the cloud environment. We will be utilizing an IDS technology, namely Snort, in order to detect indications of malicious activities. This will be integrated into a Software Defined Network controller, controlling an openflow switch, in order to take preventative actions based on detections from Snort. We will be using ideas from "SnortFlow" [17], which investigated combining these two technologies to create a full Intrusion Prevention System, and expand on the initial designs. Finally, we will be evaluating the performance of this design to determine how realistic using our implementation would be in a cloud virtual environment.

The processing flow of a SDNIPS (Software Defined Network Intrusion Prevention System) is illustrated in the figure above. The network traffic is generated from the cloud resources i.e., VMs.

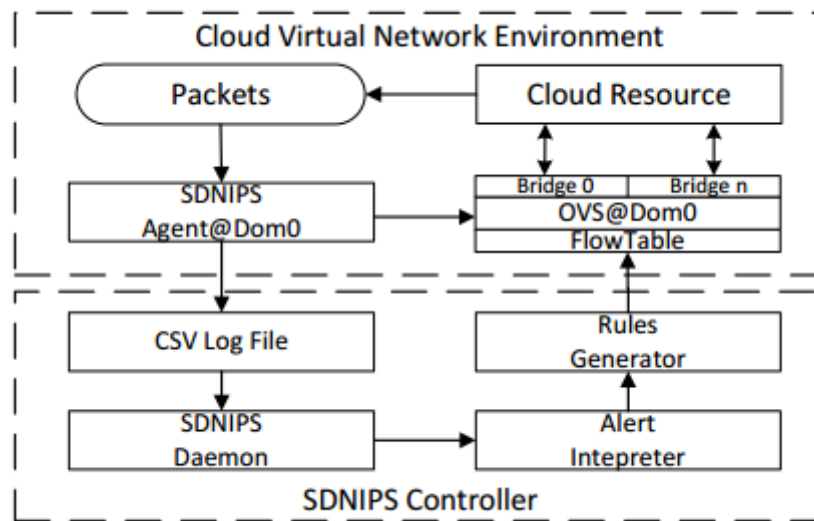


Figure 4.2: SDNIPS Flow Architecture

- Snort daemon has the advantage of directly detecting through the bridge, which is more efficient than sniffing the traffic.
- When any traffic i.e. alerts matching the Snort rules is alerted into the log file, The SDNIPS daemon will store the alert information in the database format and send over to the MySQL server at controller side.
- After that, the alert interpreter will parse the alert information and extract all necessary information, e.g., timestamp, priority, attack type, source IP, destination IP, TCP port, etc.
- Finally, the rules generator at the controller side will generate the OpenFlow rule entries and push them to the OVS to update the flow table. Therefore, the following suspect traffic matching the newly updated flow table entries will be swiftly handled with valid countermeasures in the data plane of the OVS with line rate.

Our design architecture is detailed in the figure below. It includes the ability to use the single controller to control multiple Open Flow switches, and receive alerts from multiple Snort IDS sensors in order to correlate these alerts and make intelligent rules based on this data. Each network segment will have a Snort sensor analyzing traffic

