2020

A Representational state transfer web tool for firewall service management and monitoring in a Local Area Network.

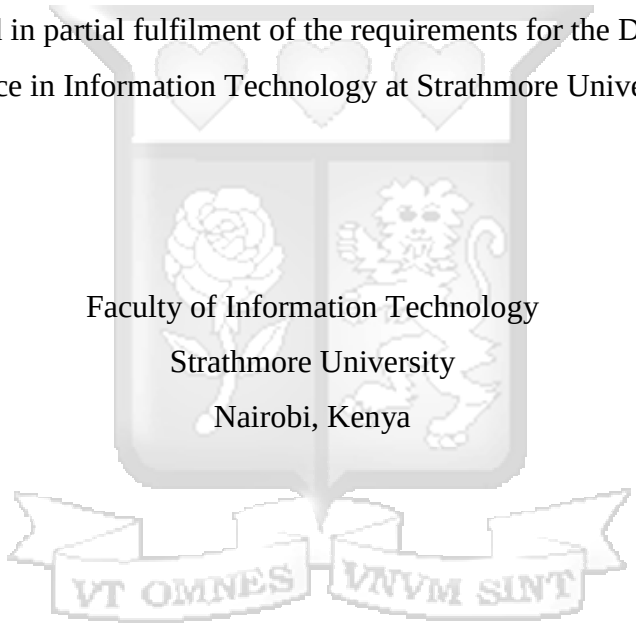
Guchu, Mary Wambui
Faculty of Information Technology
Strathmore University

A Representational State Transfer Web Tool for Firewall Service Management and Monitoring in a Local Area Network

Mary Wambui Guchu

A Thesis Submitted in partial fulfilment of the requirements for the Degree of Master of Science in Information Technology at Strathmore University

Faculty of Information Technology
Strathmore University
Nairobi, Kenya



February 2020

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Mary Wambui Guchu

Signature: 

Date: 29/06/2020

Approval

The thesis of Mary Wambui Guchu was reviewed and approved by the following:

Prof. Ismail Ateya Lukandu, D. Sc.
Director, Office of Faculty Affairs,
Strathmore University.

Dr. Joseph Orero,
Dean, Faculty of Information Technology,
Strathmore University.

Dr. Bernard Shibwabo,
Director of Graduate Studies,
Strathmore University.

Acknowledgment

First, I owe special thanks to God Almighty for giving me the grace to complete this thesis. I would like to express my sincere gratitude to my supervisor Prof. Ateya. His constructive criticism, guidance and patience enabled me to complete this thesis.

I wish to express my gratitude to my loving husband John T. Mbau for his wonderful support and words of encouragement throughout this course. My sister Dr. Salome Guchu, your prayers, encouragement and well wishes energized me.

Dedication

I dedicate this thesis to my children Purity, Princess and Mark William. You were my source of encouragement and inspiration during the entire period. I am humbled for having you in my life.

Abstract

Firewalls play a very important role in managing and securing organization resources. Firewalls implementation come in different designs and architectures. However, one thing that is common in most implementations is that they are complex to manage and configure. As the number of rules and policies on a firewall grow, rule proliferation, the complexity in managing these services grow. Firewall are mostly implemented using UNIX based systems. A number of them that range from IP tables and uncomplicated firewalls (UFW) are application that run host based firewalls system on a Linux environment. These services are mostly managed, configured and orchestrated using the command line. As the network grows and the need for complex rules arises, the management of such firewalls present a challenge. Developing web interfaces to monitor firewalls and network operations is one approach. However, a huge part of the challenge is in service deployment and orchestration of firewall services. This research proposes a system that system and network administrators can use to easily manage and configure firewalls without necessarily logging in to the console of the devices or machines or setting up secure shell (SSH) session to the command line interface (CLI). The web tool that this work proposes bases its argument on creating a web interface that allows system and network administrators to centrally manage firewall services on simple HTTP like interface with PHP REST APIs running in the background. This work used a test driven methodology coupled with agile development, specifically extreme programming to develop the RESTful web tool for easy firewall service management and monitoring.

Keywords: Firewall, Firewall Management, REST

Table of contents

Declaration ..... *i*

Acknowledgment ..... *ii*

Dedication ..... *iii*

Abstract ..... *iv*

Table of contents ..... *v*

List of Figures ..... *ix*

List of tables ..... *xi*

Abbreviations and Acronyms ..... *xii*

Definition of Terms ..... *xiii*

Chapter 1 : Introduction *1*

1.1 Background **1**

1.2 Problem Statement **4**

1.3 General Objective **5**

1.4 Research Objectives **5**

1.5 Research Questions **5**

1.6 Justification **5**

1.7 Scope and Limitation **6**

Chapter 2 : Literature Review *7*

2.1 Introduction **7**

2.2 Firewall Management **7**

2.3 Models and Frameworks of Orchestrating Dynamic Firewall Services **9**

2.3.1 MacAfee Next Generation firewall 9

2.3.2 Cisco Adaptive Security Appliance (ASA) 9

2.4 Richard Maturity Model for Classifying Restful web APIs **10**

2.5 RESTful API Frameworks **11**
2.5.1 Lumen-micro Framework 11
2.5.2 Slim Framework 12
2.5.3 Laravel framework 12
2.6 Algorithms and Design Strategies for REST APIs **12**
2.7 RESTful Architectural Style **15**
2.8 REST API and Web Services **18**
2.9 Conceptual Frame Work **20**
Chapter 3 : Research Methodology ***21***
3.1 Introduction **21**
3.2 Research Design **21**
3.3 Location of the Study **22**
3.4 Target Population **22**
3.5 Sampling Frame **22**
3.6 Data Collection **23**
3.7 Data Analysis **23**
3.8 System Development Methodology **23**
3.8. 1 System Implementation 26
3.9 Research Quality **26**
3.10 Ethical Consideration **26**
Chapter 4 : System Analysis and Design ***28***
4.1 Introduction **28**
4.2 Requirement Analysis **28**
4.2.1 Functional Requirements 28
4.2.2 Non Functional Requirements 29
4.3 System Architecture **29**
4.4 System Design **30**
4.3.1 Use Case Diagram 30
4.3.2 Sequence Diagram 31

Chapter 5 : System Implementation and Testing *33*

5.1 Introduction **33**

5.2 Model Components **33**

5.3 Test bed set up **34**

5.4 System Implementation **36**

5.4.1 Webpage GUI development 36

5.4.2 PHP APIs to Manage Firewall Service 36

5.5 System Testing **39**

5.5.1 Firewall management test results 39

5.5.2 Firewall configuration test results 41

5.5.3 System Testing Classes 43

5.5.4 System Testing Results 44

Chapter 6 : Discussion *45*

6.1 Introduction **45**

6.2 Discussion of Results **45**

6.3 Challenges Faced **46**

6.3.1 System configuration challenges 46

6.3.2 Data display and presentation challenges 46

Chapter 7 : Conclusion and Recommendation *48*

7.1 Conclusion **48**

7.2 Recommendation **49**

7.3 Future Works **49**

References *51*

Appendices *55*

Appendix A: PHP Code to Allow Traffic Based on Port Numbers **55**

Appendix B: PHP Code to Block Traffic Based on Port Numbers **56**

Appendix C: Simple Firewall Management Panel **57**

Appendix D: Simple Web Page Form **58**

Appendix E: Port Numbers Firewall Management **59**

Appendix F: IP Addresses Firewall Management 60

Appendix G: Simple Web UI for Firewall Configuration..... 61

Appendix H: Cisco ASA Firewall Configuration Page 62

Appendix I: Originality Report 63

List of Figures

Figure 2.1: Number of Errors as a Function of Rule set Complexity 8
Figure 2.2 A High-Level Architecture of REST Client on ASA 10
Figure 2.3: Richardson Maturity Model 11
Figure 2.4: Uniform-Client-Cache-Stateless-Server 16
Figure 2.5: Separation of concerns 16
Figure 2.6: Stateless 17
Figure 2.7: Layered System 17
Figure 2.8: Cacheable 18
Figure 2.9: Code on Demand 18
Figure 2.10: Key Concepts in RESTful API 19
Figure 2.11: Conceptual Framework 20
Figure 3.1: Test Driven Methodology Diagram 24
Figure 3.2: Extreme Programming Methodology 25
Figure 4.1 General System Architecture 29
Figure 4.2: Use Case Diagram 31
Figure 4.3: Sequence Diagram 32
Figure 5.1: Verification of a running iptables 34
Figure 5.2: Verification of a working Apache web server 34
Figure 5.3: Verification of correctly installed and working PHP 35
Figure 5.4: Verification of Apache added to the sudoers file 35
Figure 5.5: JavaScript working on button actions 36
Figure 5.6: PHP code running GET method for Firewall Status 37
Figure 5.7: PHP code to block IP addresses 38
Figure 5.8: PHP Code to Allow IP Addresses 38
Figure 5.9: Status, save and disable options 40
Figure 5.10: Firewall status and rules results 40
Figure 5.11: Command and output to display status 40
Figure 5.12: Command and output to save configurations 41
Figure 5.13: Disable command 41
Figure 5.14: destination port number accept 41
Figure 5.15: destination port number drop 42

Figure 5.16: Src IP drop.....42
Figure 5.17: Src IP accept.....42

List of tables

Table 5.1: System Testing Classes.....43

Table 5.2 System Testing Results.....44

Abbreviations and Acronyms

ACL	-	Access Control List
API	-	Application Programming Interface
ASIC	-	Application Specific Integrated Circuit
BASH	-	Bourne Again Shell
CLI	-	Command Line Interface
HTML	-	Hypertext Markup Language
IP	-	Internet Protocol
IPv6	-	Internet Protocol Version 6
JSON	-	JavaScript Oriented Notation
LAN	-	Local Area Network
NAT	-	Network Address Translation
SMC	-	Security Management Centre
SSH	-	Secure Shell
UFW	-	Uncomplicated Firewall
URI	-	Uniform Resource Identifier
WAN	-	Wide Area Network
WWW	-	World Wide Web
XML	-	Extensible Markup Language

Definition of Terms

SMC -A management center provided by The MacAfee Next Generation firewall. The management client or an API provide access and configuration of the management center (McAfee, 2019).

Chapter 1 : Introduction

1.1 Background

Protection against network attacks is no longer an option in most networked organizations. Firewalls play the role of putting a point of defense between a network and the internet or a network and other networks not necessarily classified as the internet. The design of firewalls is to protect the networks from internal and external threats (Nikolaidis, 2000). The threats that firewalls protect range from unauthorized access to organizations resources, compute, network and storage resources. This unauthorized access can lead to leak in information, damage of computing resources, viewing or modification of sensitive information. The aftermath effects of successful attacks lead to loss in company reputation and trust with its clients.

Owing to the nature and the effects of the threats computing resources are exposed to, the technical, implementation and deployment of a firewall should be thorough. The envisioned operation of a firewall is to allow authenticated requests while denying and blocking unauthenticated, harmful and malicious requests. Stateful and stateless firewalls are the two broad categories of the firewall. Stateful firewalls keep information about the connections that pass through them. Security policies and rules used in stateful firewalls are against the state table or a connection table. State tables are able to add and maintain information about a connection during the operation of a stateful firewall.

