An SDN-based IPS Development Framework in Cloud Networking Environment

by

Zhengyang Xiong

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved July 2014 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Guoliang Xue
Hasan Davulcu

ARIZONA STATE UNIVERSITY

August 2014

ABSTRACT

Security has been one of the top concerns in cloud community while cloud resource abuse and malicious insiders are considered as top threats. Traditionally, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) have been widely deployed to manipulate cloud security, with the latter one providing additional prevention capability. However, as one of the most creative networking technologies, Software-Defined Networking (SDN) is rarely used to implement IDPS in the cloud computing environment because the lack of comprehensive development framework and processing flow. Simply migration from traditional IDS/IPS systems to SDN environment are not effective enough for detecting and defending malicious attacks. Hence, in this thesis, we present 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. A Traffic Information Platform (TIP) is proposed as the cornerstone with several upper layer security modules such as Detection, Analysis and Prevention components. Benefiting from the flexible, compatible and programmable features of SDN, Customized Detection Engine, Network Topology Finder, Source Tracer and further user-developed security appliances are plugged in our framework to construct a SDN-based defensive system. Two main categories Python-based APIs are designed to support developers for further development. This system is designed and implemented based on the POX controller and Open vSwitch in the cloud computing environment. The efficiency of this framework is demonstrated by a sample IPS implementation and the performance of our framework is also evaluated.

DEDICATION

To my family.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor and committee chair Dr. Dijiang Huang, for his invaluable guidance and persistent help. I am fortune to have him as my advisor. I am pleased to thank my other committee members, Dr. Guoliang Xue and Dr. Hasan Dalvucu, for their helpful suggestions and comments. This thesis was made possible due to the masterly guidances of my committee members.

I would like to thank my colleague Tianyi Xing for mentoring me in this amazing research area. It is such a pleasure to work with him in last two years. I would also like to thank my friends, Chris Kawa and Qiangqiang Liu, for their encouragements and advices. I would never forget the help I got from them.

Finally, I would like to thank my parents, who support me and trust me all the time. I hope that this achievement will complete the dream that you had for me all those many years ago when you choose to give me the best education you could.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

1 INTRODUCTION 1

1.1 Overview 1

1.2 OpenFlow Technology 4

1.3 Motivation and Contribution 5

1.4 Organization 7

2 RELATED WORK 8

2.1 Traditional Security Solution for Cloud Computing Environment 8

2.2 SDN-based Network Security Solutions 10

2.3 SDN Security Extension 13

2.4 SDN-based IDPS 14

3 SYSTEM ARCHITECTURE 16

3.1 Comprehensive Defensive System Lifecycle 16

3.2 System Overall Architecture 16

4 TRAFFIC INFORMATION PLATFORM 19

4.1 Architecture of TIP 19

4.2 OpenFlow Monitor 20

4.2.1 OpenFlow Flowtable Request and Connection 20

4.2.2 Lightweight Anomaly Traffic Detection Engine 21

4.2.3 CUSUM Algorithm for Flooding Detection 23

4.3 Traffic Preprocessor 26

4.4 Traffic and System Information DS/DB 27

4.5 Traffic Packet Acquisition 30

CHAPTER Page

5 SDN-BASED IPS SECURITY MODULES 31
5.1 Detection Module 31
5.1.1 Snort Agent 31
5.1.2 Flow Expander 31
5.2 Analysis Module 33
5.2.1 Source Tracer 33
5.2.2 Topology Manager 35
5.3 Prevention Module 37
5.3.1 Network Reconfiguration 37
5.3.2 Representative NR Actions in Prevention Module 40
5.4 Python-based Security Application Development API 42
6 HOW TO BUILD A DEFENSIVE SYSTEM 45
6.1 Problem Analysis 45
6.2 Processing Flow 46
6.3 Implementation 47
7 EVALUATION 49
7.1 How SDN-based Countermeasure Improve Network Security 49
8 CONCLUSION 56
REFERENCES 57

LIST OF TABLES

Table Page

2.1 Security Solutions Comparison Table 15

4.1 Openflow Statistics Information Request Type 21

5.1 Network Reconfiguration Actions 39

5.2 TIP API Summarization 43

5.3 SDN-based Security Application Module API Summarization 44

LIST OF FIGURES

Figure		Page
1.1	OpenFlow Architecture	3
3.1	Comprehensive Lifecycle	16
3.2	System Overall Architecture.	17
4.1	Traffic Information Platform	20
4.2	Openflow Statistical Request Procedure	22
5.1	Tree-based Rule Auto Spanning Model	33
5.2	Source Tracer for Spoofing IP	34
5.3	Topology Finder Procedures	38
6.1	DDoS Defensive Solution Construction Flow	46
6.2	DDoS Detection Engine Implementation Screenshot	47
6.3	Screenshot of Source Tracer in DDoS Attack Scenario	48
7.1	Comparison Evaluation between SDN-based and Traditional Mitigation	49
7.2	Major Mitigation Options Evaluation	50
7.3	TR Traffic Handle Capacity	52
7.4	Health Traffic Impact	53
7.5	Bandwidth Performance of QA	54

Chapter 1

INTRODUCTION

1.1 Overview

Cloud computing refers the use of computing resources like hardware and software which can be delivered as a service over a network [1]. 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 user's data, information and computing resources are stored in cloud infrastructures. It has been widely deployed today 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.

Traditionally, Intrusion Detection Systems (IDS) such as Snort [2] are regarded as the common tools to detect and prevent malicious attacks within a networking system. They monitor network events and traffic to identify malicious activities, and then issue alerts and report to system administrators. The 'detection and alerting' nature of current IDS solutions demands the cloud security team to hire professional security experts. Moreover, the current cloud IDS lacks of proactive capability to prevent attacks at its initial stage. Thus, Intrusion Prevention Systems (IPS) is preferred over IDS in order to automatically take action towards the suspect network activities. Basically, the IPS can be constructed based on IDS because the detection

function is needed in an IPS solution. However. most existing IPS solutions are designed for traditional network and simple migration is not effective enough to detect and defend malicious attacks. There are several issues in the current traditional IPS system:

1. *Latency*: in-bound IPS requires inspection and blocking on each network packet, some IPS detection mechanism even need to mirror network traffic for sniffer to investigate packet content. This definitely degrades the performance and results in a high latency. For some real-time cloud services, it might not be appropriate to apply this IPS in the cloud.

2. *Flexibility*: Traditional IPS solution only provides simple prevention actions, which is *block*. However, when detection system can not confidently confirm the happen of attacking, simple block action is not the suitable choice for a cloud security solution. For a suspect traffic flow, QoS or functionally redirection may be the better countermeasures than drop.

3. *Flexibility*: Because of the characteristics of cloud computing, dynamical configuration, service-oriented function and user-friendly deployment are very necessary for an IPS in the cloud and traditional IPS contradicts with these features.

4. *Extensibility*: Many existing IPS solutions are not extensible. Even they are open source softwares, different coding styles, development environments and interfaces make customized function really hard to be deployed in the system.

Software-Defined Networking (SDN), as one of the most creative network technologies, makes security detection and prevention more agile and dynamical in responding malicious behaviors [3, 4]. Main idea of SDN is to decouple control plane and data plane to *open* traditional networking devices. This idea evolves networking

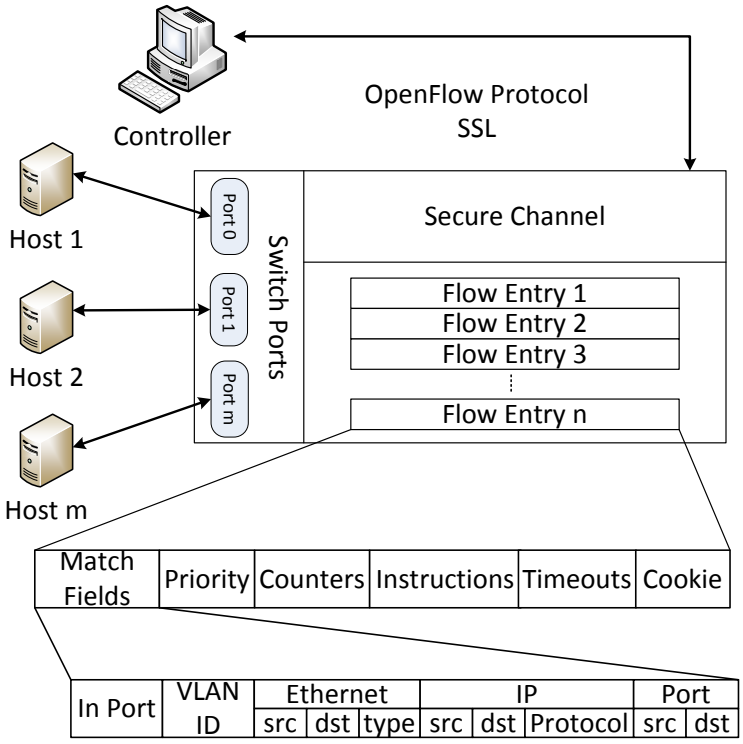


Figure 1.1: OpenFlow Architecture

technologies to a brand new level. SDN enables us to define and configure our network by programming or using software to define it. OpenFlow is the most representative protocol implementing the SDN concept. SDN-enabled devices could be manipulated by OpenFlow. 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. As we see, the programmable network interface gives us a great opportunity to define and configure network in an extremely flexible and efficient way. Thus, the emerging of SDN introduces an innovative approach to protect network with both flexibility and compatibility, which is a great fit for cloud environment.

1.2 OpenFlow Technology

OpenFlow is the most representative standard and protocol implementing SDN concept. This novel technology significantly improves the way of how current networking devices work. As shown in Fig. 1.1, it introduces a centralized and separate controller. This controller uses defined openflow interfaces [5] to manipulate networking on the control plane. The flow table in the figure is located in the data plane. It is able to handle incoming packets in line rate. It has match, priority, counters, instructions, timeouts and cookie fields. Match fields play a huge role in the openflow. We can find in port, vlan id, ethernet information, ip information and switch port information from match fields. All these fields work together to identify and define a traffic flow. The action field is used to guide how to handle a specific traffic flow, such as drop, forward or flood. The statistics information also can be found in the flowtable, all the traffic passing through the openflow-enable switch will be recorded by the table. OFPacketIn, OFPacketOut and OFFlowmod are very frequent openflow events, controller can listen to them and response to these events. The OFPacketIn counters are further classifies into separate broadcast, multicast, unicast counters and different Layer 3 and Layer 4 protocol counters to give the visibility of the types of flows in the network. It also keeps track of the number of active flows per switch. In the OpenFlow architecture, a controller executes all control tasks of the switches and is also used for deploying new networking frameworks, such as new routing protocols or optimized cross-layer packet-switching algorithms.

When a packet arrives at an OpenFlow switch (OFS), the switch processes the packet in the following three steps:

1. It checks the header fields of the packets and tries to match any entry in the local flow table. If there is no any matching entry in the flow table, the packet

will be sent to the controller for further processing. There can be multiple flow control rules. It follows a best matching procedure to pick the best rule, e.g., the one with the highest priority.

2. It then updates the byte and packet counting information associated with the rules for statistics logging purposes, which will be used to collect the traffic information to detect any anomaly traffic.

3. Once a matching rule is decided, the OpenFlow switch takes the action based on the corresponding flow table entry, e.g., forwarding to a specific port, or dropping.

Open vSwitch[6] is a production quality, multilayer open virtual switch. It has been widely planted into major cloud orchestration system such as XenServer [7], OpenStack [8], CloudStack [9], to replace the traditional Linux bridge to serve as the network back end implementation. It enables the implementation of OpenFlow in the virtual networking environment and makes OpenFlow development easier.

1.3 Motivation and Contribution

As we discussed before, SDN technology opens a new chapter for network security because of its flexibility, extensibility and feasibility. SDN-based security approaches have been considered as a trend for future network security solutions [10].

Current OpenFlow related works like OpenNetMon [11] and OpenSafe [12] both propose the network monitoring service in cloud environment to efficiently collect the traffic usage statistics and detect the malicious activities. However, they do not propose a comprehensive solution for cloud system. These works do not go beyond the stage of detection and are not able to provide further analysis and countermeasures for any attack. The 'detecting and alerting' nature of monitoring solutions demands

the human-in-the-loop to inspect the generated security alerts and manually take actions, which can not respond to attacks in a prompt fashion.

OpenFlow Random Host Mutation (OFRHM) [13] proposes a countermeasure by using Moving Target Defense (MTD) to protect the targets from being attacked through changing its identity representation, i.e., a mapping between the real internal IP and extremal floating IP. But it only works in a proactive fashion and does not work for malicious insider who knows the real internal IP address of victims.