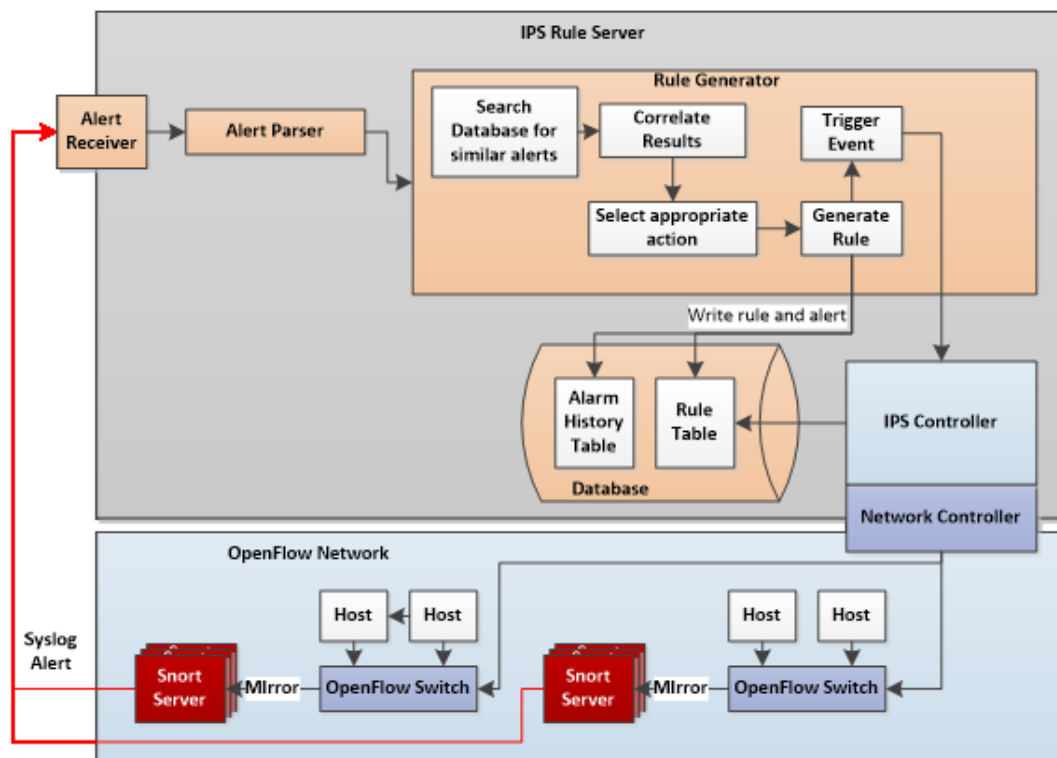


Figure 4.3: Proposed System Design

passing through the OpenFlow Switch. This will capture each packet on the network, regardless of the source, destination, or attack goal. For this project, we will assume all VMs on the cloud environment are of equal criticality an attackers goal would simply be to compromise a system using any given exploit. The actual detection of malicious activity will not be improved upon; we will be using default snort rules to perform the detection during our evaluation.

Chapter 5

ALGORITHM

1. check the ip address of source(h1) in an event.
2. if ($h1_ip == "tcp" || h1_ip == "udp"$) then $src_port = parsed.srcport$ and $dst_port = parsed.dstport$
3. Execute the query(drop)
display the dropped packet
Execute the query(quarentine)
display the quarentined host
Execute the query(blockport)
display the blocked port
Execute the query(redirect)
4. if $dpid(switch_id)$ is not in arp_table
add entry(mac) to the arptable
5. if instance(packet,ipv4)
if srcip in arpTable
return re-learned
else
update arpTable

```
if dstaddr in arpTable
    if prt==inport
        "source and destination port are same"
    else
        append actions to openflow
6. //arp table for packets with unknown destination addresses
    else if arp_for_unknowns:
        if (dpid,dstaddr) not in lost buffer
            update the bucket with entry(dpid,dstaddr)
7. //arp table for spam packets, specify an expiry time
    if outstanding_arps
        outstanding_arps(dpid,dstaddr)= time()+ expiry_time
8. elseif instance(packet.next, arp)
    update arpTable for the packet using arp protocol
9. generate fakeways:
    if arp_for_unknowns is None:
        update arp_for_unknowns = len(fakeways)
    else
        update arp_for_unknowns = str_to_bool(arp_for_unknowns)
```

Chapter 6

IMPLEMENTATION

The setup environment is needed for performing each task of the project. The different tasks are implemented by the following steps mentioned.

Task 1: Build test lab

The goal of this task is to find build a reliable experiment environment to simulate the Software Defined Network and installing some necessary software on it like databases and snort. We selected MySQL[18] as default database to store the information generated by switch and provide query function for controller in later tasks. A host running snort and having all traffic on the switch spanned to it is used to detect the traffic in the network and report unusual traffic via a syslog UDP message to the centralized IPS server. The necessary steps to build the topology are the following:

1. Install a virtualization software such as VMWare or virtual box
2. Download the MININET VM image from: <http://mininet.org/download/>
3. Extract and open the MININET VM image using the virtualization software.
4. On the MININET system

4.1 Install snort [19]

```
sudo apt-get install snort
```

4.2 Configure software (in this example, the centralized IPS server is at 192.168.2.5)

rsyslog.conf

```
sudo vi /etc/rsyslog.conf
```

```
auth. 192.168.238.128:5514
```

4.3 Restart service

```
sudo service rsyslog restart
```

4.4 Create a mininet environment

```
sudo cp ~/mininet/mininet/examples/nat.py
```

```
sudo vi ~/nat.py
```

```
add a line before CLI(net): net.addLink(net.hosts[0], net.switches[0])
```

4.5 Start MININET

```
sudo ~/nat.py
```

5. In MININET CLI:

```
xterm h1
```

5.1 On h1 terminal:

```
sudo snort -c /etc/snort/snort.conf -i h1-eth0 -l /var/log/snort/ -D -s #run syslog server  
on 192.168.2.5:5514
```