Stateless firewalls operate from a different point of view. At times, they are synonymous to packet filtering firewalls. The state of the connection is not looked by these firewalls. The packets that pass through the firewall are of the most concern. Stateless firewalls use packet information, source and destination logical address and port numbers to either permit or drop packets. According to Nikolaidis (2000), these firewalls combined with other firewalls provide tougher security.

Another stateless firewall implementation goes beyond packets filtering to application level filtering. Nikolaidis (2000) call them application level firewalls. They provide a better control environment of incoming and outgoing packets. Dropping and permitting of traffic on this firewalls is based on the OSI layer but goes beyond layer 4 on packet filtering, to layer 7, the application layer (Nikolaidis, 2000).

There are host-based firewalls shipped inside the box with Linux distributions. Alternatively, others are set up in Linux boxes to provide enhanced security. IP tables is an example of a packet filter firewall that ships off the shelf in most Linux distributions. Servers that have enabled iptables do the monitoring and filtering of both IPv6 and IPv4. Shorewall is an enhancement of IP tables as it combines IP tables and IP chains built in the Linux kernel as a firewall protection programs (Negus, 2015). Uncomplicated Firewall (UFW) eases the complexity that comes with writing iptables rules. As the name suggests, it is less complicated. Similarly, a CLI configures UFW. It is from the open source community. Iptables and Shorewall are also open source (Negus, 2015).

Firewall is rule based software with two filtering actions; allow and block, these actions are influencing the network traffic passing through the firewall. These actions for allowing or blocking packets performs according to a set of static configuration rules that use only information contained in the packet, such as source and destination network addresses, port and protocol (Voronkov et al., 2015).

The idea of developing tools for firewall service management and monitoring is obviously not new. Firewall vendors and network administrators have developed several management tools to provide smooth network operations. These existing tools are designed to support a particular usage scenario which are requirement specific and do not provide much more than what is required. Furthermore, the tools lack proper documentation and do not guarantee a bug free software. Comparatively, there are enterprise solutions provided by Cisco and other dealers with better documentation and user interface but not flexible. These tools are designed for management of a specific expected data and it is not possible to change the source code of such solutions to adapt with the changing needs of the organizations firewall usability and extensions.

Cisco has a graphical user interface tool under name Adaptive Security Device Manager (ASDM). This tool manages the Cisco Adaptive Security Appliance (ASA) firewalls and the Cisco AnyConnect Secure Mobility Client through a local, web-based interface. The software provides the ability to configure and manage Cisco firewall devices. Moreover, it provides a real time log viewer and monitoring dashboard that offers a view of firewall appliances status and health. Besides firewall management, the software offers troubleshooting features and debugging tools such as packet trace and packet capture. Furthermore, it is a java applet; providing a REST interface, which allows clients to perform create, read, update, and delete

(CRUD) operations on ASA resources. The API module builds on the HTTPS protocol and REST methodology (Frahim et al., 2014).

Firewall builder is another Linux graphical user interface tool primarily used to build firewall configurations and policies for multiple firewalls or machines based on iptables, ipfilter, OpenBSD pf, Cisco ASA & PIX, or Cisco FWSM. This software uses objects and functionality such as drag and drop; search and replace to assist ease configurations. Firestarter, gufw, peergurdian Linux and Vuurmuur are also firewall management software used to monitor and build firewall configurations. Firestarter provides management of the Iptables firewall. It allows the user to manually create inbound and outbound policies and define a whitelist or blacklist. Moreover, it enables the user to view firewall events in real-time and get stats on active network connections, including any traffic routed through the firewall. It also provides internet connection for sharing activity (Ali et al., 2018). Gufw is the graphical user interface for the uncomplicated firewall (UFW), which Ubuntu uses as its default. It provides basic inbound and outbound policies; and has preconfigured rules for various applications and multiple profile support. Peerguardian linux contain blacklists of IP addresses known to contain honey pots and prevent the user's P2P software from downloading files from these blacklisted sites (Raja et al., 2012).

Vuurmuur firewall manager being an application built on top of iptables, its works with Linux kernels 2.4 and 2.6. It has a simple and easy to learn configuration that allows both simple and complex configurations. Its managed via the Ncurses graphical user interface in console with no X required, and via SSH. It offers real-time monitoring of connections and bandwidth usage. It also supports traffic shaping and anti-spoofing features; and works with Suricata IPS and Snort. In addition, it has no REST APIs for configuration (Rebahi et al., 2015).

The identified firewall solutions have limitations because of which this works develops the REST web tool for firewall management. These proprietary solutions do not have available open source code for customization of software; the dealers do not allow editing of source code in order to safeguard software integrity. Thus, the client has no flexibility to introduce options but to work with the only available features. The limitation arises because of limited or complex options integrated in the developed software and a client flexibility to introduce new or remove option is restricted. Some of the discussed solutions has no REST API, which is notable for its remarkable layer of tractability (Fielding, 2000). Moreover, the proprietary firewall solutions

presents the user with a detailed web page, in which most of this information might not be useful in achieving the tasks the user is set to achieve. Due to these reasons, this works proposed the design and development of an easy to use web tool that manages iptables Linux firewall service; which is relevant, and flexible for future extensions. The proposed solution has a central point as the web page where the data passes to and from the firewall application. From the web page, users will send HTML requests to the web server. The web server passes them over to multiple PHP API scripts for processing based on the requests. The PHP scripts validates the data and use it to manage or configure the iptables firewall service by running BASH commands within the scripts. The PHP API scripts hold the results and send them back to the web server. Finally, the web server sends the results to the web page for presentation.

1.2 Problem Statement

Firewalls are very important network functions in a LAN or a WAN. Firewalls can run on standalone devices or as an implementation on normal PC hosts as a virtual appliance. Firewalls can also run on a physical bare metal PC, server or network device. Popular firewalls deployment and administration uses Linux based modules like iptables, Firewalld and UFW (Negus, 2015). Administration of this network service, a firewall, require the security and network administrators to physically login to the console and terminal screens of these devices to check the status or write custom Bourne Again Shell (BASH) commands for administration. These tasks are mostly complex and restrictive. These commands to execute the tasks are equally complex in structure and the probability of error is high when they run multiple times to replicate the same function. The process is restrictive because it is platform dependent (Wool, 2010). An administrator has to either login to the physical machine or run a terminal client service through SSH to configure their firewall and check the status of the service.

Currently, the most commonly used firewall management tools are web based developed by the firewall dealers limited to edit the dealers firewall products. Thus, there is need for a simple and flexible firewall management tool. Moreover, a lot changes in commercial necessities and developments, as a result administrators require a firewall management tool to emulate the corporate needs. Some commands are generic to most Linux environments. An opportunity is to automate them to run from the click of a HTML button interfaced with Representational State Transfer (REST) APIs on a web interface. The developed web tool will enable the administrator to manage firewall services to the extent of usability and observation.

1.3 General Objective

The general objective of this research is to develop a Representational State Transfer web tool for firewall service management and monitoring in a local area network.

1.4 Research Objectives

i. To analyze the challenges of managing current firewall services
ii. To review existing models and frameworks of orchestrating firewall services
iii. To design and develop a Representational State Transfer web tool for orchestrating firewall services
iv. To test the developed tool.

1.5 Research Questions

i. What are the challenges of managing current firewall services?
ii. What are the existing models and frameworks for orchestrating firewall services?
iii. How can the Representational State Transfer web tool for orchestrating firewall services be designed and developed?
iv. How will the developed tool be tested?

1.6 Justification

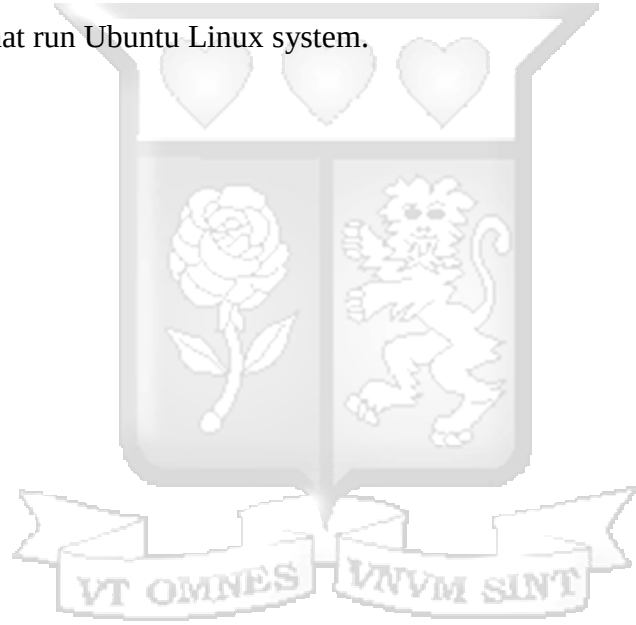
This research presents an opportunity for system administrators and developers to circumvent the previously tedious process of managing, monitoring and deploying firewall services and rules. Through the implementation of this work, a scenario where system locking in administrators to environment specific commands to run firewall services will not occur. Provided the system administrator knows the services on their network that they want to invoke control over, abstracting the inner working of firewall deployment systems from them using the web interface is important.

Successful implementation of this research presents an opportunity for reduced erroneous firewall configuration due to duplication of commands to deploy predictable and repetitive services. In addition, this research seeks through successful implementation to reduce the amount of time and effort needed to manage, deploy and monitor firewall services.

1.7 Scope and Limitation

It is important to clarify that this work's aim is not to alter the inner working of any firewall application. This work does not seek to overhaul and come up with a completely new firewall services from scratch. The focus of this research is to work on best existing firewall service application to provide a better management, monitoring and deployment solution for the existing firewall services.

The geographical scope of implementation this firewall of choice is to offer protection to a single LAN that connects to the internet and has packets flowing in and out of the LAN to the internet. This research acknowledges that most firewall applications run on UNIX based systems (Negus, 2015). This work focuses on managing, monitoring and deploying firewall application services that run Ubuntu Linux system.



Chapter 2 : Literature Review

2.1 Introduction

RESTful web services as an alternative solution for easy service management and deployment of firewall is a popular approach. Firewalls have previously been managed using CLI and console logins to deploy and orchestrate these services. This chapter creates a tie between these solutions. A discussion of the challenges that span from rule proliferation and probability of errors when configuring distributed firewalls make up this chapter.

This chapter equally presents a discussion on using REST architecture to deploy web services. This chapter focuses on the REST architecture, how the REST client and REST servers communicate. The chapter also provides a discussion on the requirements to writing a REST API. Design strategies used to write REST APIs and services form the discussion in this chapter. This chapter then analyses industry solutions like Cisco ASA firewall and MacAfee Next Generation firewall that have used RESTful approach to manage their firewalls.