The reason current SDN-based solution is not automated and comprehensive because SDN-based development framework is rare. Fresco [14] is the most famous existing SDN security development framework. It facilitates the rapid design and modularize OpenFlow-based monitor and mitigation. However, this framework still could be improved because: (1) The detection capability in this framework is not strong because this function is developed based on simple traffic counter. (2) Fresco designs a script language for developers to code their own security applications. Although this new script language makes development easier, it restricts the flexibility of deep extension. (3) Most of Fresco interfaces are single directional, it can not help third part security application to efficient fetch traffic packet and perform proper actions. Hence, in this thesis, we propose a new SDN-based IPS development framework in cloud computing environment as it is a key demand in this research area.

This framework provides following unique features:

- Propose three security modules for SDN-based security application development. Detection, analysis and prevention modules can work together to build a defensive system in cloud environment.

- Design a Traffic Information Platform (TIP) as the cornerstone of the framework to monitor traffic information. It classifies traffic and stores them into variety

databases. It sits in the middle of lower layer SDN devices and upper layer customized security applications. It has interfaces to communicate with data planes and export traffic information to applications.

- Develop a set of Python-based APIs to facilitate SDN-based security application development in cloud system.

- Develop default security applications as guideline for development. Integrated Snort to the framework as default detection engine, build a Topology Manager to discovery and manipulate network architecture. Create a countermeasure pool for system to defend different attacks in variety situations.

1.4 Organization

This thesis is organized as follow. Chapter 2 introduces the related works of traditional cloud computing security and SDN-based solutions. Chapter 3 gives the overview of the whole framework. Chapter 4 elaborates the architecture of TIP. Chapter 5 describes the three security modules and how default security applications are build in the system. Chapter 6 gives a sample of how to build a defensive system in the cloud based on our framework. Chapter 7 conducts the performance of SDN-based security applications and chapter 8 concludes the results of this research.

Chapter 2

RELATED WORK

The related and representative security solutions in cloud system are discussed in the following fashion: we first discuss the traditional security solutions for cloud computing and then we investigate the SDN-based security solutions and its extensions. At last, we discuss current SDN-enabled IDS/IPS solutions.

2.1 Traditional Security Solution for Cloud Computing Environment

Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) are traditional efforts to monitor and secure cloud computing system. As an efficient security appliance, IDS can utilize signature-based, statistical-based and stateful protocol analysis methods to achieve its detection capability. Furthermore, IPS can respond detected threats by triggering variety prevention actions to tackle malicious activities.

[15] is a Host-based intrusion detection system which deploys IDS to each host in the cloud computing environment. This design enables behavior-based technique to detect unknown attacks and knowledge-based one to identify known attacks. However, the data is captured from system logs, services, and node messages. It can not detect any intrusion which running on the VM. Another architecture is provided in [16] to detect the vulnerability in the virtual machines. It extends the capability of typical IDS to incorporate virtual machine based detection approach to the system. The provided IDS management can detect, configure, recover and prevent the VMs from visualization layer threats. However, it is not a lightweight solution.

A stateful intrusion detection system is introduced in [17]. This paper applies a slicing mechanism to divide overall traffic into subsets of manageable size and each

subset contains enough evidences to detect a specific attack. As a distributed network detection system, after the system is configured, it is not easy to reconfigure and migrate detection senors for users on demand.

In [18], authors introduce an effective management model to optimally distribute different NIDS and NIPS across the whole network. This work differs from single-vantage-point view NIDS/NIPS placement, it is a scalable solution for network wide deployment.

Packet marking technique is widely used for IP traceback even tracing back the source of attacks is extremely difficult. In [19], the authors present a marking mechanism for DDoS traceback, which injects an unique mark to each packet for traffic identification. As a Probabilistic Packet Marking(PPM) method, it has a potential that leads attackers to inject marked packet and spoofed the traffic. [20] is another important traceback method by using Deterministic Packet Marking(DPM). The victim could track the packets from the router which splits the IP address into two segments. Differ from previous methods, in [21], the authors present an independent method to traceback attacker based on entropy variations. However, most of these works do not handle the IP spoofing very well and packet modification is needed to implement these methods.

FireCol [22] is a dedicated flooding DDoS attacks detection solution implemented in traditional network system. In this design, IPSs are distributed in the network system to form several virtual protection rings with selected traffic exchange to detect and defend DDoS attacks. This collaborative system addresses the hardly detection problem and single IDS/IPS crashing problem under overwhelming attacking traffic. However, this method is not a lightweight solution such as [23], the flexibility and dynamism is limited in this system and the deployment and management is also complicate.

Similar to the FireCol, [24] presents a multiple layers game-theoretic framework for DDoS attack and defense evaluation. An innovative point in this work is the strategic thinking of attacker's perspective benefit the defense decision maker in this interaction between attacks and defenses. However, this framework is not suitable for deploying dynamic network threats countermeasures and has no real time security solution for real time attack.

Data mining technology is normally applied in flooding attack detection. [25] uses an incremental mining approach to detect large-scale attacks for network intrusion detection system. This work is different from other works because traffic information is obtained in real time and the analysis is not done by static data. This dynamic approach with fuzzy associations rules enable system to detect anomaly activities and make a decision every two seconds, but it is not a adaptive system, which can not provide countermeasure in real time.

2.2 SDN-based Network Security Solutions

SDN has been well researched to establish monitoring system [26] [12] [11] due to its centralized abstract architecture and its statistics capability. OpenSafe[12] is a network monitoring system that allows administrators to easily collect usage statistics of networking and detect malicious activities by leveraging programmable network fabric. It uses OpenFlow technique to enable some manipulations of traffic, such as selective rules matching and arbitrary flows directing, to achieve its goal. Furthermore, ALARMS is designed as a policy language to articulate paths of switches for easily network management. OpenNetMon [11] is another approach for network monitoring application based on OpenFlow platform. This work is implemented to monitor per-flow metrics to deliver fine-grained input for traffic engineering. Benefiting from the OpenFlow interfaces that enable statistics query from controller, authors proposed

an accurate way to measure per-flow throughput, delay and packet loss metrics. In [26], authors proposed a new framework to address the detection problem by manipulating network flows to security nodes for investigation . This flow-based detection mechanism guarantees all necessary traffic packets are inspected by security nodes. For dynamism purpose, provided services could be easily deployed by users through a simple script language. However, all three studies above do not further discuss the countermeasures for intrusion malicious activities but only provide the monitoring services.

FortNox[27] is a Security Enforcement Kernel to address the conflict of rule to secure OpenFlow network. Different rules are inserted by various OpenFlow applications can generate rule conflicts, which has potential to allow malicious packets bypass the strict system security policy. FortNox applies a rule reduction algorithm to detect conflicts and resolves a conflict by assigning authorization roles with different privileges for the candidate flow rule. This kernel overcomes the potential vulnerability of OpenFlow rules installment and enables an enforceable flow constraint to enhance SDN security. It is considered as a SDN extension like [28] [29]. The authors' another research, Fresco, [14] implements an OpenFlow Security Application Development Framework based on Security Enforcement Kernel. It encapsulates the network security mitigations in the framework and provides APIs to enable legacy application to trigger FRESCO module. However, this work does not have the capability to defend and protect network assets independently because predefined policies are needed to drive this system.

SDN technology is also used to detect specific threat, e.g., DDoS, in cloud system. In [23], the authors present a flow-based lightweight method for DDoS attack detection. This solution is implemented on NOX/OpenFlow platform to collect traffic statistics information from flow table by Flow Collector. Important DDoS relevant

features will be extracted from feature extractor module and be sent to classifier where classification algorithm comes from Self Organizing Maps (SOM)[30] method. By deploying SOM, this classifier could classify normal and malicious traffic with very low overhead. This work is more lightweight than others that may require heavy processing in order to extract feature information needed for traffic analysis. However, this paper does not provide a corresponding countermeasure for the DDoS attack.

Besides detection, utilizing the nature of SDN to propose prevention solutions is another key direction for SDN based security research. CONA [31] is a Content-oriented Networking Architecture build on NetFPGA-OpenFlow platform. In this design, hosts request contents and agents to deliver the requested contents while the hosts can not. Under the content-aware supervision, system can perform prevention by: (1) collecting suspect flows information from others agents for analysis, and (2) applying rate limit to each of relevant agents to slow down the overwhelming malicious traffic.

OpenFlow Random Host Mutation (OFRHM) [13] is another innovative solution by using Moving Target Defense (MTD) to protect the targets from being attacked through changing its identity representation. As a common action, scanning attack is considered as the first step to discover vulnerability of the system when performing network malicious behavior. In this research, they add a transparent virtual IP layer above real IP and make real IP addresses untouchable by unauthorized entities. The system will assign a virtual IP to each host after each mutation interval. However, this work does not provide a forensic evidence or analysis for future threats detection and prevention, and does not work when the attack is initiated from malicious insiders who are aware of the real IP address of victims.

2.3 SDN Security Extension

With the emerging of Software-Defined Network, researchers start to concern the security of SDN itself. In [29], a replication mechanism is brought up to handle the weakness of centralized controlled network architecture, which is one single point of failure could lead a downgrade of network resilient for the whole system. CPRecovery component is able to update the flow entry in secondary controller dynamically and secondary controller can control the switch automatically when primary controller is down due to overwhelming traffic or DDoS attack. This work could be considered as a solution for DDoS attack, however, simple replication mechanism hardly promises that all the secondary controllers are able to tolerate high pressure attack even more backups could be deployed in this system.

AvantGuard [28] is a SDN extension, which enhances the security and resilience of OpenFlow itself. To address the two bottlenecks, scalability and responsiveness challenge, in OpenFlow, this paper introduces two new modules: connection migration module and actuating trigger module. The former component is efficient to filter incomplete TCP connections by establishing a handshake session before packets arriving the controller. TCP connections are maintained by migration connection module to avoid the threats of TCP saturation attack. Actuating trigger module enables the data plane to report network status and active a specific flow rule based on predefined traffic conditions. This research improved the robustness of SDN system and provide additional data plane information to control plane to acquire higher security performance.

2.4 SDN-based IDPS

As we talked before, IDS and IPS are critical security appliances to protect cloud computing network. When we apply SDN to the cloud system, the decoupled switch with separated control plane and data plane, creates a network OS layer to allow programmable interface and open network control. This feature leads a flexibility and dynamic network reconfiguration, which can efficiently and effectively control the network and enable security manipulation for higher level guards. However, only a small number of works are done to implement SDN-based IDS and even fewer works on SDN-based IPS.

L-IDS [32] is a Learning Intrusion Detection System to provide a network service for mobile devices protection. It is able to detect and respond to malicious attack with the deployment of existing security system. It is more like a network service which can transparently configured for end-host mobility and enable already known countermeasures to mitigate detected threats. The authors do not provide a comprehensive solution for detected attack and more evaluations are needed to figure the most efficient response action for threats.

In a recent work [3], we 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. The originality and contribution of this work main comes from utilizing the attack graph theory to generate a vulnerability graph and achieve the optimal decision result on selecting the countermeasure. SnortFlow [4] is another recent work focusing on the design of OpenFlow based IPS with preliminary result. However, both of our previous work did not address the following issues: (1) they did not consider the multi-tenancy issue that is the key characteristic for all major cloud platform. (2) generating the attack graph for cloud virtual topology is slow (minutes

Work	SDN	Cloud	Detection	Prevention
[25]			✓	
[24]			✓	
[33]		✓		
[19]		✓		✓
[20]		✓		✓
[21]		✓		✓
[26]	✓	✓	✓	
[12]	✓	✓	✓	
[11]	✓	✓	✓	
[23]	✓		✓	
[28]	✓	✓		✓
[13]	✓	✓		✓
[31]	✓		✓	✓
[3]	✓	✓	✓	
[4]	✓	✓		✓

Table 2.1: Security Solutions Comparison Table

level) and is not practical in a dynamic changing system. (3) they only evaluated the performance between placing the detection agent at user domain and management domain, which provide evidence for design this framework where detection agent is at the management domain and directly monitor the OVS.

We believe the dynamic and adaptive capability of the SDN framework could benefit the development of IDPS. This area is worth to be well explored for SDN-enabled cloud system to build suitable and on demand IDS/IPS system. Thus, we have been setting our research target on establishing the SDN-based IPS in cloud environment. As this thesis focus on providing a SDN-based IPS development framework, we summarized some of the works to show that lack of comprehensive IPS solution in cloud environment.All related work are summarized in Table 2.1.