6. Design the database on MySQL

6.1 Create database snort

```
create database snort;
```

```
use snort;
```

6.2 Create the table alert_history

```
create table alert_history (timestamp datetime, rule varchar(12), message varchar(100),  
src varchar(15), srcport varchar(5), dst varchar(15), dstport varchar(5), classification  
varchar(30), priority varchar(2));
```

6.3 Create table rules

```
create table rules (timestamp datetime, src varchar(15), srcport varchar(5), dst var-  
char(15), dstport varchar(5), response varchar(20), redirect varchar(15));
```


7. Upload python files on controller[20]

Upload syslog-server.py and controller.py to \$POX_DIR/pox/ext

After these steps, the basic lab environment is built and the when you ping h2 to h3, h1 should send a syslog alert to the IPS Rule Server .

Task 2. Implement IPS rule server.

In this project, an IPS rule server is designed by following steps.

Task 2.1: An alert receiver was created by creating a SnortAlertServerWorker class that extends IOWorker. The init function of this class creates a listener on UDP port 5514, which will receive snort alerts through the snort devices syslog functionality. When a message is received, it calls the `_do_recv` function, which first acts as the alert parser.

Task 2.2: The alert parser takes the syslog alert message and then extracts important items from it that are to be stored in the database. The syslog alert structure is standardized, which makes it simple to extract specific data, as well as name/value pairs that can prove to be useful.

The timestamp will always occur first. Following the timestamp is the source of the syslog, followed by the snort rule that was triggered, followed by the rule description, then some name/value pairs, then the packet type, and finally source and destination IPs (as well as ports if ports are involved). As the alert is fairly standardized, it was simple to create a parser that can take in any number of name value pairs, as that is the most varying piece of the alert. Once the parser extracts all these values and places them into a python dictionary variable, this variable is passed on to the rule generator.

Task 2.3: We use MySQL as a default database to store the information of historical alerts and generated rules.

The table schemas are as follows:

Alert_History Table

timestamp: This is the timestamp the alert took place, to be used for correlation and analysis with other alerts.

rule: This is the rule ID that triggered the alert.

message: This is the rule description of the alert that was triggered. This is human readable and potentially useful for correlation analysis.

src: This is the source IP of the device that triggered the alert.

srcport: This is the source port of the device that triggered the alert.

dst: This is the destination IP of the device that triggered the alert.

dstport: This is the destination port of the device that triggered the alert.

classification: snort rules are classified into specific categories that describe the alert, such as attempted denial of service or successful privilege escalation. This is quite useful for correlation and analysis of snort alerts.

priority: This is the priority given by the snort rule.

Rules Table timestamp: This is the date and time the rule was created. This is used to expire the rule after some amount of time.

src: This is the source IP that the rule will be applied against. If the IP is ANY, or 0.0.0.0, the rule will be applied to all source IPs.

srcport: This is the source port that the rule will be applied against. If the IP is ANY, or 0, the rule will be applied to all source ports.

dst: This is the destination IP that the rule will be applied against. If the IP is ANY, or 0, the rule will be applied to all destination IPs.

dstport: This is the destination port that the rule will be applied against. If the port is ANY, or 0, the rule will be applied to all destination ports.

response: This is the action that will be taken by the rule to include actions such as drop, quarantine, blockport, redirect

redirect: if the action is redirect, this is the IP the traffic will be redirected to, such as a honeypot.

Task 3: Implement IPS on controller

The task 2.5 has implemented the basic communication among the POX event and the controller and IPS rule generator. At the controller side, the rule generator generates new rules and inserts the rules to database, besides the database updates themselves periodical.

Once it detects something unusual, the controller will query the database and send the action to the switch to operate on the stream. On the Task 3, we enhance this function and make sure the communication works properly.

Task 4: Evaluation the system

This IPS system works properly which listens, analyzes, parses and stores the alerts detected by snort and generates new rules. The functions of this intrusion prevention system are same as what we designed in the proposal reports.

In the respect of prevention, the switch strictly obeys the instruction from the controller, redirection, denying or dropping the packet.

In the respect of controlling, the controller generates and pushes new actions to the switch and updates the policies in time. Once a new policy published, the switch will be noticed and follows the new set of policy.