2.2 Firewall Management

The magnitude and complexity of firewall applications relies on the volume of traffic brought in from workloads to the enforcement points (Illumio, 2019). Depending on how fast the motion and the rate of change, managing such statistically IP based firewall services become more complex. Another research done by (Wool, 2010) tried to associate the complexity of firewall rules to probable error rates. The equation 1 shows the results presented (Wool, 2010).

$$RC = \#Rules + \#Objects + \binom{\#Interfaces}{2} \quad \text{.....Equation 1}$$

RC was used to represent rule complexity while #Rules is the raw number of rules in the rule set, #Objects is the number of net- work objects, and #Interfaces is the number of interfaces on the firewall. However, a caveat that this work provides is, as we increase the number of interfaces to manage on a firewall, the more complex it becomes and the complexity to managing these firewalls. The general findings from the research was complexity of a firewall has a direct correlation to the number of detected configuration errors. Sample data from the research shows that a rule set that has rule set complexity of 1000 is likely to have 14

errors while a rule set complexity of 10,000 is likely to have 22 errors (Wool, 2010). Figure 2.1 shows this correlation.

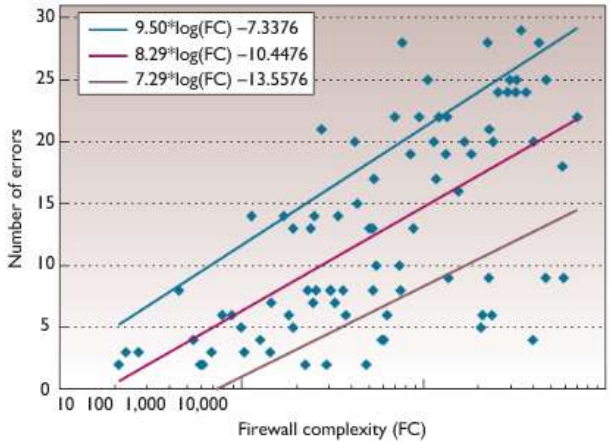


Figure 2.1: Number of Errors as a Function of Rule set Complexity

From the findings of (Wool, 2010) research, an opportunity presents itself for this work to be conducted. This research proposes to simplify this firewall configuration process by simply eliminating redundancy, which contributes to higher probabilities of errors. Illumio (2019) discuss further on rule proliferation when managing firewalls. Their discussion goes beyond the complexity of the rules to the resultant effect that come with rule proliferation. When organizations implement long and complex rules, which are most of the time centralized in one Application Specific Integrated Circuit (ASIC) firewall device, it becomes a problem when they are supposed to respond to changes in policy and troubleshooting.

Due to this delayed response to changes to adhere to policy regulations, out of date rules present security threats that can be used to access business critical applications. In addition, as organization scamper to adhere to policy regulations and the ever-dynamic nature of cyber security threats, rule duplication is a risk. This can come from a situation where new rules override old rules with the same functionality. However, in such an instance, the old rule still exists, non-deleted. Rule proliferation also presents administrators with a situation of having orphaned rules. These are active rules that allow access to computing resources that have been migrated or brought down.

A good web based monitoring platform that this research proposes seeks to ease the process of monitoring which rules system administrators have enabled on their infrastructure. The tool that this work presents summarizes the inner working of each rule to make it easier for system

administrators to track down duplicated rules. This feature works hand in hand with firewall provisioning services discussed earlier in this section.

2.3 Models and Frameworks of Orchestrating Dynamic Firewall Services

2.3.1 MacAfee Next Generation firewall

The MacAfee Next Generation firewall provides a management center. The management client or an API provides access and configuration of the management center. The SMC API allows system administrators to be able to add, edit and remove the following objects: hosts, networks, address ranges, access rules, Network Address Translation (NAT) rules and inspection rules. Besides, it also allows, through the API, uploading a policy to the engine and retrieval or effecting routing changes to the engine (McAfee, 2019).

Use cases successfully implemented through the SMC API based on REST are internal automation of tasks through scripting instead of manually configuration of the tasks from the management client on the software. Besides, there have been successful integration through the API to other third party security policy management and risk management applications like Tufin that specializes in the automation of security policy changes across hybrid platforms while improving security and compliance (McAfee, 2019).

2.3.2 Cisco Adaptive Security Appliance (ASA)

CISCO ASA product line has also developed a REST API client used to configure their firewalls. This is different from their conventional way of managing their services through the CLI or logging into the console or remote login through SSH. Cisco implements this solution uploading a plugin to the ASA flash memory that needs activation from the CLI. This REST API uses the REST standard methods to manipulate resources like network objects or Access Control Lists (ACLs). These methods include Create, Read, Update, Delete which when executed on ASA a status code of 201 with an object created message is returned (Kukan, 2019).

Under the hood, the ASA firewall in communication with the REST API supports the following requests: GET, PUT, POST, DELETE and PATCH to retrieve data from an object. PUT and POST, to add information to an existing object. If it fails, it creates the object. PATCH to modify an object and DELETE to delete an object.

The response structure in the RESTful ASA Client Cisco implementation works in a similar fashion as the HTTP request and response service. The firewall returns either a 200 OK message or a 500 internal error when the server cannot handle a request. Specifically, the 2OX family of responses as describe by (Kukan, 2019) include 200 OK for successful execution of REST requests, 201 Created for successfully created objects, 202 Accepted for accepted requests and 204 No Content for successful request but no content to return. Figure 2.2 shows a high-level architecture of the RESTful Client Cisco ASA firewall.

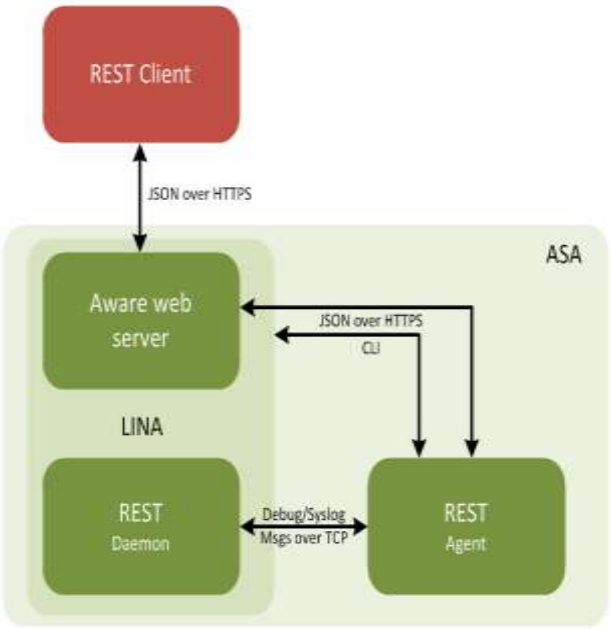


Figure 2.2 A High-Level Architecture of REST Client on ASA

2.4 Richard Maturity Model for Classifying Restful web APIs

Leonard Richardson (2008) developed a Richardson Maturity Model (RMM) used by the developers to determine how well a Web service adheres to REST Principles. This model consists of four levels as shown in Figure 2.3. Level-0 services use “Plain Old XML” (POX) over HTTP to communicate. Each service has exactly one URL (endpoint) and all the requests use the HTTP POST method. Clients invoke different operations on the service by sending different XML payloads. This approach is similar to the traditional RPC model. Thus, we can categorize SOAP-style Web services as level-0 services.

Level-1 services use resources to represent code and data abstractions. Developers identify each resource as a separate URL; the service consists of multiple URLs (endpoints).

However, like level-0 services, these services also rely on a single HTTP method (usually POST). Sending different payloads on the same resource invokes different operations.
Level-2 services use different HTTP methods to invoke different operations on the same resource. The goal of level 2 is for the service to adhere more closely to the underlying semantics of HTTP in terms of its operations and responses. Level-3 services send hypermedia content in response messages that identify the options a client can take for the subsequent operation. Client applications make local state transitions based on the received hypermedia content and navigate their way through the REST API. Level-3 services make effective use of HATEOAS.

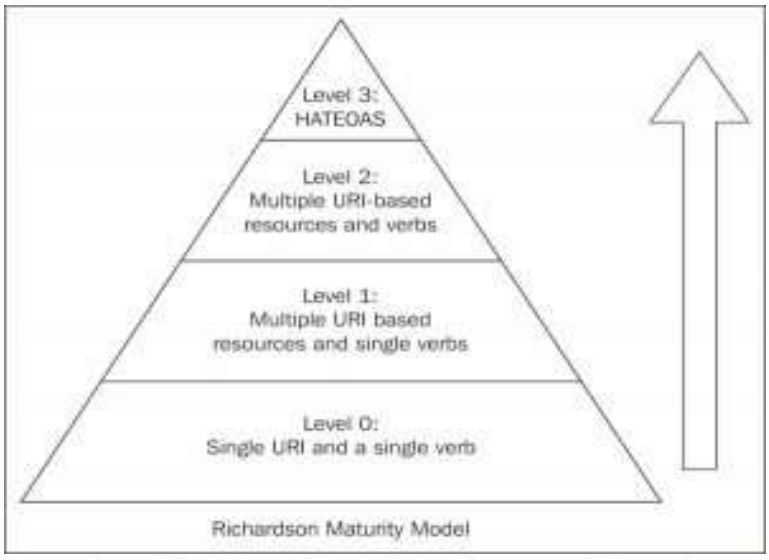


Figure 2.3: Richardson Maturity Model

2.5 RESTful API Frameworks

Frameworks play a significant role in system design as they offer the basis for better understanding of technologies. To build fast and reliable development there are several frameworks in each field; some of the frameworks in existence include:

2.5.1 Lumen-micro Framework

This is fast and lightweight API micro-framework by Laravel using PHP. It has features like Authentication, Authorization, Cache, Database, Encryption, Errors & Logging, Events, and Queues. It is one of the fastest micro-frameworks available.

2.5.2 Slim Framework

This is a powerful PHP 5 framework to create Restful web applications. It has features like Enable or disable application debugging for debug API. If true, Slim displays debugging information for errors and exceptions. It provides developers to create complete PHP web service with only a single PHP file in quick minutes.

2.5.3 Laravel framework

Laravel is one of the frameworks adopted by many developers in developing customized web application faster and easily. Laravel is open source and follows Model View Controller (MVC) that makes it more useful.

2.6 Algorithms and Design Strategies for REST APIs

According to ("How to design a REST API – REST API Tutorial", 2019) understanding the RESTful architecture does not guarantee building a good and efficient REST APIs. There are steps proposed by ("How to design a REST API – REST API Tutorial", 2019) designing REST services. This work presents a design process for REST services for a network-based application.

The first step is identification of object models to present as resources. For a networked environment, these resources include routers, devices, managed entities and device configurations. After identification of the resources or the object, each of them gets an identifier. The object or the resource can also have sub resources. For example, a device can be a resource and the device configuration can be the sub-resource.