Chapter 3

SYSTEM ARCHITECTURE

3.1 Comprehensive Defensive System Lifecycle

Motivated by the aforementioned research challenges, we are aiming to build a IPS development framework with a full lifecycle. The complete lifecycle can be divided intro three stages as shown in Fig. 3.1: *Traffic Monitor*, *Attack Analysis* and *Actions Execution*. In the traffic monitor stage, all traffic should be monitored through network monitoring approaches, e.g., sFlow, NetFlow, SPAN port, etc. Then, different algorithms or mechanisms are run to analyze the traffic pattern in order to determine what kind of this attack is. Finally, mitigation actions should be efficiently executed to prevent and tackle malicious activities in the cloud system.

3.2 System Overall Architecture

Fig. 3.2 is the framework of proposed SDN-based IPS Development System. It is composed by Data Plane Elements and OpenFlow Controller. All SDN enabled data plane elements including OpenFlow switch, Open vSwitch, additional virtual & physical devices are connected and controlled by a SDN Network Controller. It is implemented based on POX controller [34]. This component is designed to guarantee

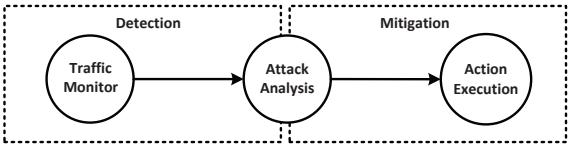


Figure 3.1: Comprehensive Lifecycle

the regular network routing and switching. All the daily network communication is operated and delivered by this controller. The proposed SDN Traffic Information Platform is designed on the top of SDN network controller. It grabs network view, detailed traffic information such as IP/MAC addresses, port number from lower layer Data Plane Elements. It also exposes several traffic and management related interfaces (e.g., data access, detail traffic statistics and operational functions) to upper layer security applications.

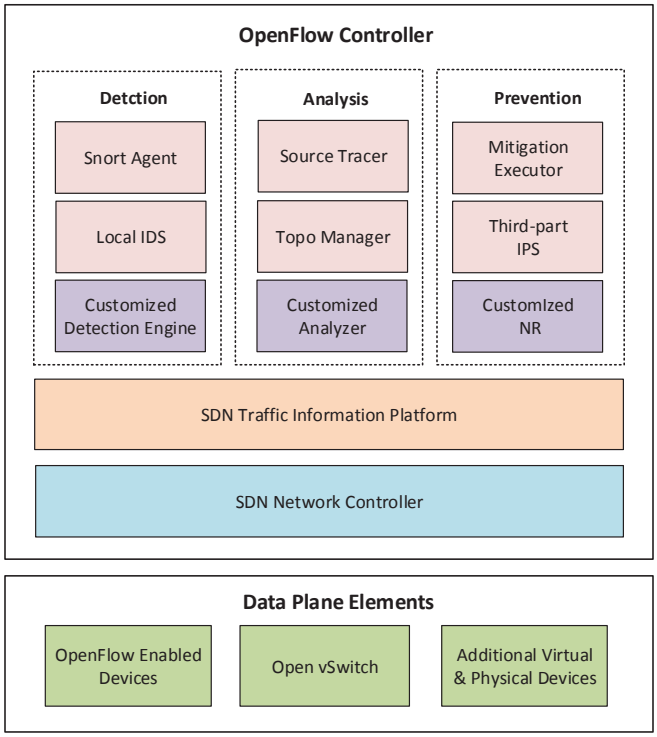


Figure 3.2: System Overall Architecture.

Above the TIP, there are three highly integrated security application modules: Detection, Analysis and Prevention Module. Each module has default pre-developed security applications and extensible interfaces for customized security appliances. A Snort Agent is plugged in Detection Engine by default while local IDS service is en-

abled as the alternative solution for Detection Module. Developers also could design their own detection engine by using proposed Python-APIs and the statistical traffic information from SDN TIP. Source Tracer and Topo Manager are two default security analysis tools provided by system. Users can easily apply their own analysis algorithms to their IPS because TIP is the cornerstone of upper layer security applications and all the network information can be easily fetched from it. Finally, the prevention modules come with Mitigation Executor and Third-part IPS component as the default prevention solutions. More flexible and comprehensive countermeasures could be performed when users design them through proposed API.

Chapter 4

TRAFFIC INFORMATION PLATFORM

When we talk about the network security, traffic information is really important to any kinds of security system. From the perspective of defenders, we need to monitor the traffic patterns to realize the existing of networking attacks. Anomaly behaviors always come with uncommon traffic flows. We need collect and analyze these information to decide further countermeasures. On the other hand, from the perspective of attackers, they always try to bypass the networking monitor, perform spoofing and try to avoid any kinds of tracking. In OpenFlow network, most traffic information is stored in flowtables. We can find basic flow information such as destination IP, source IP, destination port, source port ,e.g, there. With the help of counter, we can know the volume of a flow and the transited packets. However, flowtables are always dynamically updated by openflow and its original statistics information is very simple. Thus, we propose a Traffic Information Platform in our development framework to support constructing security appliances easily and efficiently.

4.1 Architecture of TIP

TIP is designed to collect detail traffic information and provide flexible data access as well as operational functions. There are three goals of TIP design: (1) Retrieving traffic information fast and easily. (2) Providing detail statistics information for upper layer application inquiry. (3) Accessing traffic and packet flexibly.

The architecture of TIP is shown if Fig. 4.1. Four main components are planted in the platform, which are *OpenFlow Monitor*, *Traffic and System Information DS/DB*, *Traffic Preprocessor* and *Traffic Packet Acquisition*. The flowtable monitor watches

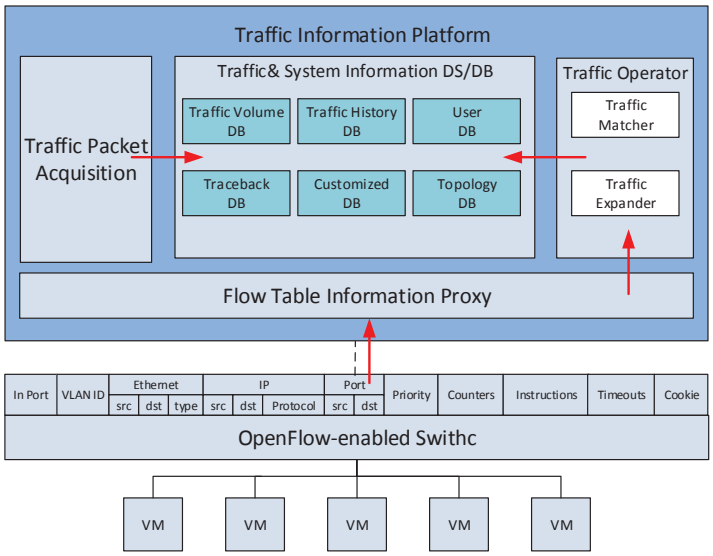


Figure 4.1: Traffic Information Platform

the traffic flows go through the OpenFlow-enabled switches. It also retrieves flowtable information periodically. The Traffic Preprocessor includes two modules, which are traffic matcher and traffic trimmer.

4.2 OpenFlow Monitor

OpenFlow monitor maintains the connections to all OpenFlow devices. All the active openflow connections are monitored periodically. Every interval time, monitor will send a request to all switches to collect useful flowtable information.

The main function of openflow monitor is to establish a connection from monitor controller to openflow switches and warn the anomaly traffic behavior earlier.

4.2.1 OpenFlow Flowtable Request and Connection

OpenFlow messages are how OpenFlow switches communicate with controllers and they are defined in openflow specification. The newest version of openflow spec-

Request Message Type	Openflow Request Type
Description Statistics	opf_desc_stats
Individual Flow Statistics	ofp_flow_stats_request
Aggregate Flow Statistics	ofp_aggregate_stats_request
Table Statistics	ofp_table_stats
Port Statistcs	ofp_port_stats_request
Queue Statistics	ofp_queue_stats_request
Vendor Statistics	OFPST_VENDOR

Table 4.1: Openflow Statistics Information Request Type

ification is 1.4 version and could be found in [5]. In this system, we are using POX controller and it supports version 1.0 currently. The procedure of how to fetch openflow statistical information is shown in Fig. 4.2. To communicate with openflow switch for statistical information, firstly, openflow controller needs to check how many connections from controller to switches are active. For the example in the figure, two connections are active, so controller will send two opf_stats_request to each switch. There are several request types are provided for controllers to request. More request information types could be found in table 4.1. In our monitor implementation, we send individual flow type to switches and handle openflow statistics reply event.

4.2.2 Lightweight Anomaly Traffic Detection Engine

We planted a lightweight anomaly detection engine in our openflow monitor to report the suspect of tenant in the cloud system is under anomaly overwhelming traffic attack in time. The goal of deploying this detection engine in the monitor is to reduce the traffic pressure of detection module in the upper layer when over-

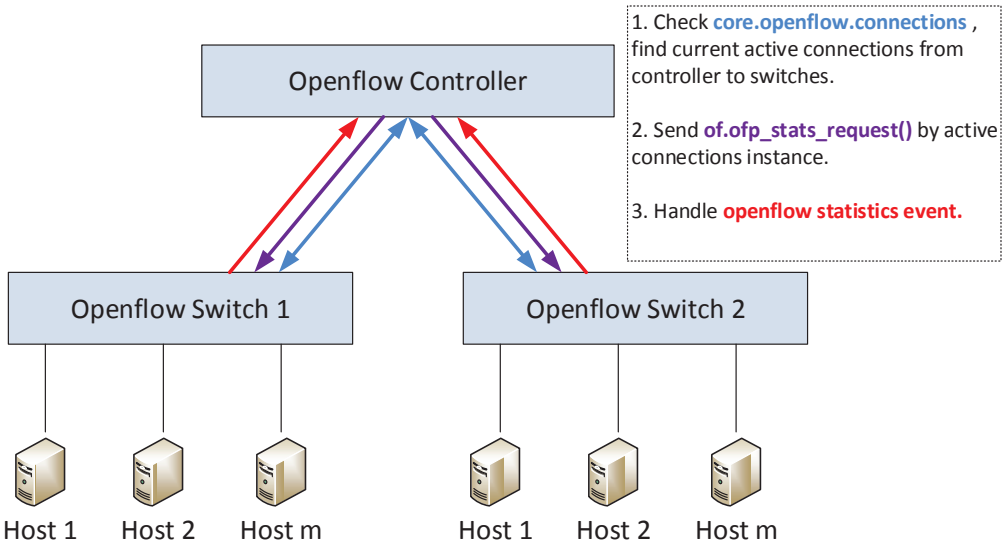


Figure 4.2: Openflow Statistical Request Procedure

whelming attacking traffic is performed. The detection accuracy of Snort and other detection applications will downgrade when detection engined is flooded. So we build this anomaly traffic detection engine in the monitor to tackle flooding traffic. We propose a lightweight collaborative detection engine to detect if cloud system is under overwhelming traffic attacks, such as DDoS attack. All openflow switches are connected and controlled by a centralized detection engine over POX-based openflow controller. The statistics module natively collects all the counter information from each connected OVS, which means that statistics information of every matched flow is collected. After the raw statistics data is collected from all connected switches or OVS, the detection engine aggregates those flow entry statistics information into a centralized statistics data set for our proposed detection algorithm to read data. This is a lightweight detection process, which means we can realize the anomaly network traffic in time with least overhead. It is common to see that traditional network monitor mechanisms collect all the traffic features at the first stage and inspect their

headers and content to alert the anomaly network behaviors as well as classifying legitimate traffic and malicious traffic by using some data mining algorithms which is heavily overheaded. We decouple the whole complicate monitoring phases to two separate stage: (1) Build a lightweight flooding detection engine in the openflow monitor. (2) Implement the complete detection components in detection modules.

The two main benefits of this design are:

- A simplified detection engine can realized traffic anomaly in time.
- This fast detection mechanism can enhance the performance of security modules, improve the detection accuracy and reduce the traffic pressure.

4.2.3 CUSUM Algorithm for Flooding Detection

We propose a lightweight flooding detection engine in the openflow monitor to report the anomaly overwhelming traffic in time and trigger the corresponding mitigation to tackle the flooding attacks. This engine applies a change point detection algorithm to report the traffic spikes. As one of the major sequential change point detection algorithms, non-parametric Cumulative Sum (CUSUM) [35] algorithm is selected to detect the occurrence of abrupt change of the system overall traffic volume. CUSUM algorithm could minimize the detection delay with a fixed false alarm rate [36] and efficiently report the anomaly change. The idea behind this approach is that if a change occurs, the probability distribution of the random sequence will also be changed[37]. Our following proposed method is inspired by previous work [38, 39]. In [38], authors apply a CUSUM algorithm to detect the occurrence of large difference between the number of SYNs and FINs while authors in [39] detects the anomaly increase of new source IP ratio by the CUSUM algorithm, which are different from our work.