Chapter 7

RESULT ANALYSIS

7.1 System Specifications

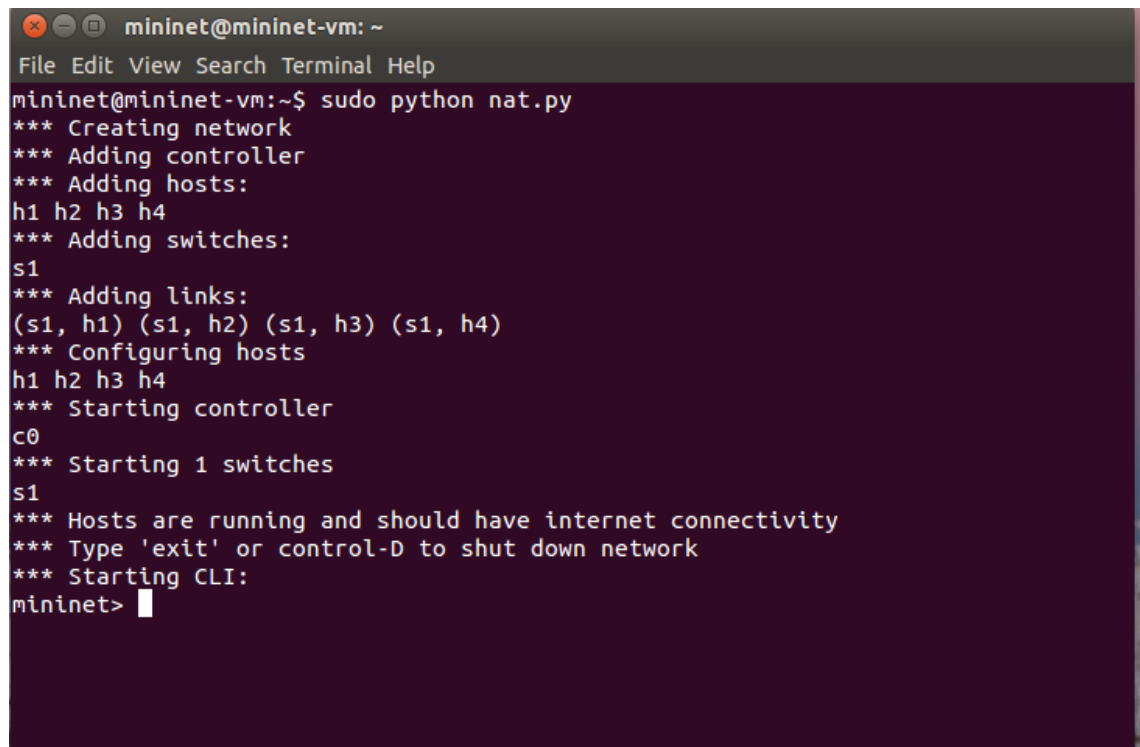
In the terms of configuration, the networks configuration should be configured properly when using the VMWare to run the MININET and controller. The type of networks should be set in bridged networks, otherwise the controller VM and MININET VM cannot know each other and the two VM will be use a same IP address. VMWare does not have this issue. There is an additional software needed installed for Python connecting MySQL. Besides, the root user should be granted all privileges to access database stored on the MySQL when run the system. Using root is not ideal, but as POX required to run under root, this was the easiest way to implement user based authentication with the MySQL server. A more production-ready system should put more effort into better securing this design, such as by using a dedicated server, or using an OpenFlow language that doesnt require to run under root.

Systems with:

8 GB RAM Dual Core 2.5 Ghz Processor 100 GB Hard Disk Ubuntu 14.04 (LTS)
with Python Mininet NICs Virtualization enabled

Software Requirements:

The software we will need for this project includes : Git MININET MySQL Database
POX Controller Snort IDS Linux OS Vmware Workstation



```
mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo python nat.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1
*** Hosts are running and should have internet connectivity
*** Type 'exit' or control-D to shut down network
*** Starting CLI:
mininet>
```

Figure 7.1: Mininet nat.py starting

7.2 Output Screens

The following section depicts screenshots of the current functionality of the system. First, in Figure 7.1 , we have started MININET with the nat.py script. We then open up host 1 (h1) in an xterminal. The Snort IDS is running on the host 1 (h1).

Figure 7.2 is the xterminal window for host 1. Here, we start the snort service as a Daemon, then ping a different IP. The default rules in snort contain a rule that alerts on pinging, which makes it useful for testing, and we utilize to verify our end to end system quite often.

Now once, the alerts are received, they are sent to the snort database. The below figure shows the alert_history table.