After object modeling is complete, creating URIs follows. URIs bring the connection between the resources and the sub resources. The resources are named using nouns. Verbs or operations are avoiding when naming and creating URIs. The Code 2.1 shows a resource device with its sub resources device configuration.

```
/devices
/devices/{id}

/configurations
/configurations/{id}

/devices/{id}/configurations
/devices/{id}/configurations/{id}
```

Code 2.1: Naming Resource Devices with its sub Resources Device Configuration

Determining the representation of the URIs follows. XML or JSON present these URIs. Resource representation involves representing each resource differently or using a collection of resource file. However, when doing this resource representation, it is important to note recommendation to emphasize on keeping the size of the payload small. The sub-resources take the same approach of single resource representation and collective resource representation. Code 2.2 and Code 2.3 shows how to achieve single resource representation and collective resource representation when represented with a device as a resource. Another important thing to note when representing resources, each of them contains a link to itself. Therefore, to access a resource, we need to get through the specific URI.

```
<devices size="2">

    <link rel="self" href="/devices"/>

    <device id="12345">
        <link rel="self" href="/devices/12345"/>
        <deviceFamily>apple-es</deviceFamily>
        <OSVersion>10.3R2.11</OSVersion>
        <platform>SRX100B</platform>
        <serialNumber>32423457</serialNumber>
        <connectionStatus>up</connectionStatus>
        <ipAddr>192.168.21.9</ipAddr>
        <name>apple-srx_200</name>
        <status>active</status>
    </device>

    <device id="556677">
        <link rel="self" href="/devices/556677"/>
        <deviceFamily>apple-es</deviceFamily>
        <OSVersion>10.3R2.11</OSVersion>
        <platform>SRX100B</platform>
        <serialNumber>6453534</serialNumber>
        <connectionStatus>up</connectionStatus>
        <ipAddr>192.168.20.23</ipAddr>
        <name>apple-srx_200</name>
        <status>active</status>
    </device>

</devices>
```

Code 2.2: Resource Representation: Collection of Devices

```
<device id="12345">
    <link rel="self" href="/devices/12345"/>

    <id>12345</id>
    <deviceFamily>apple-es</deviceFamily>
    <OSVersion>10.0R2.10</OSVersion>
    <platform>SRX100-LM</platform>
    <serialNumber>32423457</serialNumber>
    <name>apple-srx_100_lehar</name>
    <hostName>apple-srx_100_lehar</hostName>
    <ipAddr>192.168.21.9</ipAddr>
    <status>active</status>

    <configurations size="2">
        <link rel="self" href="/configurations" />

        <configuration id="42342">
            <link rel="self" href="/configurations/42342" />
        </configuration>

        <configuration id="675675">
            <link rel="self" href="/configurations/675675" />
        </configuration>
    </configurations>

    <method href="/devices/12345/exec-rpc" rel="rpc"/>
    <method href="/devices/12345/synch-config"rel="synch device configuration"/>
</device>
```

Code 2.3: Resource Representation: Single Device

Different resources need different HTTP methods. These methods allow the user to perform operations on their networks. The solution described by ("How to design a REST API – REST API Tutorial", 2019) allowed the user to create, delete and update operation on their network applications. The GET, POST, PUT and DELETE methods browse, create, update and delete resource information respectively. Code 2.4, Code 2.5, Code 2.6, Code 2.7, Code 2.8 and Code 2.9 depict this implementation.

```
HTTP GET /devices/{id}
HTTP GET /configurations/{id}
```

Code 2.4: RESTful HTML Operations: Browse- GET

```
HTTP POST /devices
HTTP POST /configurations
```

Code 2.5: RESTful HTML Operations: Create- POST

```
HTTP/1.1 201 Created
Content-Type: application/xml
Location: http://example.com/network-app/configurations/678678

<configuration id="678678">
    <link rel="self" href="/configurations/678678" />
    <content><![CDATA[...]]></content>
    <status>active</status>
    <link  rel="raw configuration content" href="/configurations/678678/raw" />
</configuration>
```

Code 2.6: RESTful HTML Operations: Create- POST feedback

```
HTTP PUT /devices/{id}
HTTP PUT /configurations/{id}
```

Code 2.7: RESTful HTML Operations: Update- PUT

```
HTTP/1.1 200 OK
Content-Type: application/xml

<configuration id="678678">
    <link rel="self" href="/configurations/678678" />
    <content><![CDATA[. updated content here .]]></content>
    <status>active</status>
    <link  rel="raw configuration content" href="/configurations/678678/raw" />
</configuration>
```

Code 2.8: RESTful HTML Operations: Update- PUT feedback

```
HTTP DELETE /devices/{id}
HTTP DELETE /configurations/{id}
```

Code 2.9: RESTful HTML Operations: Delete- DELETE

2.7 RESTful Architectural Style

REST architecture advocates for six constraints. One of them is a uniform interface as shown in Figure 2.4. For the interface to be uniform, the request to server has to include the

resource identifier, which is the URI. The response from the server must include sufficient information to allow the client modify the resource. Hence, request and responses from the server and the clients must contain sufficient information for the server to handle the request and the client to understand the response. The response from the server should also contain hypermedia, which is just more information or hyperlinks that the client can use to change the state of the web application. These constraints have been set up to ensure that the request from different client look the same whether they are running a python script, android app or a Linux server.

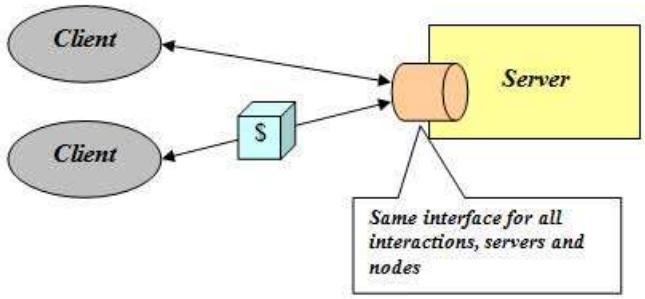


Figure 2.4: Uniform-Client-Cache-Stateless-Server

Another constraint specified by REST as Massé (2012) explains is client and server separation as shown in Figure 2.5; which simply means that the client and server act independently and both of them can operate without the other. Another constraint that RESTful services advocate for is statelessness; Figure 2.6 illustrates it. To effect this, the server should not be able to remember requests sent to it from the client or a user of the API.

Figure 2.5: Separation of concerns

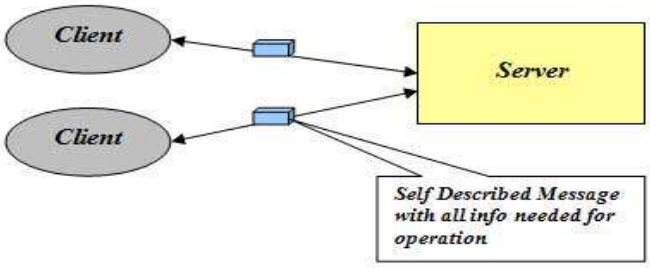


Figure 2.6: Stateless

REST architecture also requires the client should be agnostic to the layers of servers and their functionality between the client and the servers (Massé, 2012). This means that despite how many interconnected servers between the state of the resource presented and the client, their configurations, security implementation or communication should not affect the request or the response. Refer to Figure 2.7.

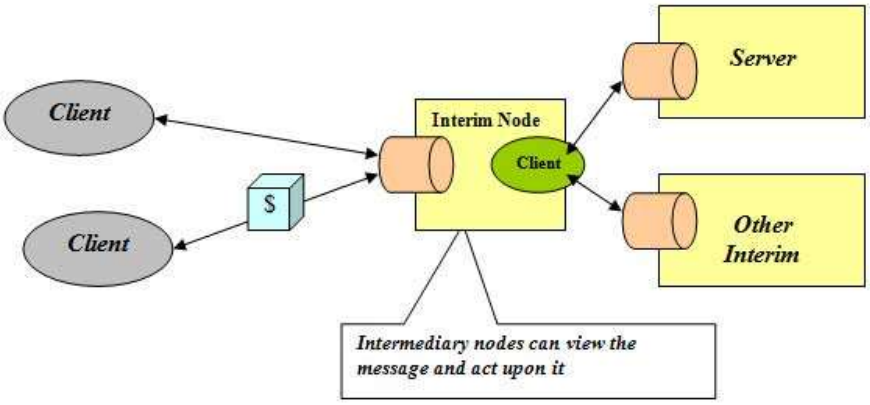


Figure 2.7: Layered System

Specifying the Cacheability of data sent from the server is a function that REST architecture must do. This means, sending notifications on whether the data is cacheable or not when a client gets a response from the server is mandatory. If the data is cacheable, versioning using version numbers follows to maintain integrity of the data and allow updates and expiry or server responses. Figure 2.8 illustrates this.

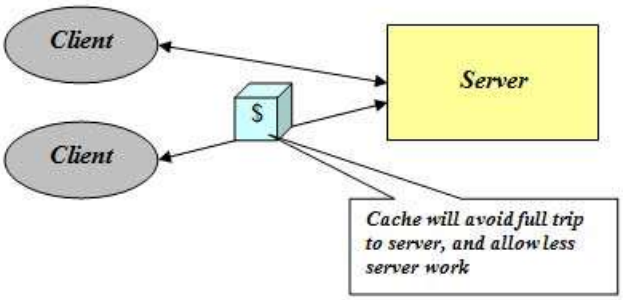


Figure 2.8: Cacheable

An optional constrain in REST architecture is client can request code on demand from the server as illustrated in Figure 2.9. The response from the server is be presented in form a script which the client can execute (Massé, 2012).

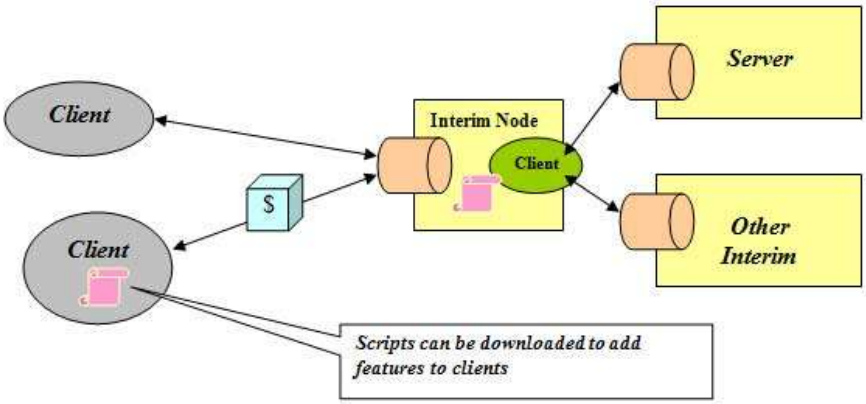


Figure 2.9: Code on Demand

2.8 REST API and Web Services

A web service as defined by (Massé, 2012) is a collection of open protocols and standards used to exchange data between systems programmed in different languages. Most of these systems utilize web services built on different platforms. Through web services, they are able to communicate and exchange data over the internet.

REST as a web service, it emulates HTTP and similar protocols and constraints them to a well-defined set of generic operations. It utilizes less bandwidth and resources as well. REST web service is language and platform independent that makes use of any programming

language and executes in any platform. REST API architecture has the ability to handle multiple types of calls, return different data formats and modification primarily with the correct application of hypermedia. Moreover, it is outstanding for its incredible layer of tractability (Fielding, 2000).

According to the REST design described by (Fielding, 2000) accessing these web resources happens from a common interface using the HTTP protocol and its standard methods of GET, PUSH, DELETE, POST, OPTIONS. A RESTful web service usually defines a Uniform Resource Identifier (URI) that provides resource representation and points exactly at the resource a RESTful service should manipulate. Whenever we represent an API in a RESTful format, it presents the states of its resources for manipulation. The representation of the state can be in JSON, XML or HTML format (Kukan, 2019). The Figure 2.10 illustrates the key concepts in a RESTful API.