To simplify the engine and reduce the overhead, we only focuses on the system overall traffic volume. Different from other detection engine which based on machine learning and data mining methodologies, less features are considered and involved in the computation will reduce the overhead and enhance the detection delay. The system overall traffic volume information is captured by openflow monitor. For different detection requirement, the engine could modify the record resolution of tip module to acquire different date sequence with selected time interval ΔT

In this stage, our goal is to implement a quick response alert engine which detects the abrupt traffic volume change in the cloud and figures out the potential victim in the cloud system. This detection engine answer one question: Is there any VM in the system which its ingress traffic increase abruptly? To answer this question, we formulate a problem :

Given a sequence of observed random variables $\{X_{i,n}\}$ for each VM i, report if there exists a VM k which the statement is changed abruptly from statement θ_0 to θ_1.

To keep the computation and storage cost low, we only focus on the ingress traffic volume to each VM because this type attacks are designed to delivery overwhelming traffic to exhaust the network resources of target victim. So from the perspective of victim, the continuous and abrupt increase of ingress traffic to one VM is reasonable to be considered as an anomaly behaviors and is needed for further investigation. So we create a profile-based monitoring table in the centralized controller with simple data structure to watch the ingress traffic of each VM. The data structure of the monitor table in the controller is : {*Destination IP, Source VM IP sets, Traffic Volume, Time Stamp*}. The controller fetches the traffic volume statistics information from data plane of each OVS every Δt period and accumulate all the traffic volume reports from each OVS based on the same destination IP Address. Let $R_i(t)$ be

the traffic volume record for VM i at monitor table at time stamp t, where t is the discrete time index. For each sample data sequence $\{S_i(t), t = 0, 1, ..., n\}$, we have $S_i(t) = R_i(t + \Delta t) - R_i(t)$ for each VM i. To make our approach more general, we normalize the sample data sequence by following equation:

$$X_i(t) = \frac{S_i(t)}{\hat{S}_i(t)} \tag{4.1}$$

where $\bar{S}_i(t)$ is the exponential moving average (EMA) of data sequence $S_i(t)$ and it could be calculated by :

$$\bar{S}_i(t) = \begin{cases} S_i(t) & \text{if } t = 1, \\ \alpha S_i(t) + (1 - \alpha)\bar{S}_i(t - 1) & \text{if } t > 1. \end{cases} \tag{4.2}$$

where α is a constant smoothing factor between 0 and 1, which represents the degree of weighting decrease.

We apply the non-parametric Cumulative Sum (CUSUM) algorithm [35] to detect the occurrence of abrupt change of the ingress traffic volume sequence of each VM. CUSUM is an efficient algorithm for change point detection, the idea behind this approach is that if a change occurs, the probability distribution of the random sequence will also be changed[37].

Given a data sequence $X_i(t) = \{x_{i,1}, x_{i,2}, .., x_{i,n}\}$, $x_{i,n}$ is the traffic volume in the nth time interval for VM i. The non-parametric CUSUM algorithm assumes that the mean value of sequence is negative during the normal conditions and it becomes to positive when a change occurs. So we needs to pull down the sequence $X_i(t)$ to a new sequence $Z_i(t)$, which $Z_i(t) = X_i(t) - \beta$ without compromising the statistical feature. β is a constant to help us transform the random variate sequence $X_i(t)$ to suitable sequence $Z_i(t)$ where both majority data value and mean of normal data are negative.

The non-parametric CUSUM could be presented by following:

$$y_i(t) = S_i(t) - \min_{1 \le k \le t} S_i(k) \tag{4.3}$$

where $S_i(k) = \sum_{n=1}^{k} Z_i(n)$,with $S_i(0) = 0$.

To make computation easier, this algorithm can be simplified as follow based on[35]:

$$y_{i,t} = (y_{i,t-1} + Z_i(t))^+, y_{i,0} = 0, \tag{4.4}$$

To determine a potential flooded attack to VM i, the decision function can be presented as follows:

$$d_N(y_{i,t}) = \begin{cases} 0 & \text{if } y_{i,t} \le N, \\ 1 & \text{if } y_{i,t} > N. \end{cases} \tag{4.5}$$

N is the flood threshold and $y_{i,t} > N$ indicates the occurrence of flood traffic to specific VM i at time stamp t.

4.3 Traffic Preprocessor

The returned flowtable information from OpenFlow switches is raw data and further processing is needed. Traffic Preprocessor is designed to trim, organize, compute and classify raw statistics traffic data read from flowtable monitor.

Traffic Matcher is an openflow object which can match a specific traffic flow. This matching fucntion is the native feature of openflow protocol and packets or flow could be matched by a single field of a set of combinations.

Traffic Trimmer can be used to trim raw statistics data into a desired or customized format. This module works with database interfaces to generate the following statistics DB/DS, i.e., *Traffic Volume DS*, *Traceback DB*, *History Traffic DB*. Users can also use these modules to build their own customized data set, i.e., *Customized DS*, to fulfill their own design purposes.

4.4 Traffic and System Information DS/DB

In this section, we explicitly discuss the design of database and data sets in traffic information platform. By default, all these data information is stored in the MySQL [40] database. For some security applications which need high speed read and write, data set is designed for this situation. We also provide programmable data interfaces and database APIs for users to design and create their own customized data structures. This compatible extension facilitate upper layer security applications to easy read and store data information. Currently, we introduce six default databases storing corresponding real time traffic information and processed data.

Traffic Volume DB is designed to help the lightweight anomaly detection engine in openflow monitor to identify anomaly behaviors in the system. It also can be used for users to develop their own volume-based upper layer detection engines. Most this kind engine needs the traffic volume sequences to watch system status and detect the anomaly traffic behaviors. TIP will request each openflow switch to return its current ingress traffic volume of each port attached by an active VM every ΔT time interval. Then the Traffic Trimmer will trim the raw data as follows formation: {*Destination IP, Source VM IP sets, Traffic Volume, Time Stamp*}.

For each VM i in the system, let $R_i(t)$ be the summation of traffic volume record from all the source IP addresses to the destination VM i at time stamp t, where t is the discrete time index. So at time stamp t, the system overall traffic volume record $V(t)$ could be calculated by $V(t) = \sum_{i=1}^{k} R_i(t)$, k it the total number of current active VMs in the system. After we have the system traffic volume record at each time stamp t, to generate the system traffic volume data sequence $\{S_t, t = 0, 1, ..., n\}$, we have $S(t) = V(t + \Delta t) - V(t)$. The request time period Δt is selected by corresponding Detection Engine and detection resolution.

User DB is designed to support user-based security applications. This database can be extended for user authentication, group-based authentication and management system.

Traceback DB is designed to support upper layer application Source Tracer to track back the attacking sources in cloud system. In traditional cloud environment, when attack is performed with spoofing technique, it is truly difficult to trace them back. However, when we apply openflow in the cloud system, the decoupled data and control plane enable us to track the attacking sources by looking up the traffic ingress and egress ports of flow table, which can not be forged. Thus, as the database behind source tracer, traceback database provides the real *Source VM* and *Destination VM* pairs, which means even the source IP address of traffic packet is spoofed, we still can traceback the source virtual machine by looking up this database. To generate the real source and destination pairs, TIP obtains the raw statistics data from flow table of each openflow switch, the traffic trimmer will trim the data into following structure: {*DPID, Ingress Port, Interface type, Destination IP, Egress port,*}. This table information is stored in each openflow switch and its flow table. *DPID* indicates which switch this flow belongs to. *Ingress Port* and *Egress Port* represent which port a flow comes through and which port this flow is delivered to. *Interface Type* shows the type of the ingress port of this flow, which could be Controller, switch or Virtual Machine. We could derive the interface information by *ovs-ofctl show bridge* command and request detail flow table information by openflow protocol. This table tells us the abstract topology of current network connection and the true location of each switch with its attached active VMs. It also shows that where each traffic flow is exactly delivered to. We can have an accurate picture of real source & destination pairs from this database. The Source Tracer will traceback the attacking sources based on this database.

Customized DB is an empty dataset for users to develop their own real time dataset based on their requirements. *Traffic Trimmer* provides several basic methods for developers to trim the raw statistics information to their own data structure and sequence. Those basic methods could be found in Table. 5.2. This dataset provides a flexible interface for upper layer applications to acquire customized real time data set.

For example, authors in [39] present a method to proactively detect the DDoS attack by using source IP address monitoring. To plug-in this detection method, a IP address monitoring data set is required. Users could use get_active_switch function to find all the connected OVS and use get_raw_data (dpid, start_time_stamp, end_time_stamp) to query the history traffic information. We can create the IP Address Database from the history traffic database. Then based on the detection method mentioned in [39], we apply the customized_monitor (match, start_time_stamp, time_period, recurring = true) function to watch all the real time traffic information. We set the $\Delta_n (n = 1, 2, 3, ...)$ as the detection resolution and the parameter in the function, then collect IP address information during each time slot Δ_n and format a real time system IP address data set. This data set could support upper layer detection engine and algorithm to analyze the system status and report the potential DDoS attack.

Topology DB works closely with Topology Manager in analysis module. Topology manager discoveries network topology in an openflow network and store the topology information in the database. Topology DB includes three main tables which are openflow devices table, network nodes table and connections. Openflow switches information is stored in openflow devices table and unique openflow dpid serves as primary key in the table. Network nodes table stores the node information in the cloud system. It includes IP address, MAC address, node type and other related information. MAC address is the primary key because it is the unique identity for

network nodes. Connections table stores all the discovered connections in the network. We also could save all these information to a wrapped JSON file when we want to handle web-based data exchange.

4.5 Traffic Packet Acquisition

Some detection engines or DPI agents might need to investigate a packet content for some deep detection purpose. Traffic packet acquisition is designed to fulfill this requirement. TIP not only can obtain traffic statistics information, but also fetch a complete traffic packet and redirect it to security agent. This modular will poll the real packets instead of traffic statistics information to the controller that further forwards to the security applications. Users are able to define what traffic packet or a traffic flow is needed to be polled to the controller or other agents to be inspected. The traffic matcher in the preprocessor module, enables developers to specify what packets they are looking for. After users determines the packet they are interested in, new flow table rule will be injected into all connected OpenFlow devices so that forwarding plane in openflow will redirect all the following matching packets to desired security agents or controllers.

Chapter 5

SDN-BASED IPS SECURITY MODULES

5.1 Detection Module

5.1.1 Snort Agent

Snort is a multi-mode packet analysis tool dominating the IDS/IPS market and has overall performance strength over other products [41]. Sniffer, packet logger, data acquisition tools are main components of Snort. In its detection engine, rules form signatures to judge if the detected behavior is a malicious activity or not. It has both host and network-based detection engines and it has a wide range of detection capabilities including stealth scans, OS fingerprinting, buffer overflows, back doors, and so on.

5.1.2 Flow Expander

When we develop SDN-based detection applications, we need to deep investigate traffic information and packet contents. Traffic packet acquisition can be used to fetch specific traffic packets to the DPI agent. However, due to the different flow rules configuration in different openflow switches, sometimes there is not enough detail information for each flow in the flowtable. At this point, we would like to expand raw traffic flow rules and informations in the flowtable to further analyze the traffic patterns. So we plant a flow expander in the system to expand raw flows to several more detailed flows in the system.

The core of flow expander is a Rule Expander Algorithm. It can expand raw traffic rules into detail one for further analysis. By default, the openflow switch only

maintains the IP or lower layer flow table entries if it is configured as a layer2/3 switch. Once the Flow Expander is triggered, all openflow switch will capture all the traffic going to the network nodes, it will look into the transport protocol and corresponding port information. If the flowtable entry exists, the counter in openflow switch will be incremented; if the flowtable entry does not include the transport layer information, then the switch will create a set of flow table entries including the transport layer information, i.e, TCP port number. Once the detailed flow entry is injected into the flowtable, the network monitor and customized SDN-based detection application is able to collect more detailed statistics information of the targeted traffic and traffic trimmer is able to acquire them to formulate the customized data sets.