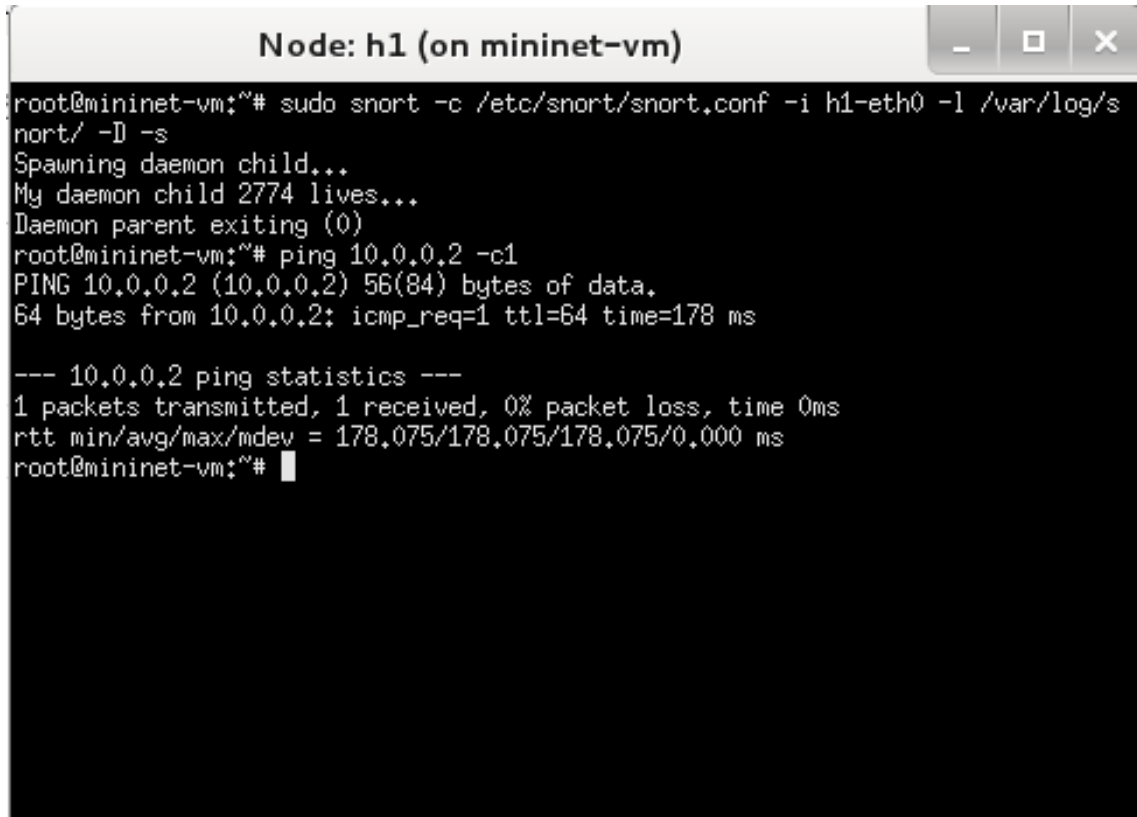


Figure 7.2: Starting Snort and Pinging a device to test snort

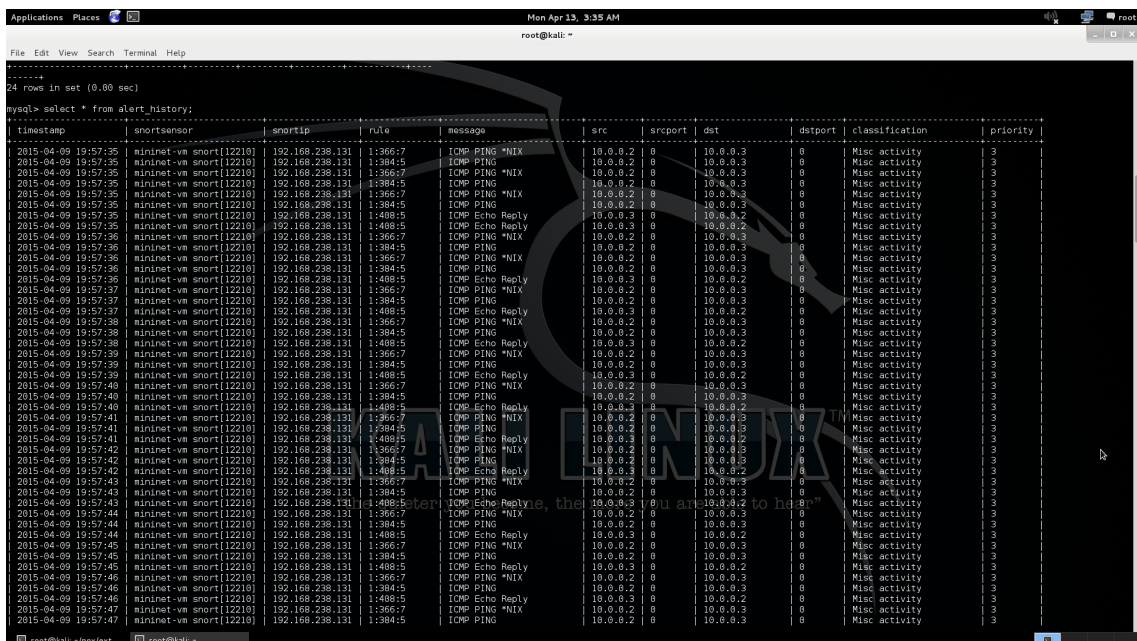


Figure 7.3: Alert History Table

```

2015-04-12 00:40:31 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:31 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:40:42 | mininet-vn snort[19865] | 192.168.238.131 | 1:366:7 | ICMP Ping *NIX | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:42 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:42 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:40:43 | mininet-vn snort[19865] | 192.168.238.131 | 1:366:7 | ICMP Ping *NIX | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:43 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:43 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:40:44 | mininet-vn snort[19865] | 192.168.238.131 | 1:366:7 | ICMP Ping *NIX | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:44 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:44 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:40:45 | mininet-vn snort[19865] | 192.168.238.131 | 1:366:7 | ICMP Ping *NIX | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:45 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:45 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:40:46 | mininet-vn snort[19865] | 192.168.238.131 | 1:366:7 | ICMP Ping *NIX | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:46 | mininet-vn snort[19865] | 192.168.238.131 | 1:384:5 | ICMP Ping | 10.0.0.2 | 0 | 10.0.0.3 | 0 | Misc activity | 3
2015-04-12 00:40:46 | mininet-vn snort[19865] | 192.168.238.131 | 1:488:5 | ICMP Echo Reply | 10.0.0.3 | 0 | 10.0.0.2 | 0 | Misc activity | 3
2015-04-12 00:42:18 | mininet-vn snort[19865] | 192.168.238.131 | message rep :527: | 0.0.0.0 | 68 | 255.255.255.255 | 67 | Potentially Bad Traffic | 2
298 rows in set (0.00 sec)

mysql> select * from rules;
+-----+-----+-----+-----+-----+-----+-----+
| timestamp | src | srcport | dst | dstport | response | redirect |
+-----+-----+-----+-----+-----+-----+-----+
| 2015-04-12 03:40:39 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:39 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:40 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:40 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:40 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:41 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:42 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:42 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:43 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:43 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:43 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:43 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:44 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:44 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:45 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:45 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:45 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:46 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:46 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:47 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
| 2015-04-12 03:40:47 | 10.0.0.2 | 80 | 0.0.0.0 | 80 | blockport | NULL |
24 rows in set (0.00 sec)

mysql>

```

Figure 7.4: Rules Generated by Snort

Rules table

Once everything is setup we tested the setup against a general rule in which every ping is considered as a miscellaneous activity and the subsequent port is blocked (though the ping is generally allowed, we blocked it just for the testing phase).

System before the port 8080 was blocked

Initially, host 1 can connect to the open port 80 on host 2 with no problems.

We used the ping snort rule to cause a Misc Activity detection, which we then configured to perform our various tests (otherwise its difficult to trigger snort rules intentionally). In this example, we configured the misc-activity detection to block port 80 between the two servers.

The ping is detected ,and the controller identifies the source IP , and the port number. We use telnet to successfully test connection between the two hosts via the 80 port.