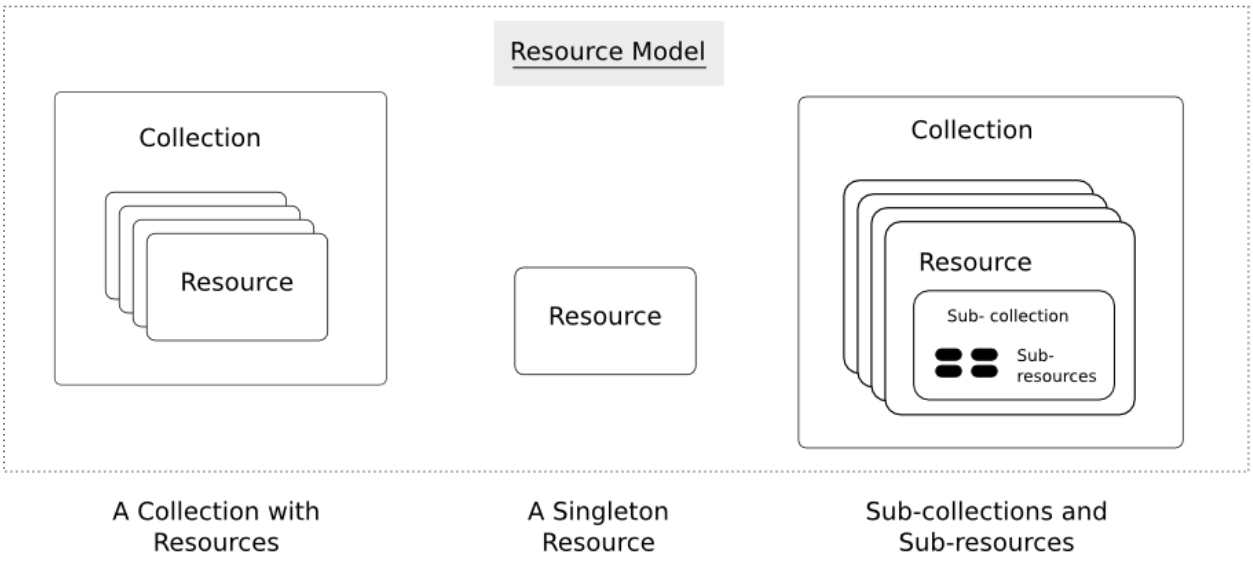


Figure 2.10: Key Concepts in RESTful API

2.9 Conceptual Frame Work

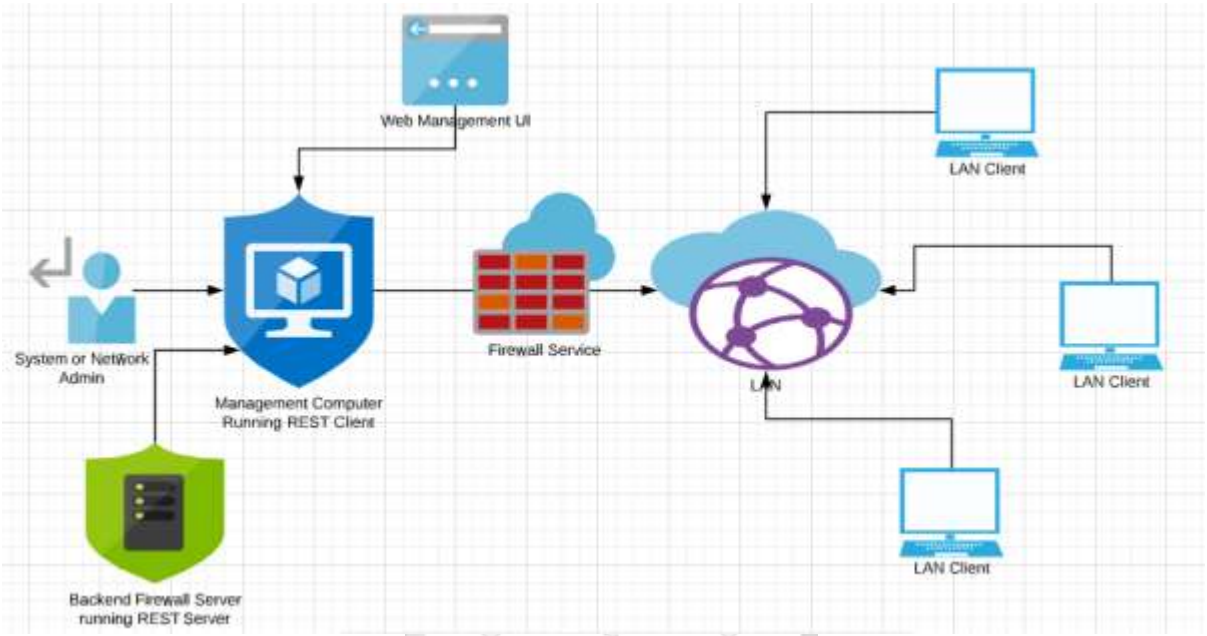


Figure 2.11: Conceptual Framework

As shown in Figure 2.11, a system administrator has a management web interface where they will issue REST API requests to manage the firewall. The web interface is the system admins point of interaction with the system. The REST API receives requests by the REST client and the server interprets them. The REST services then parses the requests from the user using the web client through the REST client to the firewall module. The backend firewall service implements the specified actions. This backend firewall service runs the firewall module. The system returns a response to the user based on the effect the request had on the firewall. Users within the protected LAN validate the rules invoked by the REST API web service.

Chapter 3 : Research Methodology

3.1 Introduction

As noted by Kothari (2004), research methodology concerns the various steps adopted by a researcher in studying the research problem along with the logic behind them. This chapter looks at the qualitative and quantitative research approaches adopted by this work. A discussion on how primary data and secondary data collection was done is covered in this chapter. The chapter discusses the adoption of LAN as the location of the study. The chapter covers the target population and sampling technique used to set up the environment for this research. Further, a discussion on how to present and analyze the test results on the web interface used to configure the firewall follows. The system development methodology, agile and test driven, forms the next issues of discussion. This chapter looks at how to implement them in this research. Finally, the chapter concludes by discussing the research quality and ethical considerations that this work put into account.

3.2 Research Design

According to Kothari (2004), research design is the arrangement of conditions for collection and analysis of data in a manner that aims to combine relevance to the research purpose with low cost in procedure. Thus, this research leveraged a lot on qualitative and quantitative approach for it to succeed. The qualitative approach this research took is when the researcher explored under-the-hood functionality behind the invocation different firewall rules. This information is important to give the user an easy guide on how to invoke the commands and any additional information around the command. This means that the researcher needed to study whitepapers and software documentations of these commands to understand and implement these firewall services modules and rules to the users in a language they can understand and interact with it.

A quantitative approach taken by this research involves collecting primary data during the tests conducted in both test-driven methodologies and agile methodologies proposed in section 3.8. This primary data collected during the scientific experiments served a purpose of making sure the firewall rules worked when invoked. In addition, the same experiments were conducted to ensure that the front facing web interface that the user interacts with is able to

communicate to a firewall module and invoke commands from a web based API to the backend firewall modules and services.

3.3 Location of the Study

The study took place in an emulated environment. This work set up a virtual LAN interfacing with an existing LAN. The virtual LAN created was on a Virtual environment with the host being virtual machines connected through the physical network infrastructure. The firewalls and the management services were set up on the virtual machines. This set up represented the test bed and made up the environment where the researcher will conduct the study.

3.4 Target Population

A research population consists of all individuals or objects of interest to the researcher (Marczyk et al., 2005). The target population for this study was virtual machines and a physical network infrastructure. Inside the virtual machines, this study focused on packet filtering firewall modules. The packet filtering firewall modules that were of interest in this research were iptables rules and commands. This research focused on getting the CLI invocation procedure of these firewall modules on a web interface for easier management and monitoring. The researcher further shrunk the target population by only focusing on commands that could be tested and validated within a LAN.

3.5 Sampling Frame

A sample is simply a subset of the population (Marczyk et al., 2005). The researcher used non-probability sampling technique where samples are based on the subjective judgment of the researcher rather than random selection (Etikan, 2017). Owing to the specific solution that this research developed, the sampling technique of choice was judgment sampling. A judgment sample is based on a non-probabilistic sample space where the units to be sampled are known to the researcher based on information and experience (Kothari, 2004). Based on the information out there on implementing firewall services and experience from other works on deploying firewall services, this work employed the judgment sampling methodology to ensure the selection and usage of correct units as explained in section 3.4.

3.6 Data Collection

Mugenda & Mugenda (2003) asserts that data refers to all the information a researcher gathers for the research. The two categories comprise of Primary data and the secondary data. Primary data refers to the direct information that a researcher obtains from a variable of interests for a particular purpose while secondary data refers to the information obtained from sources that already exist (Sekaran & Bougie, 2016).

The researcher collected primary data during the testing and implementation phase to ensure that the web based interface can invoke a firewall module service with ease. Possibly, with the click of a button. To ensure this is validated, the researcher was required to test the actual services that was specified or invoked from the web interface and confirm if it took effect or not. The researcher obtained data mainly through observation, analysis and comparison to expected experiment outcomes. This work preferred observation, analysis and comparison methods of collecting data because the experiments conducted presented the researcher with relevant information for them to synthesize. It is only through comparing and analyzing that this work was be able to validate whether the solution proposed works or not.

3.7 Data Analysis

Based on the nature this research took, the qualitative and quantitative approach, presentation and analysis of the data adopted descriptive analysis. This work used a web interface to manage backend firewall services. However, for validation and presentation of the outcome, terminal screens described the background service calls and processes. In addition, descriptive analysis demonstrated how this research met the objectives during the system development and testing phase. This research used images and accompanying explanations and descriptions to analyze the data collected by the researcher. This work goes ahead to describe the images by analyzing the observations.

3.8 System Development Methodology

This research combined two system development methodologies to implement the intended solution. Agile and test-driven approach were the methodologies this research explored. A test driven methodology according (Janert, 2004) allows creation of failing tests first before the actual working solution. This approach was suitable to deliver the proposed solution. Coming up with a web-based tool that uses REST API to manage firewalls requires

a functioning firewall management procedure to interface with the web management portal. The firewall application used is an Ubuntu based Linux firewalls. The researcher has to write, test and prove the commands are working before the system implements and abstracts them from the users.

Every commands that this research presents to the user on the web interface must pass the test. For example, if the researcher wants the user to block a particular port on the firewall from the web interface, they must test and try the commands that invokes the blocking of ports functionality. The researcher must improve the failing tests before a front end for the command is developed. For *n* commands that this research was able to implement, the same approach applies.

In addition, a test driven methodology provides room for the developer to come back and make amendment to their solutions or enhance their solution whenever they saw the need in the testing stage. Figure 3.1 illustrates the processes that are involved in a test driven methodology that this research explored.