Another example is shown in the Fig. 5.1. The packets come from node A is forwarded to node B by the rule with only source IP, destination IP and action information which is {A-B, forward}. However, this information is insufficient for a detection engine to differentiate the malicious traffic type from healthy traffic types. Thus, the traffic expander will inject more detailed rules based on a spanning model. In this example, two medium priority rule {A-B, TCP, forward} and {A-B, UDP} are injected. If detection engine would like to know more detail information, four highest priority rules {A-B, UDP, 53, forward}, {A-B, UDP, 2049, forward},{A-B, TCP, 22, forward} and {A-B, TCP, 80, forward} can be injected to the flowtable. Due to the different priority, the TCP traffic with port 80 from A to B only matches the highest priority rule, which is {A-B, TCP, 80, forward} and it will not create duplicate counter information. In summary, we utilize the priority feature of the flow table entry and we always want to match leaf node rule to get the most detailed rule statistics information. After collecting sufficient statistics information, SDN-based customized detection application can identify potential malicious traffic more confidently.

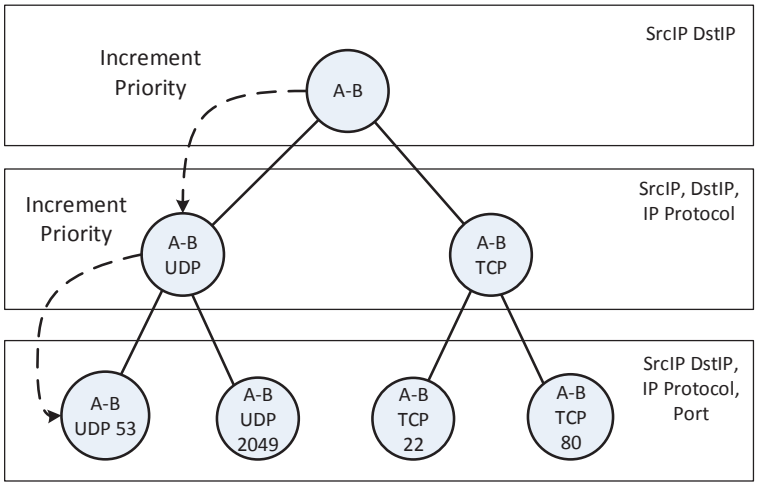
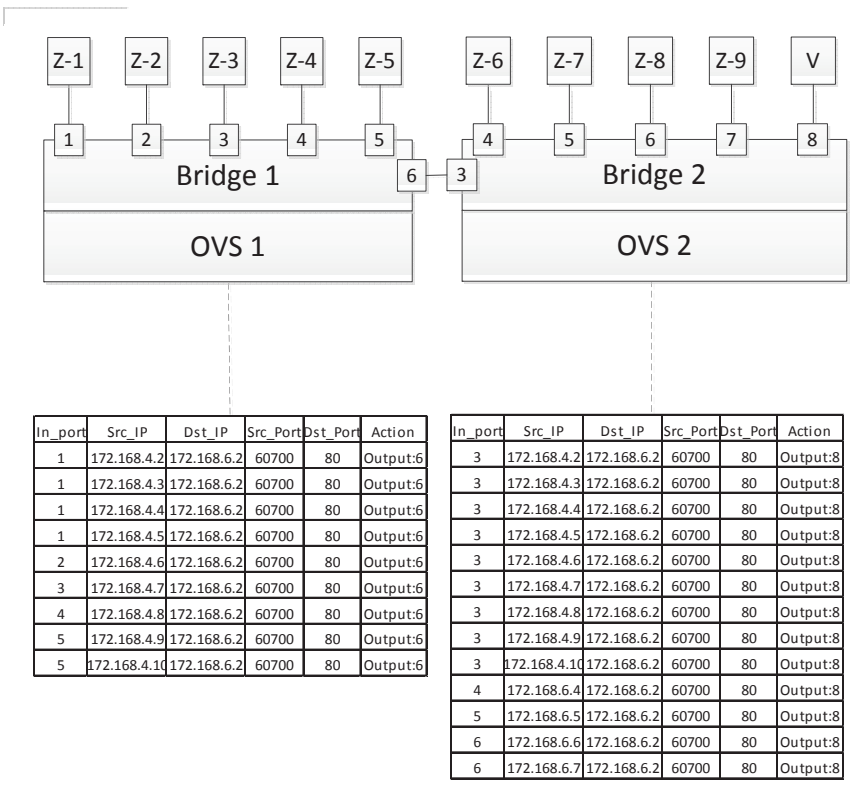


Figure 5.1: Tree-based Rule Auto Spanning Model

5.2 Analysis Module

5.2.1 Source Tracer

To support the analysis module, we plant source tracer in our system. As we see, IP spoofing is widely used to forge the source of attackers and bypass the forensic and prevention system. So when this technique is performed, we can not simply trace back to locate all attackers by using the source IP in the flowtable record because the source IP addresses are spoofed. Traditional solution such as packet marking technique[42, 43] to perform IP traceback always needs to modify the packets and leads overhead which does not handle IP spoofing very well. However, in SDN environment, the openness of data plane and control plane gives us an opportunity to look at the detail of data exchange inside the switch, which benefits the traceback of attacking sources. When a network node is performing malicious attack with IP Spoofing, we can find several fake IP addresses in the flow table pointed to the victim IP address come from the same ingress port, which means as long as the traffic is



In_port	Src_IP	Dst_IP	Src_Port	Dst_Port	Action
1	172.168.4.2	172.168.6.2	60700	80	Output:6
1	172.168.4.3	172.168.6.2	60700	80	Output:6
1	172.168.4.4	172.168.6.2	60700	80	Output:6
1	172.168.4.5	172.168.6.2	60700	80	Output:6
2	172.168.4.6	172.168.6.2	60700	80	Output:6
3	172.168.4.7	172.168.6.2	60700	80	Output:6
4	172.168.4.8	172.168.6.2	60700	80	Output:6
5	172.168.4.9	172.168.6.2	60700	80	Output:6
5	172.168.4.10	172.168.6.2	60700	80	Output:6

In_port	Src_IP	Dst_IP	Src_Port	Dst_Port	Action
3	172.168.4.2	172.168.6.2	60700	80	Output:8
3	172.168.4.3	172.168.6.2	60700	80	Output:8
3	172.168.4.4	172.168.6.2	60700	80	Output:8
3	172.168.4.5	172.168.6.2	60700	80	Output:8
3	172.168.4.6	172.168.6.2	60700	80	Output:8
3	172.168.4.7	172.168.6.2	60700	80	Output:8
3	172.168.4.8	172.168.6.2	60700	80	Output:8
3	172.168.4.9	172.168.6.2	60700	80	Output:8
3	172.168.4.10	172.168.6.2	60700	80	Output:8
4	172.168.6.4	172.168.6.2	60700	80	Output:8
5	172.168.6.5	172.168.6.2	60700	80	Output:8
6	172.168.6.6	172.168.6.2	60700	80	Output:8
6	172.168.6.7	172.168.6.2	60700	80	Output:8

Figure 5.2: Source Tracer for Spoofing IP

delivered through the switch, the ingress port of OVS can not be forged and it is an essential clue to traceback all the spoofed attacking sources.

Fig. 5.2 gives an example of IP Spoofing scenario. We can see all the VMs in the system are connected to the OVS bridge by virtual interface (VIF) and each network interface has attached to one switch port. IP Spoofing is performed in this situation, so we can find more source IP addresses than current active VMs in the system. Zombie machine 1 inserts 4 flow entries in the flow table and it is sending 4 kinds of traffic to victim VM with forged source IP addresses. Zombie machine 5 and 9 also apply IP Spoofing technique to challenge the traceback of zombie machines. In this situation, we can not track the zombie machines by IP address because they are spoofed and changed intentionally. However, the good news here is the ingress port

of OVS can not be changed. From the flow table of OVS 1, we can see the source IP address {172.168.4.2, 172.168.4.3, 172.168.4.4, 172.168.4.5} come from the same ingress port and this ingress port is connected to only one active VM. So when we traceback the zombie machines, we only focus on the ingress port, no matter how IP address is spoofed or not, the goal here is to find out the specific "physical" zombie machines who are sending flooding traffic.

Thus, we propose an IP spoofing prevented trace back mechanism based on the OVS virtual port tracing. The benefit of considering the OVS port over the traditional network switch port is that it is possible for us to know if dedicated OVS port is connecting a VM directly or another OVS's tunnel port. As shown in Fig. 1.1, each flow entry has a field called in_port. Through this way, we are able to trace back to the zombie VM from the OVS that victim directly connects. Since the connectivity of all OVS can be modeled as a graph. We first need to define a function called traceback as below:

$$v.traceback(dst\_IP, transport\_type) = \{OVS/VM\}, \tag{5.1}$$

for all OVS/VM sending traffic with destination IP = dst_IP, transport type = transport_type directly to OVS v. Thus we propose a trace back algorithm based on depth-first search, which is described in Algorithm 1.

5.2.2 Topology Manager

Topology Manager is planted in the analysis module to help security application to analyze attacking in the network and evaluate the corresponding countermeasures. Topology Manager is composed with two main components: Topology Finder and Mininet Generator. Topology finder can discovery all the network devices and nodes in current cloud system. It also can depict the connections between network nodes

Algorithm 1 Source Trace Back Algorithm

Input: $TrafficType_{DDoS}$, VM_{victim}, OVS_{victim}

Output: {Set of Zombie VMs}

1: $v \leftarrow OVS_{victim}$

TraceRecursion (OVS v, Set ZVM)

2: **if** v = VM **then**

3: ZVM.add(v)

return

4: **end if**

5: **for** each $x \in v.traceback(dst\_IP, TrafficType_{DDoS})$ **do**

6: **if** $x.visited = FALSE$ **then**

7: **TraceRecursion**$(x, list)$

8: **end if**

9: **end for**

10: **RETURN** *list*

and devices. This component helps attack graph applications development and provide overview architecture for developers to deal with malicious activities. Mininet Generator can depict the network architecture as well, but it also could create a virtual networking environment through Mininet [44]. Mininet contains a real network switch kernel and application. It runs on a virtual machine or a single PC and creates a real virtual network. It is a great experiment environment for Openflow and software-defined networking systems.

Topology Finder could be used to depict the network architecture in cloud computing system. There are three main components in the topology finer: topology discovery, host tracker and data manager. In the topology discovery, controller will send link layer data packets (LLDP) every certain time. This module can discover

the connectivity between all active openflow switches. Each openflow switch listens to a LLDP event and parse this LLDP packet to instance handler. Each LLDP comes with the dpid of original switch, so we maintain a table with original dpid and confirm the connection when original dpid is received and this dpid is in current active openflow core connection lists. All the topology information including openflow switch, network host, links between host and switch is stored in the topology database in traffic information platform. If users would like to display the topology in the web, a JSON manager will translate the topology information to a JSON file and send it to the web service by HTTP request or other type of web services.

Experiment Mininet Generator could be used to generate a virtual networking environment in a single host or on a virtual machine for developers to run security related experiments. After the network topology is discovered by topology finder, a embedded translator in the mininet generator will create a python-based mininet script based on the input topology. This translator can translate JSON format information of switch dpid, host mac address and ip address, connection between network devices into python language. Then we can simulate the this real environment based on the mininet application as a custom topology.