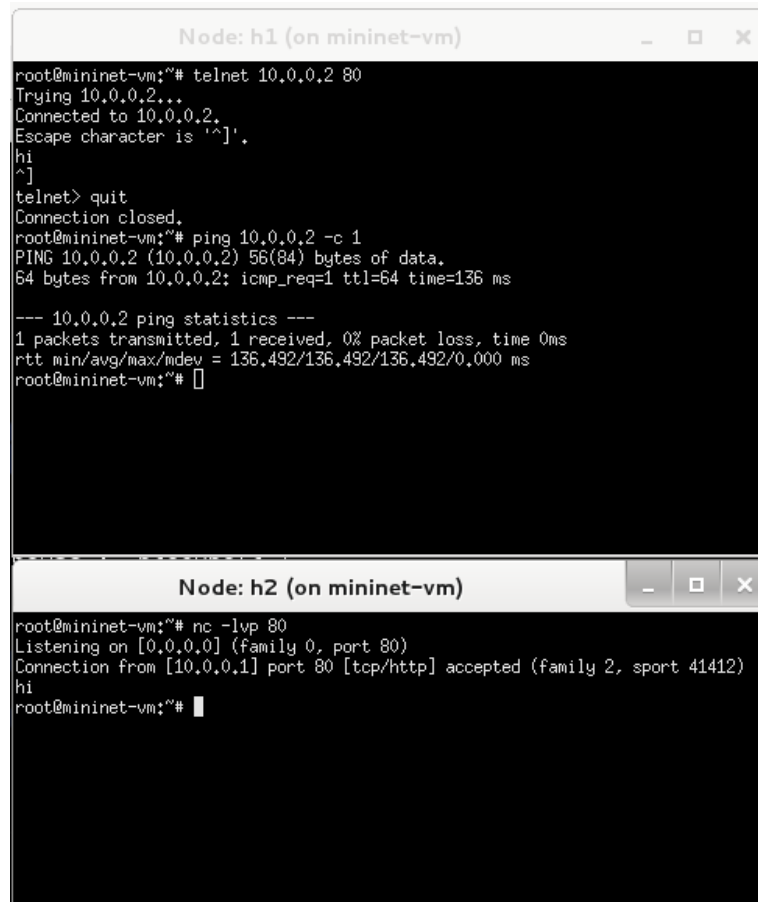
```
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openFlow.of_01:[00-00-00-00-00-01 1] connected
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Event Start-----
<controller.l3_switch object at 0x1f19190>
<<ServerWorker>: {'srcip': '10.0.0.1', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
-----Handle Packet-----
-----Event Start-----
<controller.l3_switch object at 0x1f19190>
<<ServerWorker>: {'srcip': '10.0.0.1', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
-----Event Start-----
<controller.l3_switch object at 0x1f19190>
<<ServerWorker>: {'srcip': '10.0.0.2', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
```

Figure 7.5: Host1 pings Host2,Snort detects port 8080

After the Controller queries the rules table and then finally blocks the port 80 , we will see that no communication will take place between the two hosts.

After the controller blocks the port.

Thus here we can successfully see that the Controller has blocked the target port as specified in the rules table.



```

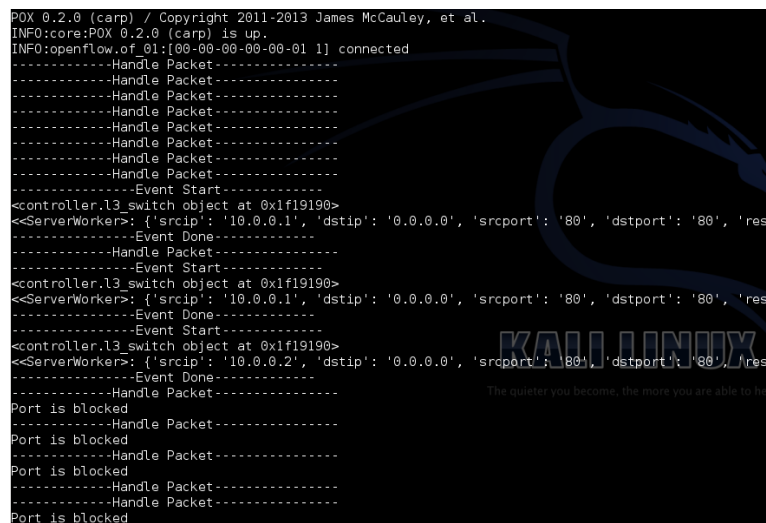
Node: h1 (on mininet-vm)
root@mininet-vm:~# telnet 10.0.0.2 80
Trying 10.0.0.2...
Connected to 10.0.0.2.
Escape character is '^'.
hi
^]
telnet> quit
Connection closed.
root@mininet-vm:~# ping 10.0.0.2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=136 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 136.492/136.492/136.492/0.000 ms
root@mininet-vm:~# █

Node: h2 (on mininet-vm)
root@mininet-vm:~# nc -lvp 80
Listening on [0.0.0.0] (family 0, port 80)
Connection from [10.0.0.1] port 80 [tcp/http] accepted (family 2, sport 41412)
hi
root@mininet-vm:~# █

```

Figure 7.6: Host1 and Host2 communicate via telnet



```

POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Handle Packet-----
-----Event Start-----
<<ServerWorker>: { 'srcip': '10.0.0.1', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
-----Handle Packet-----
-----Event Start-----
<<ServerWorker>: { 'srcip': '10.0.0.1', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
-----Event Start-----
<<ServerWorker>: { 'srcip': '10.0.0.2', 'dstip': '0.0.0.0', 'srcport': '80', 'dstport': '80', 'res
-----Event Done-----
-----Handle Packet-----
Port is blocked
-----Handle Packet-----
Port is blocked
-----Handle Packet-----
Port is blocked
-----Handle Packet-----
-----Handle Packet-----
Port is blocked