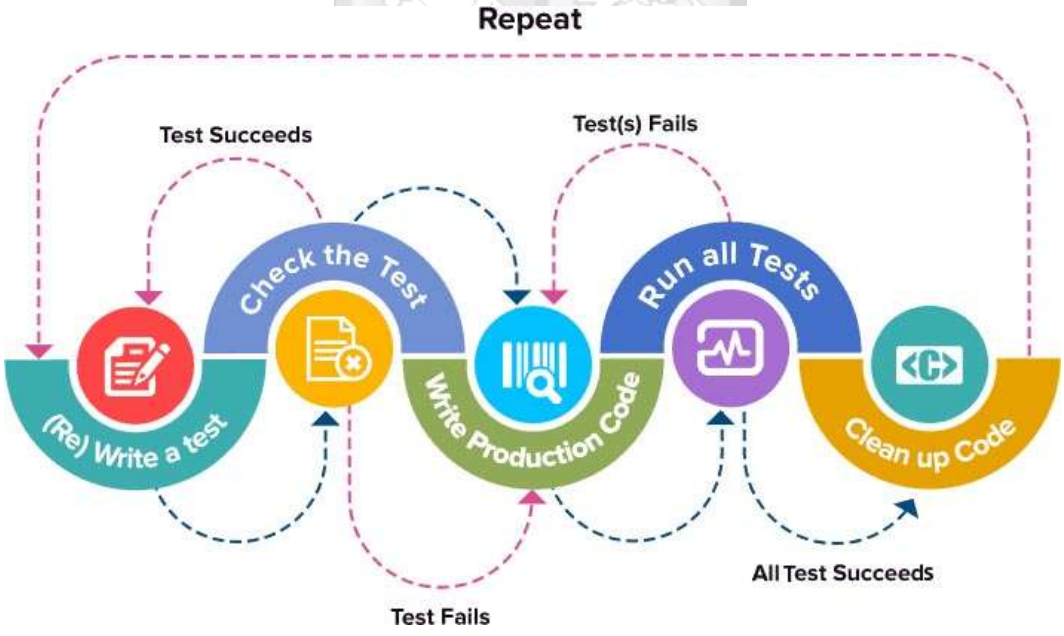


Figure 3.1: Test Driven Methodology Diagram

In conjunction with the test-driven methodology, the researcher chose agile methodology to develop the next module that provides the users with a web interface to manage monitor and orchestrate firewall services. The philosophy of agile is to divide the whole block of work into manageable smaller chunks. The smaller chunks form categories based on their

similarities to make up a feature. A minimum viable product then comes from these features. The customer receives the minimum this minimum viable product (Akins, 2003).

In the case of this research work, individual work, the researcher played the role of the user who will provide the user stories used to come up with user acceptance tests when ticking the delivery boxes. Within agile as a framework for software development, this research resort to extreme programming which is a methodology that spans out of agile and what agile advocates for.

The use of this methodology came up with the different components that made up the web interface used to manage, monitor and orchestrate firewall services. This research anticipated that the different firewall rules that the users want to invoke made up the user stories. The user stories specified the sprint and gave the sprint a period for development depending on the simplicity and complexity of the rules. The researcher deliberated this during stand up meetings that they held with any interested party to this research. Figure 3.2 shows an extreme programming method, which contextualizes the process described in the paragraph above.

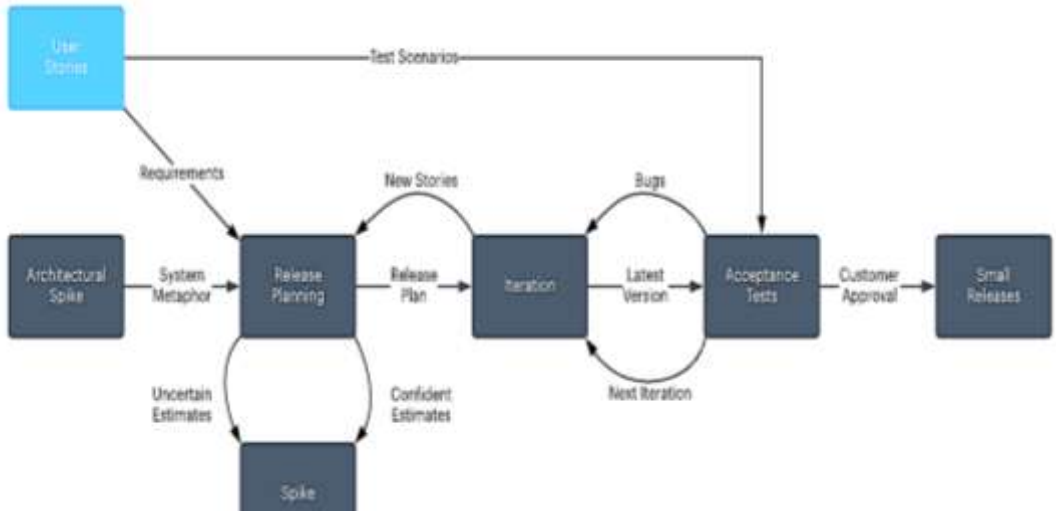


Figure 3.2: Extreme Programming Methodology

3.8. 1 System Implementation

This research developed a Representational State Transfer (REST) Web Tool for Firewall Services Management and Monitoring in a LAN. To bring this to life, this work implemented the solution using the following tools.

Virtual Box was the hypervisor used to abstract the virtual hardware resources to enable the researcher create virtual machines. The virtual machines implemented a host that had the REST architecture server and REST client software. In addition, this virtual machine equally deployed the packet filtering firewall services, Iptables.

The virtual machine, both REST client and server was based on Ubuntu Linux 18.04. The virtual machine used to run the firewall service run on Ubuntu Linux 18.04, server version. The virtual machine where the web interface was set run Ubuntu Linux 18.04 server version. In this case, the researcher installed Apache as the web server to host the web interface. The researcher wrote the web REST APIs in PHP. HTML and CSS helped the researcher to design the interfaces.

3.9 Research Quality

This research aimed to build on top of current working firewall modules. This research did not re-invent the wheel on how to write firewall modules. In addition, this research complemented current working firewall modules and services through interfacing them with an easy to use web services based on REST APIs. This research ensured that replication and adoption of this work is easy through thorough analysis of different implementation and integration strategies. This research delivered a precise and easy to understand documentation of the proposed solution to assist in the adoption of the work.

3.10 Ethical Consideration

Since this research worked with firewall services on the LAN, the researcher ensured that during the development, tests and implementation phases, service interruption did not occur. The researcher ensured that the development environment is isolated and did not disrupt traffic or any services by other users in the network. In an instance when the researcher needed the participation of other LAN users in the network, permission was asked, an explanation

provided to them and they were allowed to make a choice on whether to participate in the experiment or not.

Chapter 4 : System Analysis and Design

4.1 Introduction

As stated by Faisandier (2013), System analysis and design is the process of defining the description of the system design, architecture, components and interface in order to match the user's specified requirements. System analysis involves the collection and analysis of the user specified requirements and translating them into logical and conceptual models. System design is the process of describing the architecture, components, data and interfaces for a system to satisfy indicated requirements (Dennis et al, 2015). The aim of this chapter is to analyze the design procedure applied to the solution presented by this research. In addition, the chapter presents the software analysis and modeling diagram that brought the solution to life. Requirement gathering, both functional and non-functional requirements starts the discussion in this chapter. The chapter then presents the diagrammatic representation of how the system will work using a general system architecture diagram, a use case diagram and a sequence diagram.

4.2 Requirement Analysis

This section looks at the various requirements this work has to meet in order to achieve the design and implementation of a simple to use web tool and API to manage firewall services, as envisioned by the objectives of this research.

4.2.1 Functional Requirements

i. The web tool should be able to ensure connectivity between the web server and the firewall service in the system
ii. The web tool should be able to provide an easy to use HTML web page with forms to allow a configuration interface
iii. The web tool should be able to interpret values and variable passed by the user to system commands to configure the firewall
iv. The commands, through the web tool should successfully configure the firewall service to either block, allow or reject services based on addresses or applications

4.2.2 Non Functional Requirements

The proposed solution provides a web tool for managing firewall services. To achieve this, the web tool should be platform independent. As long as the system administrators are able to get access to a device with a web browser, be within the subnet of the server running the firewall and web services, they should be able to access the web page and do management and configurations.

4.3 System Architecture

Figure 4.1 outlines in general how the solution will be set up and work. The entry point to managing and configuring the firewall is the web page. It is the central point in the system where data passes to and from the firewall application. From the web page, users will send form data or HTML requests to the web server. The web server passes them over to multiple PHP API scripts for processing based on the requests. The PHP scripts validate the data and use it to manage or configure the iptables firewall service by running BASH commands within the scripts. The PHP API scripts hold the results and send them back to the web server. Finally, the web server sends the results to the web page for presentation.

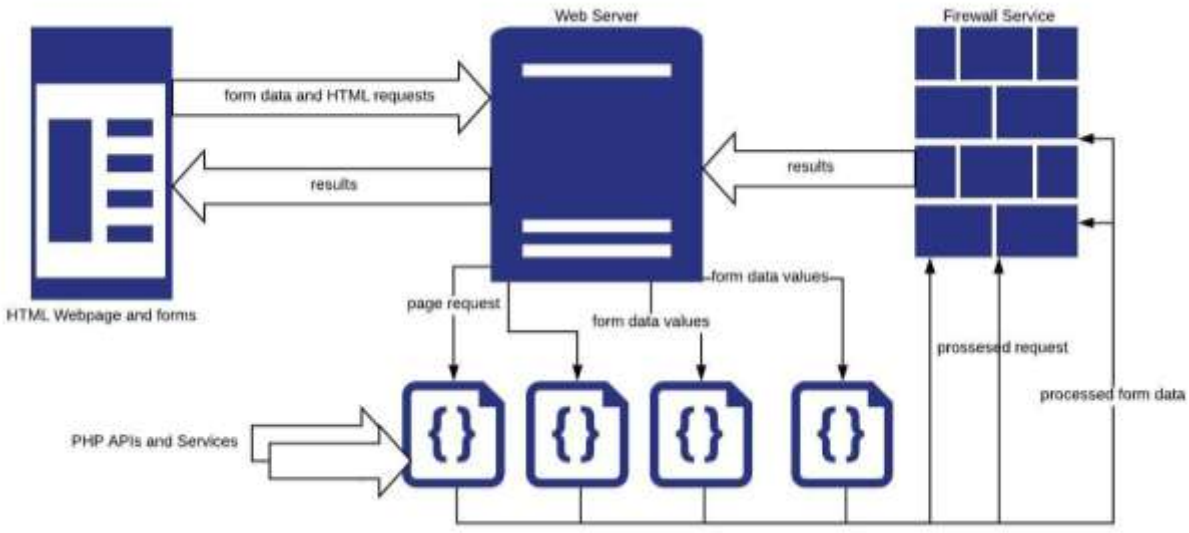


Figure 4.1 General System Architecture

4.4 System Design

4.3.1 Use Case Diagram

The Use case diagram shown in Figure 4.2 depicts four potential entities that interact with the system. The first user is a system administrator who wants to manage and configure the firewall via the web page. The user is responsible for filling in the web forms in order to configure the firewall and check or manage the firewall status. In the second use case scenario, check firewall status, the user performs basic status checking of the firewall like displaying existing rules, deleting them and saving configurations. The user can only perform these actions once they have logged in to the system using a set of valid credentials.

The web form is responsible for receiving user requests and form data. This PHP APIs receives this data. Each of the web forms work with specific PHP scripts used to receive their requests. Additionally, the web forms are dynamic and provide and entry point into system management and configuration.