5.3 Prevention Module

5.3.1 Network Reconfiguration

Network reconfiguration is a flexible approach to manipulate the network characteristics. It can change network topology, traffic packet header, connection speed and network parameters and so on. When we enable SDN in the cloud system, network reconfiguration is used to counter malicious activities in a IPS system. Major network reconfiguration can be found in Table 5.1 : 1) Traffic Redirection (TR) is an action

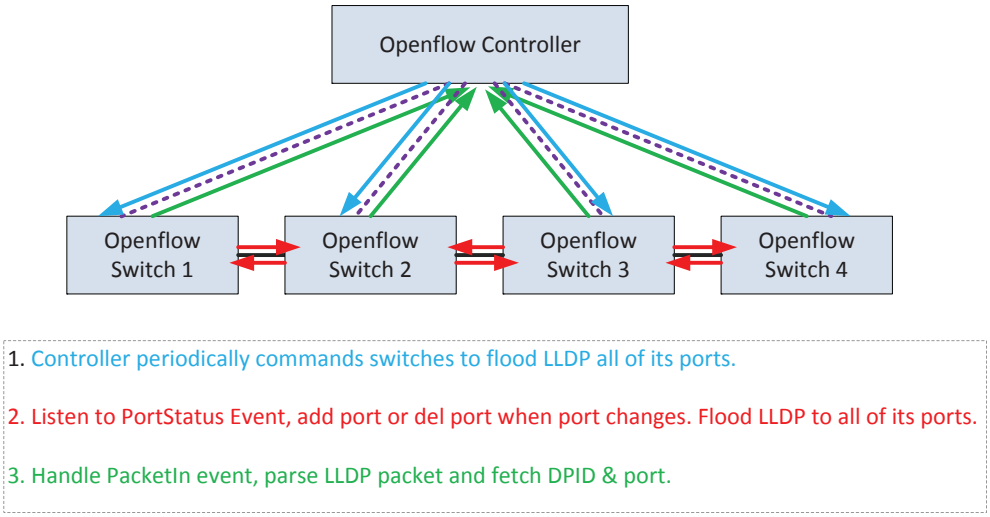


Figure 5.3: Topology Finder Procedures

which redirect a traffic flow to a secure agent (e.g. DPI unit, Honeypot) by modifying the packet header. It is usually implemented by rewriting MAC/IP address in header. Controller also can push entry to flowtable which could rewrite packet header on matching packets instead of sending the packet to remote controller. 2) QoS Adjustment (QA) is an efficient reconfiguration to handle overwhelming traffic in some specific scenarios. Openflow switch can adjust the QoS parameters of any attached virtual ports. We can make sure suspect attacking traffic only generate limited impact on the network and hosts by decreasing the QoS rate in the openflow switch. QA also can be applied to work with other NRs such traffic isolation or redirection. 3) Traffic Isolation (TI) is a high level implementation of traffic redirection. It provides an isolated virtual networking channel separated, e.g., separated virtual bridges, isolated ports or GRE tunnel. Malicious traffic will be redirected into those virtual bridges and only hosts on isolated channel can be impacted. 4) Filtering is similar with the filter action in Iptables, but they are different in that filtering in NR will handle packets at openflow switch kernel space and will not be forwarded to a

remote controller or snort agent. MAC/IP address change is a very straightforward way to prevent the victim from being attacked by the malicious traffic. The default IPS action, i.e., drop, can be also regarded as a filtering rule that drop the matching packets. 5) Blocking is a common action both in traditional IPS or openflow network reconfiguration. Some attacks are performed by exhausting network resource of a public service port. By blocking ports, the attack can be prevented as the attacking path is disconnected. 6) Quarantine is a comprehensive approach to isolate traffic in cloud virtual networking environment. It works similarly with TI but it isolates the suspect network resources (not just the suspect traffic) such as virtual machines or servers. Quarantine mode can apply more flexible and self-defined rules to reconfigure a network envrionment. For example, you can quarantine suspect VMs with only ingress permission and decline all egress requests. Thus, such VMs can only receive traffic but can not send traffic to network.

No.	Countermeasure
1	Traffic Redirection
2	QoS Adjustment
3	Traffic Isolation
4	Deep Packet Inspection
5	Filtering
6	MAC address change
7	Packet Header (MAC/IP Address) Change
8	Block Port
9	Quarantine

Table 5.1: Network Reconfiguration Actions

5.3.2 Representative NR Actions in Prevention Module

In this section, we would like to discuss two main network reconfiguration approaches which are traffic redirection and QoS adjustment.

Traffic Redirection

Traffic redirection can be implemented by three methods: MAC Address Rewriting, IP Address Rewriting, and Port Rewriting. When detection engine detects suspect packets, the controller will firstly inject corresponding openflow entries (i.e., matching packet header fields and corresponding actions) to openflow switch to update the flowtable. Header changing is done by flowtable when certain packets matches certain entries. Then corresponding redirection will be performed for matched traffic flows. Actions are defined in the traffic rules. When the header fields of flowtable are changed, e.g., source IP, destination IP, source MAC, destination MAC, the original traffic flows will be handled based on the new flowtable rules and they will be delivered to new destination. When these information is changed, the OVS will naturally forward packets to the changed destination address. It is especially efficient when dealing with the suspect packets which can not be confidently confirmed as attack and are expected to redirect to a detection agent for further checking, e.g., Deep Packet Inspection (DPI) or honeypot.

Port rewriting can be used to implement TR as well. It also comes from the natively feature of openflow. Each virtual bridge or physical switch in openflow environment can be considered to provide virtual ports or physical ports. All virtual machines are connected these ports via their network interfaces. Thus, openflow switch can simply forward a traffic packet by outputting it to a specific port or flooding. Then the VM connected port can receive this packet. Thus, SDN-based network

reconfiguration can set any virtual or physical port as output to implement the traffic redirection function without changing the packet header. One of the obvious benefits of this implementation is that any packet header will not be changed while traffic is being redirected, which is efficient and effective to some components when collecting original network packets for further study.

QoS Adjustment

QoS Adjustment (QA) is a desired countermeasure to defend cloud networking from flooding attacks such as DoS and DDoS. It is also suitable for delay suspect traffic. For example, the network connections of suspect victims are under high workload, but detection engines can not confirm the flooding attacks exist, it is not supposed to shutdown all the connections to victims, so QoS adjust is introduced to handle this kind of situation.

Based on the configuration of Open vSwitch, reseting QoS parameters on virtual interface or port are two ways to implement QA. These two implementations are used in different scenarios. When setting the QoS limitation on virtual interface, packet source, i.e., attacker, is needed to be located. Thus, the number of suspect attackers can not be too large. On the other hand, if the attacking sources is a huge set, such as the zombie machines in a DDoS attack, it is infeasible and impractical to locate all attackers and limit rate by through their virtual interfaces. So another implementation of QA, by limiting the incoming port on OVS, is introduced to solve this issues. When packets arrive at OVS, there is a packet_in event port, which is also called in port, as shown on the flowtable in Fig. 1.1. We can adjust the QoS of this incoming port, then all arriving packet which are exceeding the limit will be dropped without further process. Also, when IP spoofing is used by attackers , QA is also efficient in this situation.

5.4 Python-based Security Application Development API

Table. 5.2 illustrates all Python-based APIs for TPI modules. Table. 5.3 illustrates all Python-based APIs for service modules including detection module, analysis module and prevention module.

Table 5.2: TIP API Summarization

Function	Parameters	Explanation
get_active_switch		return all active connections
get_active_port	dpid	return all active ports of specified OpenFlow switch
get_port_volume	dpid, port, start_time, end_time, ingress	get history port traffic volume from start_time to end_time, time format shoud be python standard time format, ingress value is true by default and set it to false to watch the egress traffic volume of the port
monitor_port_volume	dpid, port, start_time, time_period, recurring, ingress	monitor port ingress or egress traffic volume from start_time every time_period seconds the method monitor ingress traffic by default, when ingress=false, it will monitor the egress traffic volume
monitor_volume	match, start_time, time_period, recurring	monitor traffic volume from start_time every time_period time, user needs to set the attribute of match to define the flow to be monitored and the function will return real time data set
get_volume	match, start_time, end_time, time_period, ingress	return the history traffic volume information for the matched flow from start_time to end_time, by default ingress value is true and set it to false to monitor the egress traffic volume
flow_table_expander	in_port, out_port, ethernet_layer, ip_layer, transport_layer	expand the raw flow table rules to detail flow rules automatically, set the ethernet_layer, ip_layer, transport_layer parameter to false to disable the expand for corresponding layer
get_packet	match,controller_port	send all matched packets to controller
traceback_IP	destination_IP, traffic_type	this function return all the source ip_addresses who are sending specific traffic type to destination_ip
customized_monitor	match, start_time, time_period, recurring	set up a monitor, match flow and start_time, return the match flows information every time_period
check_customized_counter	match, start_time, counter	check if the number of matched packets exceed the counter value
get_raw_data	dpid, start_time, end_tim	get raw traffic statistics information of selected switch from start_time to end_time. It returns a python dictionary with dpid and record time as the key

Table 5.3: SDN-based Security Application Module API Summarization

Function	Parameters	Explanation
detection_engine_connector	match, detection_engine_ip	get the matched packet from the OVS and forward it to the third party detection engine
traffic_volume_monitor	ovs, port, threshold	set up a monitor to watch the system traffic volume, when the traffic exceed the threshold, return a notification
customized_counter	match, start_time_stamp, counter	set up a counter for matched flow, when the number of matched flows reach the threshold, it will return a notification
attack_source_traceback	victim_ip, victim_ovs, ip_proto, port_number	return the list of port and the attached OVS attack_traffic_type is needed to traceback the source attackers from victim_ovs
traffic_drop	match	drop all the traffic based on the match fields
traffic_redirect	match, destination_ip	redirect all the matched traffic to new destination_ip
traffic_redirect_spoof	match, destination_ip, spoof_ip	redirect all the matched traffic to new destination_ip and change the source ip to spoof_ip
traffic_qos	dpid, port, qos_rate, ingress, egress	set the port of openvswitch ingress and egress traffic rate ingress and egress traffic rate is limited by default, set ingress or egress to false to disable the rate limit

Chapter 6

HOW TO BUILD A DEFENSIVE SYSTEM

In this chapter, I will introduce how to build a DDoS Defensive System by using proposed development framework. We first present the DDoS attack model and analyze the feature of this attack. Then we design a DDoS defensive architecture to address this problem. At last, we build this defensive system based on our development framework and show the processing flow of this implementation.

6.1 Problem Analysis

Distributed Denial-of-Services(DDoS) is an attack which uses many computer hosts to launch a coordinated DoS attack against one or more targets [45]. Attacker is able to multiply the damage of DoS significantly by controlling the resources of multiple innocent computer hosts, which can be considered as the platform of attack. This attack is distinguished from other attacks because the locations of attackers are "distributed" over the network and it aggregates these forces to create lethal traffic. DDoS can be simply divided into two categories, the bandwidth exhausting attack and the resource exhausting attack. The first one is performed to flood the victim's network with overwhelming traffic that prevents legitimate traffic from reaching the victim, while the resource exhausting attack is trying to break the victim's system by either compromising the protocol running on victim's system or crashing the victim's system by sending incorrect data packets.

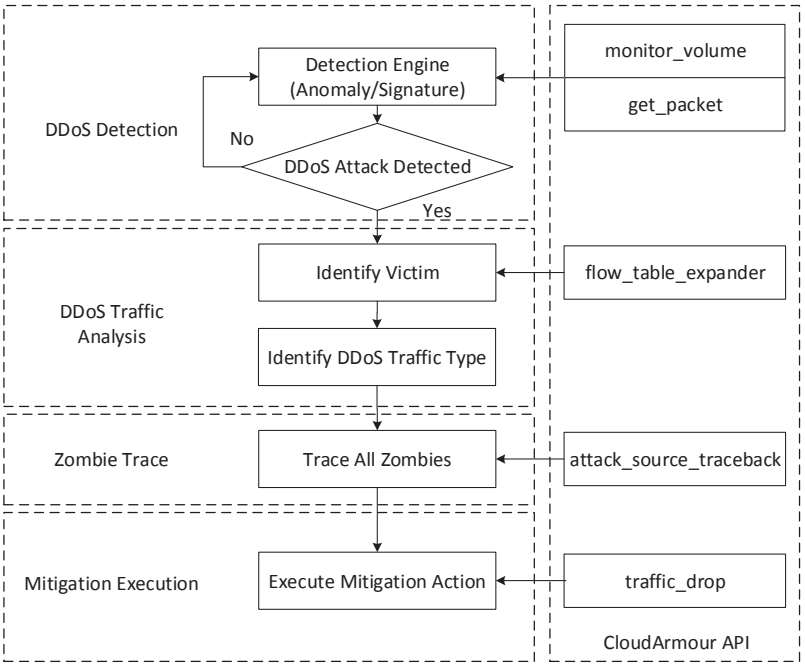


Figure 6.1: DDoS Defensive Solution Construction Flow

6.2 Processing Flow

To construct the DDoS defensive solution based on our framework, we first divide the defensive solution into four phases: DDoS detection, attack analysis, zombie machines traceback and mitigation execution. These four modules are executed in an assembly line fashion. The processing phases of the DDoS defensive solution is illustrated in Fig. 6.1.