```

Figure 7.7: Host1 and Host2 no longer able to communicate

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENTS

In this project, we propose a SDN-based IDPS development framework in cloud computing environment. We proposed implementing an IPS as a way of securing infrastructure in cloud environment, which combines the detection and prevention abilities of snort. We assume that every node is equally critical and equally vulnerable. We have completed all the four tasks: building out the environment, implementing the IPS Rule Server to include the Alert Receiver, the Alert Parser, the database, and the Rule Generator, implement the IPS on the controller. We achieved it by implementing a design based on the SnortFlow architecture, which comprises of establishing snort intrusion prevention system over the openflow switches. Furthermore, additional capabilities such as scalability, rule updating, synchronization and introduction of preventative measures was implemented. Through analysis the alerts, system is able to generate new rules and update them, utilizing a database to store all the rules and historical alerts. We are comparing a normal intrusion prevention system against an SDN based intrusion prevention system.

Implementing a neural network based intrusion detection system can greatly enhance the level of security provided at the control plane. We believe that denial of service

and other network-based attacks leave a faint trace of their presence in the network traffic data. Ours is an anomaly detection system that detects network-based attacks by carefully analyzing this network traffic data and alerting administrators to abnormal traffic trends. It has been shown that network traffic can be efficiently modeled using artificial neural networks[21]. Therefore we further aim to use neural networks to examine network traffic data.

Bibliography

- [1] R. U. Rehman, *Intrusion detection systems with snort*. in BRUCE PERENS OPEN SOURCE SERIES.
- [2] C. C. S. Alliance, “Top threats to cloud computing v1.0,” *White Paper*, 2010.
- [3] Yogita Hande, Aishwarya Jadhav, Achaleshwari Patil, Rutuja Zagade, “Software defined networking with intrusion detection system,” *International Journal of Engineering and Technical Research (IJETR)*, Volume-2, vol. Volume-2, Issue-10, October 2014.
- [4] P. Porras et al., “A security enforcement kernel for open-flow networks,” *FRESCO: Modular Composable Security Services for Software-Defined Networks*.
- [5] Tianyi Xing , Zhengyang Xiong , Dijiang Huang and Deep Medhi, “Sdnips: Enabling software-defined networking based intrusion prevention system in clouds,”
- [6] Jafar Haadi Jafarian ,Ehab Al-Shaer and Qi Duan, “Openflow random host mutation: transparent moving target defense using software defined networking,” *Proceedings of the first workshop on Hot topics in software defined networks*, pp. Pages 127–132.
- [7] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, “Nice: Network intrusion detection and countermeasure selection in virtual network systems,” *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Special Issue on Cloud Computing Assessment, pp. Pages 127–132, 2013.

- [8] Tianyi Xing , Zhengyang Xiong , Dijiang Huang and Deep Medhi, “Sdnips: Enabling software-defined networking based intrusion prevention system in clouds,” *IEEE Transactions on Dependable and Secure Computing (TDSC), Special Issue on Cloud Computing Assessment*.
- [9] Zhengyang Xiong, “An sdn-based ips development framework in cloud networking environment,”
- [10] Yung-Li Hu¹, Wei-Bing Su¹, Li-Ying Wu, Yennun Huang and Sy-Yen Kuo, “Design of event-based intrusion detection system on openflow network,”
- [11] Nick Feamster , Jennifer Rexford and Ellen Zegura, “The road to sdn: An intellectual history of programmable networks,”
- [12] Openflow switch specification version 1.4.0.[Online], “<https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>,”
- [13] Wikipedia, Cloud Computing. [Online]., “Available: http://en.wikipedia.org/wiki/cloud_computing,”
- [14] OpenFlow, “Available: http://archive.openflow.org/wk/index.php/openflow_tutorial,”
- [15] K. Vieira, A. Schuler, C. Westphall, and C. Westphall, “Intrusion detection for grid and cloud computing,” *IT Professional*, vol. vol. 12, no. 4, p. pp. 3843, July 2010.
- [16] Mininet. [Online]., “Available: <http://mininet.org/>,”
- [17] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, “Snortflow: A openflow-based system in cloud environment,” in *GENI Research and Educational Experiment Workshop, GREE*, 2013.
- [18] MySQL. [Online]., “Available: <http://www.mysql.com>,”
- [19] SourceFire Inc., “Available: <http://www.snort.org>,”
- [20] POX Conroller[Online], “Available: <http://sdnhub.org/tutorials/pox/>,”

-
- [21] A. Bivens, C. Palagiri, R. Smith, Boleslawszymanski, M. Embrechts, “network-based intrusion detection using neural networks,” vol. vol.12, ASME Press, New York, pp. pp 579–584, 2002.