The PHP APIs are the core of achieving the main job of the solution. These scripts work in conjunction with the data presented from the web forms. First, the scripts validate the data from the web form. Validation of the data involves checking the range of port numbers and IP addresses. Once this process is complete, the scripts will then make sure a working BASH command is executed and a response is held awaiting transfer to the web page.

Finally, the firewall service presents another scenario where the actual configuration of the firewall application happens. The firewall service, iptables receives the BASH commands passed by the PHP scripts and act on them. The firewall service is responsible for actual configuration and generating responses pushed back to the PHP scripts.

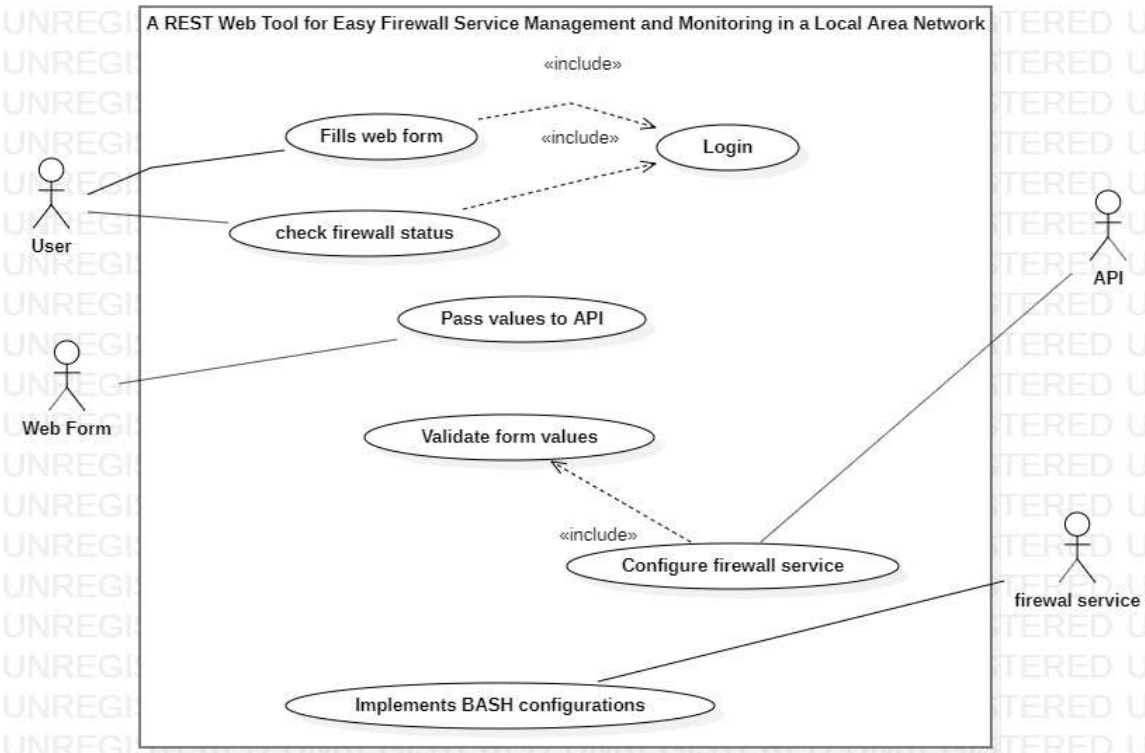


Figure 4.2: Use Case Diagram

4.3.2 Sequence Diagram

The sequence diagram shown in Figure 4.3 presents the possible order of events from the first action to the last action when interacting with the system. The user firsts login the system. The assumption made in this case is that user accounts exist directly in the login scripts. A successful login returns the homepage to the user with different menu items they can use to interact with the system.

The user then proceeds to firewall configurations where they fill in the web forms. The web forms are either web forms that allow the user to configure the firewall based on IP addresses or based on port numbers. From the forms, the user will choose an action, the action are buttons that either allow or reject connections based on port numbers or IP addresses.

This web page and forms capture the web data together with the specific action. The specific action guides the system on which PHP script to pass this data. There are scripts written for each specific task. The task could be to block a port number or allow a port number. The script specified in the action will receive the values captured from the forms.

The next step is to validate the values passed to the API by the forms. The validation is different for each set of values. For instance, when working with port numbers, the scripts handling port numbers will check if the port numbers are valid numbers and are within range. The same thing happens to scripts handling IP addresses. If the values are correct, processing of the values begins.

Processing involves running the BASH commands. The BASH commands are processed and passed to the firewall service for implementation and actual configuration of the firewall service happens. Lastly, the user can query the firewall service to check the status of the firewall by passing that requests to the API through buttons on the web forms. The APIs query the firewall; the firewall responds with appropriate information e.g. the existing rules. The web page presents the results to the user.

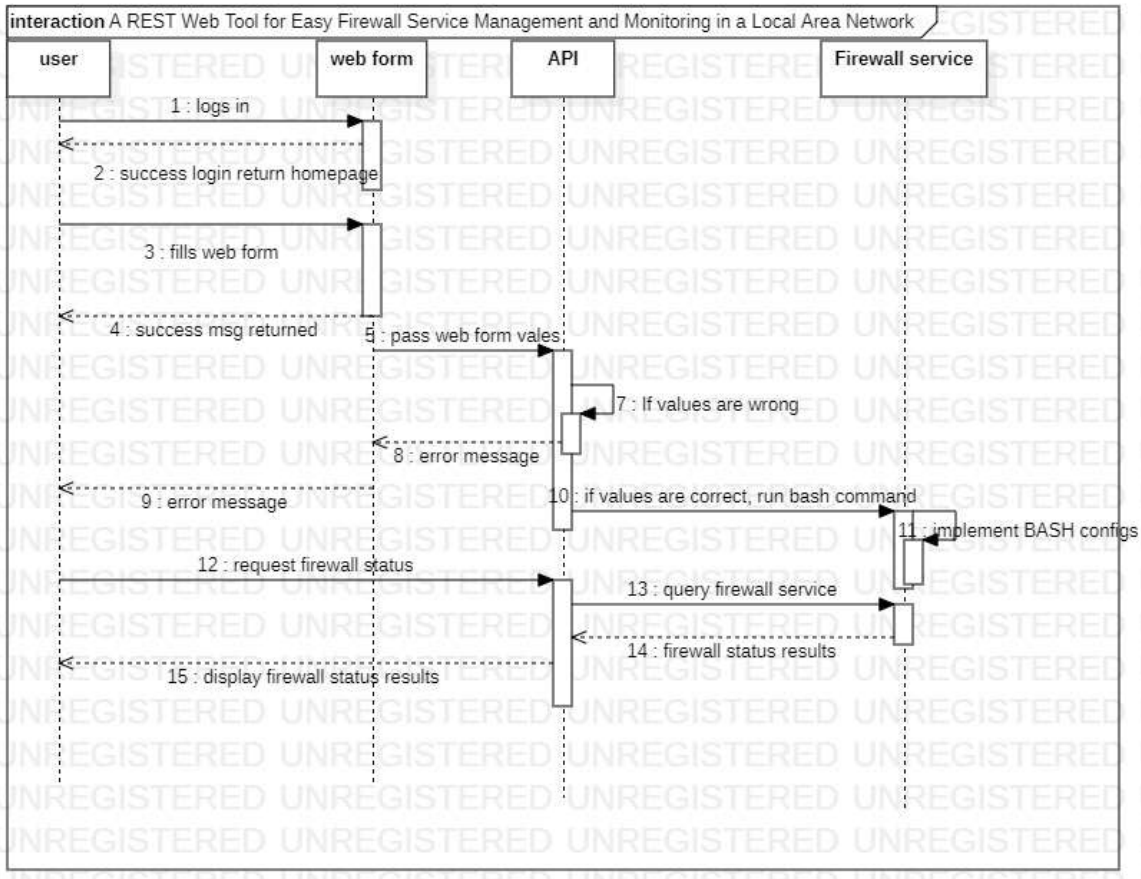


Figure 4.3: Sequence Diagram

Chapter 5 : System Implementation and Testing

5.1 Introduction

Proper implementation follows the design principles outlined in the previous chapter 4. The model components used to implement the web tool kick off the discussion in this chapter. When discussing model components, the chapter looks at the software and hardware components pieced together in a very specialized test bed environment that includes a web server, and a firewall system. This section presents the process of implementing the system that includes creating the web page and GUI of the system. This chapter also looks at and analyzes the PHP APIs that interact with the web forms. This chapter then proceeds to test the application. Tests conducted include firewall management, basic firewall operations like disable, save and firewall status. A discussion on advanced tests that handle firewall configurations actions based on addresses follows. Finally, this chapter presents a system testing class table to validate the working of the functional and non-functional requirements

5.2 Model Components

The test conducted to achieve the objectives of this research influenced the software and hardware components needed.

1. Hardware components; A Personal Computer

   i. 4 Gigabyte (GB) Random Access Memory (RAM)

   ii. 500GB hard drive disk storage

   iii. Intel core i5-443OS at 2.7 GHz of processing speed with DirectX 11 graphical capabilities.

2. Software Components

   i. Windows 10 Host Operating System

   ii. VirtualBox Hypervisor

   iii. Ubuntu Server 18.04

   iv. Putty terminal emulator, serial console and network file transfer application

5.3 Test bed set up

To implement this experiment, the researcher had to ensure the environment met certain bare minimums. First, the virtual machine running on the host OS had to be running the latest version on iptables packet filtering firewall. Figure 5.1 verifies the running of iptables.

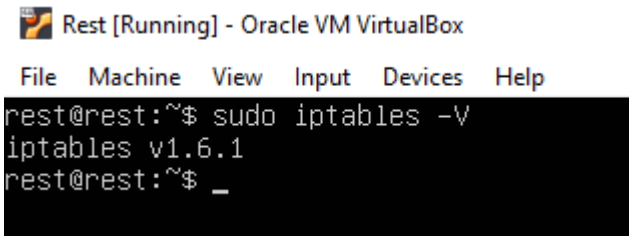


Figure 5.1: Verification of a running iptables

In addition, the researcher had to install and set up a working web server. This work opted to set up Apache server on the same Ubuntu 18.04 where the firewall service was running. Figure 5.2 indicates a working Apache web server.