The DDoS detection engine is designed to detect the DDoS attack behavior. It can be implemented as either a signature-based approach or an anomaly-based one. The anomaly-based detection engine can be simply build by calling monitor_volume(match,start_time, time_period,recurring) function. We can also plant an existing signature-based detection engine such as Snort and call detection_engine_connector(match,detection_engine_ip) function to forward traffic flows to Snort or other

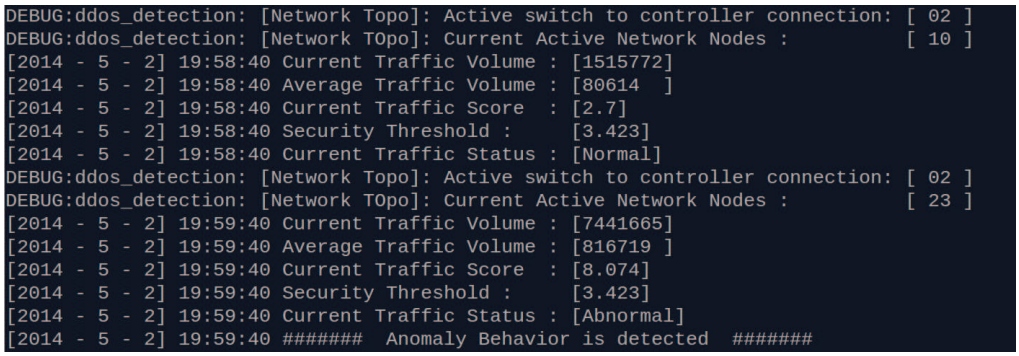


Figure 6.2: DDoS Detection Engine Implementation Screenshot

detection engine. After the attacking behavior is detected, further inspection needs to be done to determine: (1) which host is victim; (2) what is the type of DDoS traffic initiated from zombies. So, we can call Flow_table_expander(in_port, out_port, ethernet_layer, ip_layer, transport_layer) function to generate more elaborated traffic statistics information, which can be further analyzed to answer previous questions. When DDoS behavior, DDoS traffic type, and victim are all confirmed by the previous steps, attack_source_traceback(victim_ip, victim_ovs, attack_traffic_type) function is called to identify all DDoS traffic sources in zombie traceback phase. After zombie machines are all identified by traceback function, mitigation will be performed to stop the DDoS attacking traffic at all port entries of the OpenFlow devices.

6.3 Implementation

We implemented this DDoS Defensive Solution based on our framework. Fig. 6.2 is the snapshot of detection engine. We choose the detection resolution ΔT equals 60 seconds so that the DDoS detection engine queries the traffic volume, average traffic volume, and CUSUM statistics score every 60 seconds. When the CUSUM statistics score exceeds the threshold, Traffic Analyzer and Mitigation Executor will be trigged to protect the system.

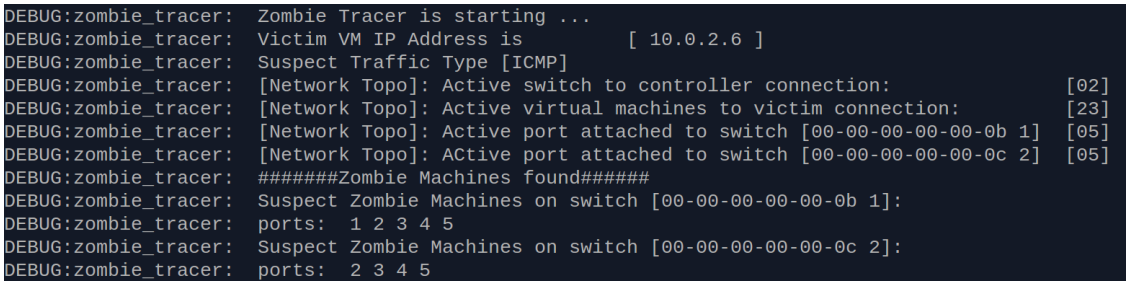


Figure 6.3: Screenshot of Source Tracer in DDoS Attack Scenario

Seagull[46] is used to generate background traffic at a normal speed and Pktgen[47] is used to emulate DDoS traffic due to its ability of generating packets at extremely high speed in the kernel.

Source Tracer module is able to track all the packets even they are intentionally spoofed to bypass security agent. This module can find the attacker who is sending malicious packets with certain pattern or distributed attackers who are sending overwhelming traffic at the same time and forge their sources. In this scenario, we develop a zombie tracer application based on our module and APIs.

Fig. 6.3 is the screenshot of the this zombie traceback application. All the VMs in the system are connected to OVS bridge by virtual interface. Some of VMs are zombie machines who are sending overwhelming traffic with spoofed source IP addresses. This application is able to find the zombie machines in the network and will identify the ports that zombie machines are connected to in the openflow switch. From the screenshow we can see that the victim IP address is 10.0.2.6 and attack traffic type is ICMP. It also reports the number of current active IP addresses, i.e., 23, and the number of active OVS ports, i.e., 5, and then it tracebacks all the ports of suspect zombie machines in each OVS. From the screenshot, ports 1,2,3,4,5 on OVS1 and ports 2,3,4,5 on OVS2 are determined as the ports that the zombie machines are connected.

Chapter 7

EVALUATION

7.1 How SDN-based Countermeasure Improve Network Security

In this section, we are comparing the performance of SDN-based mitigation approaches and traditional IPS mitigation approaches, i.e, Iptables-based IPS. In SDN experiment, we only deploys the default NR action (drop) to make the comparison fair, since Snort/Iptables IPS does not have extra NR capability beside drop.

Fig. 7.1 is the comparison evaluation result between SDN-based mitigation and traditional mitigation. In the openflow-based mitigation, we can find openflow switch

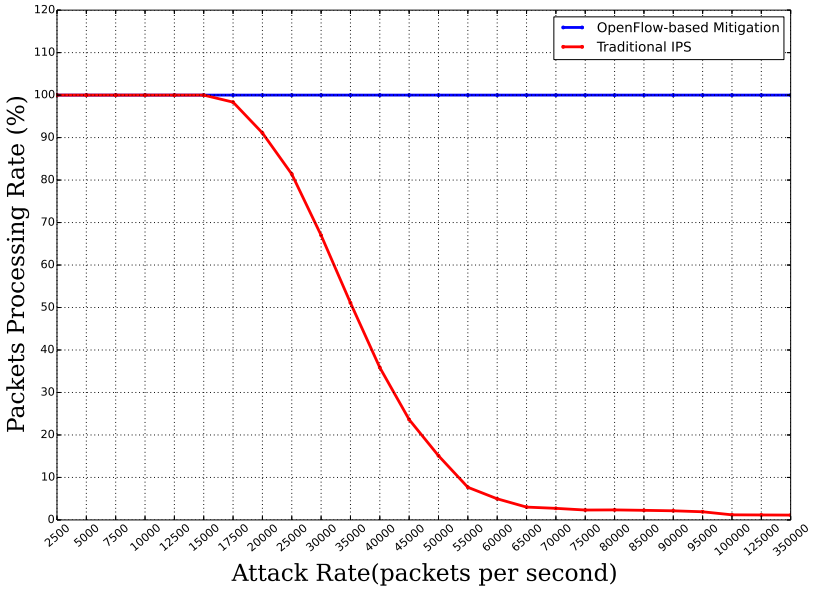


Figure 7.1: Comparison Evaluation between SDN-based and Traditional Mitigation

can handle 100% packets even the attack rate reaches to 350,000 packets per second. The reason why openflow switch has such a good performance is that it can block all the already known malicious packets before they are forwarded. However, in traditional mitigation, the mechanism of Iptables and Snort can not fully prevent the impact of flooding attack because all the traffic packets are always congested into Iptables Queue. The capability of Snort or other traditional mitigation is not able to process all the traffic packets when the attack rate exceeds 17,550 packets per second. In SDN-based mitigation mechanism, the detection kernel is separated from countermeasure module, the mitigation is applied in the data plane of the switch in line-rate, which guarantees the detection engine is free of already known flooding attack and packet processing function still can work properly even system is under overwhelming network traffic.

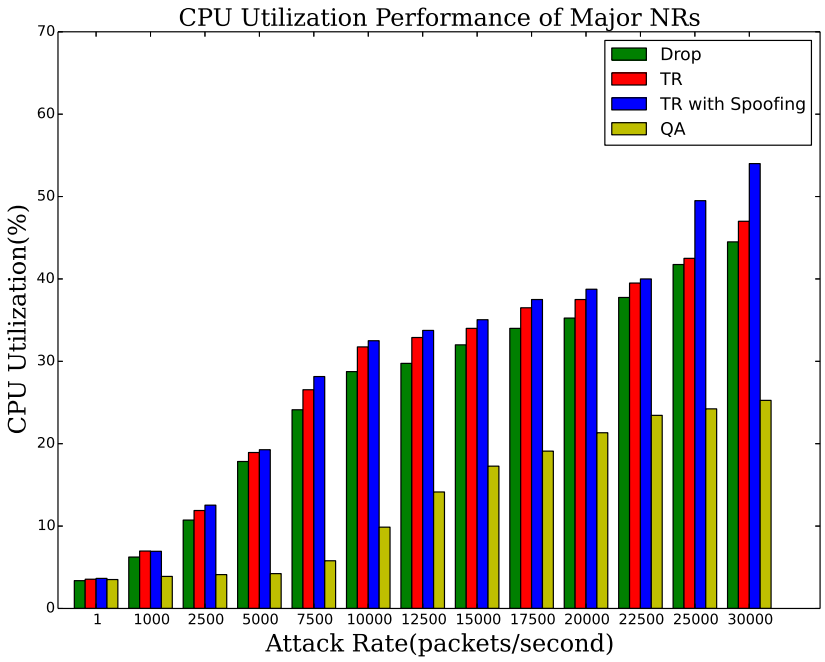


Figure 7.2: Major Mitigation Options Evaluation

In Fig. 7.2, we evaluate four major network reconfiguration in terms of the CPU utilization in an Open vSwitch. We attack a host in our network by using packet generator[47] to create certain traffic packets. We apply four major SDN-based mitigation options to tackle this attack and evaluate their performance. The traffic redirection approach is implemented by using destination IP & MAC rewriting; while TR with fake reply is implemented by modifying not only destination IP & MAC address but also source IP & MAC of victims to forge attackers. By using this action, attacking traffic will be forwarded to a security appliance that is able to spoof the attacker by replacing the source address of reply packet with victims' IP & MAC address. TR with spoofing feature consumes a more resources than the pure TR because switch modifies more packet fields to spoof attackers. The default countermeasure, i.e., drop packets, consumes less system resources because the switch does not modify the fields of matching flows and it just simply drops them (output to a non-existing virtual port in POX controller implementation). In the QA scenario, it has the best performance among all mitigations because the rate limiting action is performed based on Open vSwitch native mechanism, which means excess packets will be discarded and OVS does not have to inspect and match the packet with all kinds of fields.

Beside evaluating the system (CPU) and network (TCP bandwidth) performance of TR, Fig. 7.3 provides the performance on the capacity of OVS and secure appliance VM, e.g., DPI checker. In this scenario, we use TR (port rewriting) to redirect all the attacking traffic to a DPI checker VM (4 vCPUs, 2,048 MB memory, Ubuntu 12.04 server OS). When the attacking packet rate is increased from 1,000 packets per second up to 400,000 packets per second, the OVS can handle all the traffic without any packet loss, which validates that the packet process capacity of OVS is expected. The receive packet means the percentile of OVS redirected traffic has been received

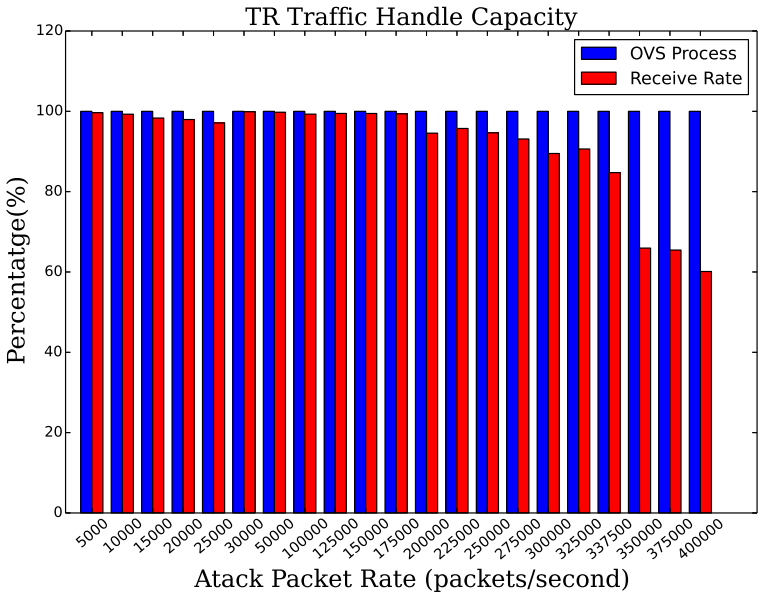


Figure 7.3: TR Traffic Handle Capacity

by the DPI checker's VIF. When the packet rate reaches the 400,000 packets per second (we assume that there is no more than 400,000 packets generated in a single physical server per second), the OVS can still handle 100% of the packet and security appliance can receive 60% of them due to its VIF capacity.

Fig. 7.4 shows the health traffic forwarding capability when we use both mitigations to protect our network from flooding attack. To compare the difference, we apply SDN-based mitigation and traditional mitigation as security proxy to sit between two virtual hosts. This security proxy is responsible to forward health traffic packets and block malicious packets. We use hacking tools [48] to generate the DoS attacking packets towards the security gateway at different attacking rate. Ping of Death (PoD) and SYN flood attack are two major DoS attacks that are used in this experiment. We ask one VM to send health packets to another VM through gateway at the rate of average normal traffic rate and the gateway is attacked as the same

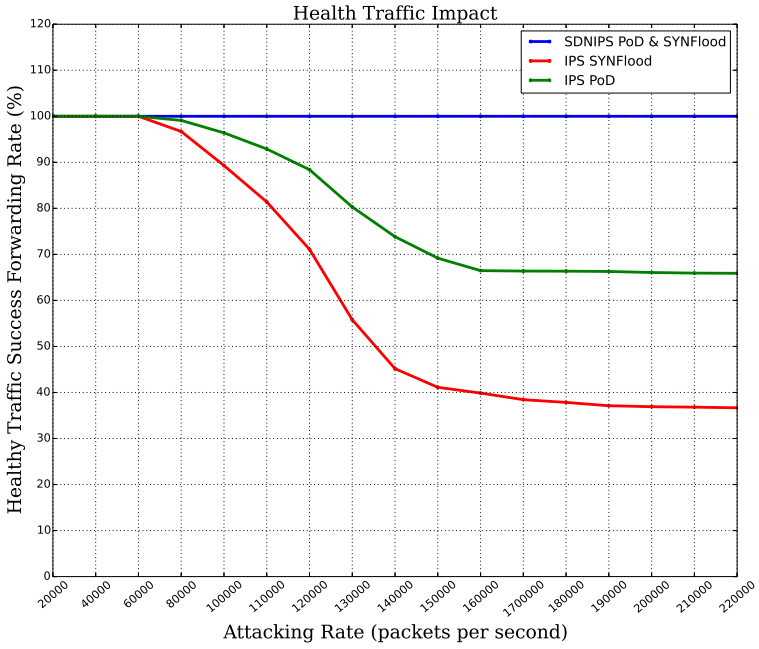


Figure 7.4: Health Traffic Impact

time. In traditional mitigation, malicious packets are firstly captured by detection engine and they will be dropped by Iptables in the queue. In SDN-based mitigation, the openflow switch fulfills the same goal and it drops these packets in the data plane which is more efficient and faster. This mechanism increases the system performance dramatically because the malicious traffic is handled by openflow switch fast path in line rate. As we can see from the Fig. 7.4, SDN-based mitigation under both type of flooding attacks can protect health traffic 100 % from malicious traffic flows. This means all normal traffic can be properly forwarded when we apply SDN-based mitigation to protect network even they are under significant traffic stress. For traditional mitigation, the health traffic success forwarding rate has decreased around 69% and 37% under PoD attack and SYN flood respectively when the attack increases to 150,000 packets per second because of the weakness of traditional mitigation mechanism.

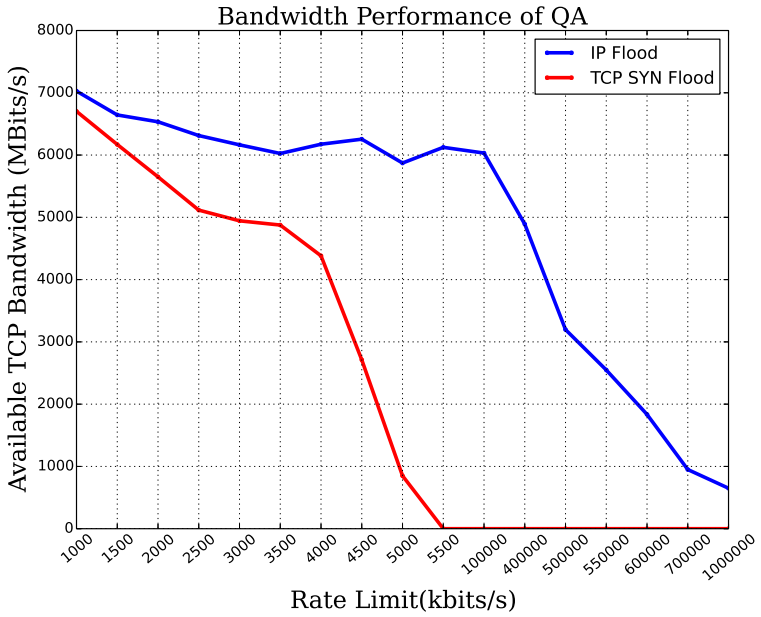


Figure 7.5: Bandwidth Performance of QA

Fig. 7.5 shows how QA can protect TCP bandwidth at different speed limit rate. We attack our network by IP flood and SYN flood in two experiments. Both of them try to crush our network by sending overwhelming traffic packets at a rate of 300,000 packets per second. We configure the incoming port of attacker in the OVS to restrict attacking traffic before being processed. The default incoming speed is 10Gbits/s without any rate limit in XenServer OVS. The range of the speed limit is from 1 Mbits/s to 10Gbits/s. For IP flooding attack, the available TCP bandwidth is in good condition (around 8 Gbits/s) when the rate limit is less than 5 Mbits/s.This means we can make sure the victim host will not be affected by flooding attack when we set the rate limit under 5 Mbits/s. We we raise the rate limit of incoming port, more bandwidth is occupied by the attacking traffic and less bandwidth is available for victim. In SYN flood attack, when the speed limit increases to 4 Mbits/s, the TCP

bandwidth starts degrading because TCP SYN flood will establish TCP connection between attacker and victim. This will definitely impact the TCP bandwidth performance. When the speed limit is above 5.5 Mbits/s, almost no available bandwidth can be used for the victim. This evaluation result provides a valuable reference for any system when deploying the QA NR as a security mitigation.

In summary, the evaluation comparing between proposed SDN-based mitigation and traditional IPS mitigation validates the analysis mentioned before, which is that the SDN-based IPS mitigation has better network and security performance in cloud networking environment.

Chapter 8

CONCLUSION

In this thesis, we propose a SDN-based IPS development framework in cloud computing environment. The goal of this framework is to efficiently and effectively development a SDN-based defensive system with detection,analysis module, and perform the mitigation. The key challenge, which we address, is to build a traffic information platform to collect, organize traffic information for developer. Furthermore, we generate three security modules for developers to construct a full lifecycle defensive system. Some security applications such as Snort Agent, Topology Manager, Network Reconfiguration Pool are planted in the module as the applicable security tools and guideline to facilitate IPS development. We build a DDoS defensive system as a sample to explain how our system works and how to address a realistic security problem. We evaluate the proposed development framework, the advantage of SDN-based mitigation and how SDN technology improve cloud system security. We provide all these evidences to prove that our development framework benefits SDN-based IPS development in cloud system.

References

[1] Wikipedia, "Cloud Computing." [Online]. Available: http://en.wikipedia.org/wiki/Cloud_computing

[2] "SourceFire Inc." [Online]. Available: http://www.snort.org

[3] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," in *IEEE Transactions on Dependable and Secure Computing (TDSC), Special Issue on Cloud Computing Assessment*, 2013.

[4] 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.

[5] "Openflow switch specification version 1.4.0." [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[6] Open vSwitch, http://openvswitch.org/.

[7] Citrix System Inc. [Online]. Available: http://www.citrix.com

[8] docs.openstack.org, *OpenStack Compute Starter Guide*, August 2011.

[9] CloudStack, http://cloudstack.apache.org/.

[10] V. Inc., "Vmware vcloud networking and security overview," in *white paper*, 2012.

[11] N. L. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks."

[12] J. R. Ballard, I. Rae, and A. Akella, "Extensible and Scalable Network Monitoring Using OpenSAFE," in *INM/WREN*, 2010.

[13] J. H. Jafarian, E. AI-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *HotSDN*, 2012.

[14] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *NDSS*. The Internet Society, 2013.

[15] K. Vieira, A. Schulter, C. Westphall, and C. Westphall, "Intrusion detection for grid and cloud computing," *IT Professional*, vol. 12, no. 4, pp. 38–43, July 2010.

[16] S. Roschke, F. Cheng, and C. Meinel, "An extensible and virtualization-compatible ids management architecture," in *Information Assurance and Security, 2009. IAS '09. Fifth International Conference on*, vol. 2, Aug 2009, pp. 130–134.

[17] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer, "Stateful intrusion detection for high-speed network's," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, 2002, pp. 285–293.

[18] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter, "Network-wide deployment of intrusion detection and prevention systems," in *Proceedings of the 6th International COnference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 18:1–18:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921192

[19] M. T. Goodrich, "Probabilistic packet marking for large-scale ip traceback," *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 15–24, Feb. 2008. [Online]. Available: http://dx.doi.org/10.1109/TNET.2007.910594

[20] A. Belenky and N. Ansari, "Ip traceback with deterministic packet marking," *IEEE Communications Letters*, vol. 7, no. 4, pp. 162–164, 2003.

[21] S. Yu, W. Zhou, R. Doss, and W. Jia, "Traceback of ddos attacks using entropy variations," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 412–425, 2011.

[22] J. Francois, I. Aib, and R. Boutaba, "Firecol: A collaborative protection network for the detection of flooding ddos attacks," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 6, pp. 1828–1841, Dec 2012.

[23] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, Oct 2010, pp. 408–415.

[24] G. Yan, R. Lee, A. Kent, and D. Wolpert, "Towards a bayesian network game framework for evaluating DDoS attacks and defense," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 553–566. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382255

[25] M.-Y. Su, G.-J. Yu, and C.-Y. Lin, "A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach," in *Computers & Security*, Feb 2009, pp. 301–309.

[26] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proceedings of the 2012 20th IEEE International Conference on Network Protocols (ICNP)*, ser. ICNP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ICNP.2012.6459946

[27] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 121–126. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342466

[28] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, 2013, pp. 413–424.

[29] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient openflow-based networking," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 933–939.

[30] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep 1990.

[31] Y. Choi, "Implementation of content-oriented networking architecture (cona): A focus on DDoS countermeasure."

[32] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-defined ids for securing embedded mobile devices," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, Sept 2013, pp. 1–7.

[33] S. Yu, Y. Tian, S. Guo, and D. Wu, "Can we beat DDoS attacks in clouds?" pp. 1–1, 2013.

[34] Murphy McCauley. [Online]. Available: http://www.noxrepo.org/pox/

[35] E. Brodsky and B. Darkhovsky, *Nonparametric Methods in Change Point Problems*, ser. Mathematics and Its Applications. Springer, 1993. [Online]. Available: http://books.google.com/books?id=Lu_XN8KswvwC

[36] A. Tartakovsky, B. Rozovskii, R. Blazek, and H. Kim, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods," *Signal Processing, IEEE Transactions on*, vol. 54, no. 9, pp. 3372–3382, Sept 2006.

[37] W. Lu and H. Tong, "Detecting network anomalies using cusum and em clustering," in *Advances in Computation and Intelligence*. Springer, 2009, pp. 297–308.

[38] H. Wang, D. Zhang, and K. Shin, "Detecting syn flooding attacks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, June 2002, pp. 1530–1539.

[39] T. Peng, C. Leckie, and K. Ramamohanarao, "Proactively detecting distributed denial of service attacks using source ip address monitoring," in *In Proceedings of the Third International IFIP-TC6 Networking Conference (Networking 2004*. Springer, 2004, pp. 771–782.

[40] "MySQL." [Online]. Available: http://www.mysql.com/

[41] A. Alhomoud, R. Munir, J. P. Disso, I. Awan, and A. Al-Dhelaan, "Performance evaluation study of intrusion detection systems," in *The 2nd International Conference on Ambient System, Networks and Technologies*, 2011.

[42] A. Yaar, A. Perrig, and D. Song, "Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 10, pp. 1853–1863, Oct 2006.

[43] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2001, pp. 878–886 vol.2.

[44] "Mininet." [Online]. Available: http://mininet.org/

[45] L. D. Stein and J. N. Stewart, in *The World Wide Web Security FAQ, Version 3.1.2*, Feb 2002.

[46] HP Opencall Software, "http://gull.sourceforge.net/."

[47] R. Olsson, "pktgen the linux packet generator."

[48] "Back Track Linux." [Online]. Available: http://www.backtrack-linux.org