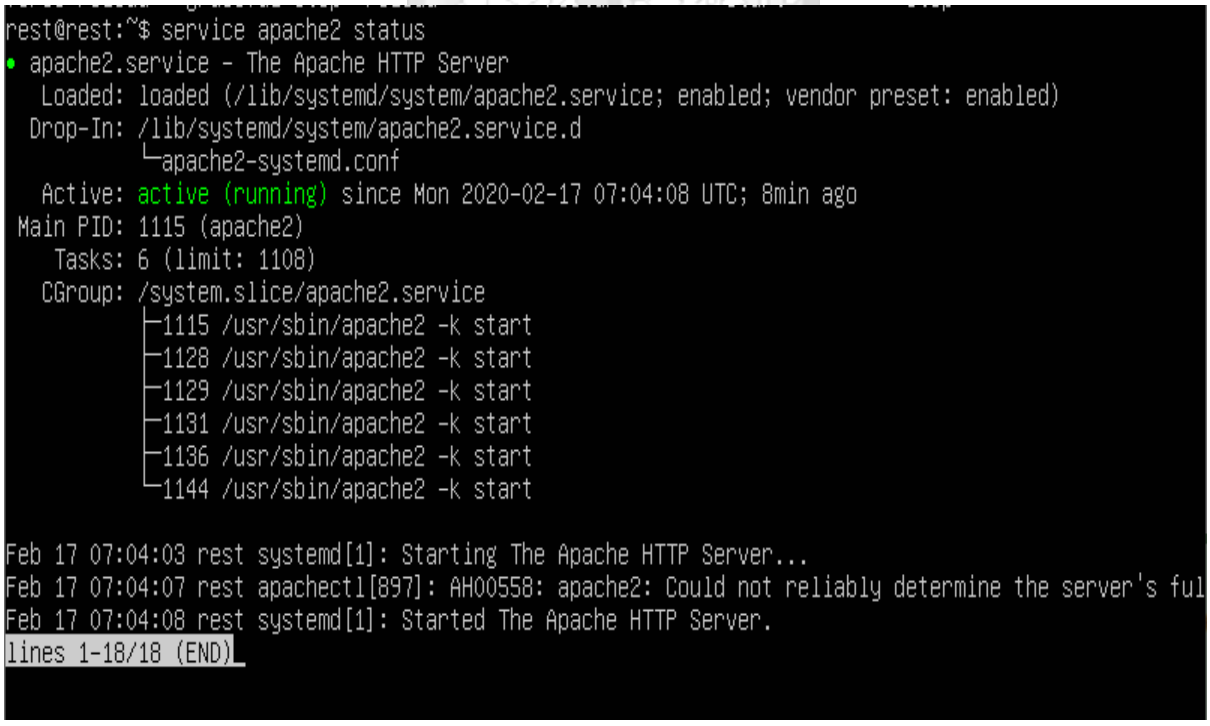


Figure 5.2: Verification of a working Apache web server

Besides the web server, the researcher had to install the latest version of PHP and confirm that it works. Figure 5.3 indicates a correctly installed and working PHP.

Figure 5.3: Verification of correctly installed and working PHP

Finally, for PHP to parse the configurations used to manage the firewall, the research had to add Apache to the sudoers lists to execute administrative commands to invoke and set up Iptables rules. Otherwise, the commands would not work. Figure 5.4 shows Apache added to sudoers list.

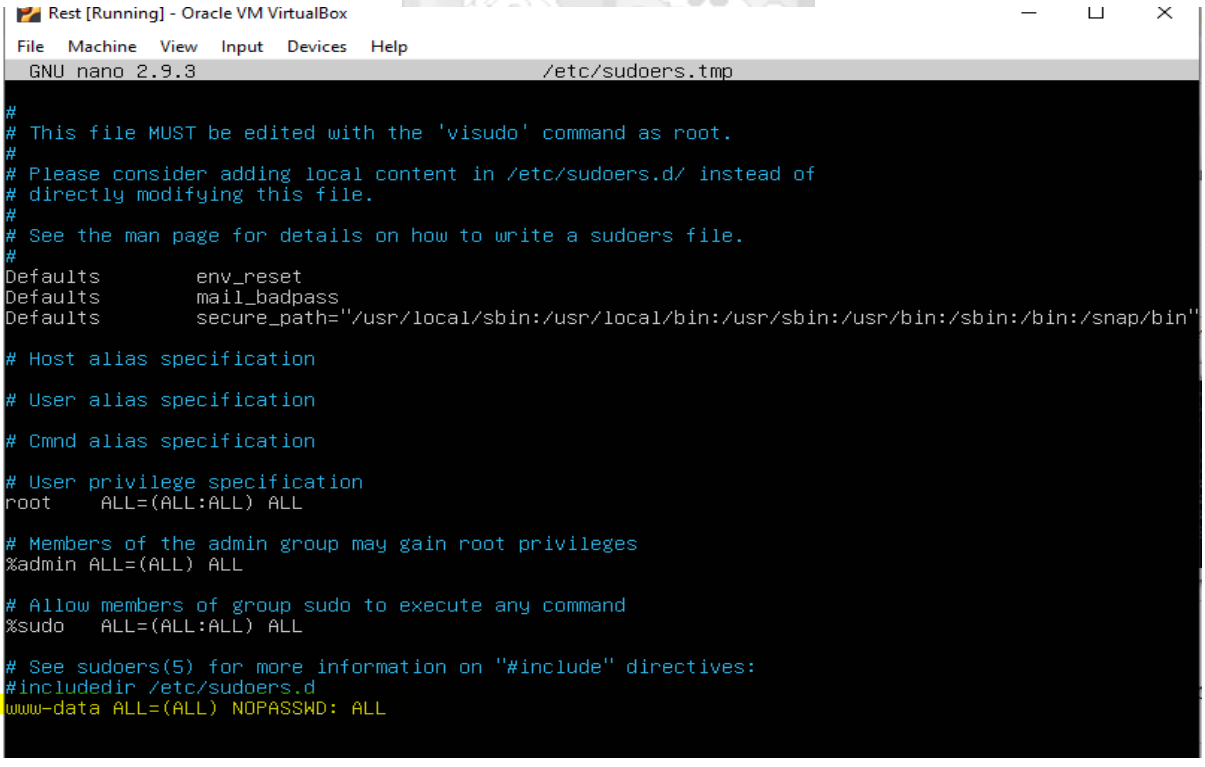


Figure 5.4: Verification of Apache added to the sudoers file

5.4 System Implementation

After test bed was setup and configured correctly as discussed in section 5.3, the system implementation involved two stages. The first stage was creating a web page that has the GUI to manage the firewall using web forms and buttons. The next step was to pick user variables from these forms through a PHP API, validate them, process them and pass it to the firewall service via the API for complete configuration.

5.4.1 Webpage GUI development

This work used HTML to develop the web page supported by CSS and JavaScript to control the button actions. At this step, this research required to pass variables through the POST method and the GET methods. The POST methods were to be implemented using the submit button. However, owing to the nature of how HTML works, using more than one submit button on a single page proved to be tricky. This research opted to use java script functions to capture button actions and send them on a different PHP page for processing. Figure 5.5 illustrates the corresponding JavaScript to work on the button actions.

```
<div class="panel panel-widget forms-panel">
    <div class="forms">
        <div class="form-two widget-shadow">
            <div class="form-title">
                <h4>Customize based on Port Numbers:</h4>
            </div>
            <div class="form-body" data-example-id="simple-form-inline">
                <form class="form-inline" action="#" method="post" form id="block_basic_ip">
                    <div class="form-group">
                        <label for="exampleInputName2">Port Number</label> <input type="number" name="s-port" class="
                        form-control" id="exampleInputName2" placeholder="Src IP"> </div>
                        <button type="button" onclick="submitForm('block_basic_port.php')"  class="btn btn-default
                        w3ls-button" >Block </button>
                        <button type="button" onclick="submitForm('allow_basic_port.php')"  class="btn btn-default
                        w3ls-button" > Allow </button>
                    </form>
                </div>
            </div>
        </div>
    </div>
```

Figure 5.5: JavaScript working on button actions

5.4.2 PHP APIs to Manage Firewall Service

The values picked from the webpages in the system could invoke either a POST or a GET method. For example, the image that shows the button status could run a GET method

and query the firewall to printout the list of existing rules that are running in the background. The index.php file implemented this function is shown in Figure 5.6.

```
<div class="table-heading">
        <h2>Firewall Status | Existing Rules </h2>
        <?php
                $output=shell_exec('sudo iptables -L -n');
               echo "<pre>".$output. "</pre>";

         ?>
</div>
```

Figure 5.6: PHP code running GET method for Firewall Status

To manipulate the firewall using different form data, the researcher was required to get variables from the form and invoke a POST method that will send the data to different PHP files that will first validate the data to check if the IP address is within the valid range. The first conditional if statement does that. If that condition is true, a shell execution function runs and perform specific actions to configure the firewall. In this case, captured by the code in Figure 5.7, the code drops the traffic from the specific IP addresses held in variables s-ip and d-ip.

Other snippets of code that validates correct IP addresses then goes ahead to block and allow them as shown in Figure 5.7 and Figure 5.8 respectively. The PHP files in question that communicated with the HTML forms were ***block_basic_ip.php*** and ***allow_basic_ip.php***.

rest@rest: /var/www/html/firewall

```php
<?php
if( $_POST["s-ip"] || $_POST["d-ip"] ) {

        $src_ip=$_POST["s-ip"];
        $dst_ip=$_POST["d-ip"];
        if (filter_var($src_ip, FILTER_VALIDATE_IP)) {
                echo $src_ip . "is a valid IP address" ."<br/>";
                echo "The src ip to block is ". $src_ip. "<br />";
                $output = shell_exec('sudo iptables -A INPUT -s '.escapeshellarg($src_ip).' -j DROP');
                echo "<br/>". $output;
        } else {
                echo $dst_ip. ": The Source IP address is not a valid IP address or you did not enter one";
        }

        if (filter_var($dst_ip, FILTER_VALIDATE_IP)) {
                echo $dst_ip. " is a valid IP address"."<br/>";
                echo "The dst ip to block is ". $dst_ip. "<br />";
                $output = shell_exec('sudo iptables -A OUTPUT -d '.escapeshellarg($dst_ip).' -j DROP');
        } else {
                echo $dst_ip. ": The destination IP address is not a valid IP address or you did not enter one";
        }

     exit();
   }
?>
~
```

Figure 5.7: PHP code to block IP addresses

rest@rest: /var/www/html/firewall

```php
<?php
if( $_POST["s-ip"] || $_POST["d-ip"] ) {

        $src_ip=$_POST["s-ip"];
        $dst_ip=$_POST["d-ip"];
        if (filter_var($src_ip, FILTER_VALIDATE_IP)) {
                echo $src_ip . "is a valid IP address" ."<br/>";
                echo "The src ip to allow is ". $src_ip. "<br />";
                $output = shell_exec('sudo iptables -A INPUT -s '.escapeshellarg($src_ip).' -j ACCEPT');
                echo "<br/>". $output;
        } else {
                echo $dst_ip. ": The Source IP address is not a valid IP address or you did not enter one";
        }

        if (filter_var($dst_ip, FILTER_VALIDATE_IP)) {
                echo $dst_ip. " is a valid IP address"."<br/>";
                echo "The dst ip to allow is ". $dst_ip. "<br />";
                $output = shell_exec('sudo iptables -A OUTPUT -d '.escapeshellarg($dst_ip).' -j ACCEPT');
        } else {
                echo $dst_ip. ": The destination IP address is not a valid IP address or you did not enter one";
        }

     exit();
   }
?>
~
```

Figure 5.8: PHP Code to Allow IP Addresses

This work took the same procedure to develop PHP files that takes form data to manipulate port numbers configuration. More information on this is found on the Appendices A and B.

5.5 System Testing

This research conducted an experiment to present a simple web tool that used by system administrators to manage and configure firewalls. HTML forms that capture user data presents this. This user data is IP addresses and port numbers. The system administrator passes these variables down to an Iptables firewall system running in the virtual machine to configure the firewall. A PHP script helps in processing, validating and passing these commands from HTML form down to the Iptables system.

The application used in testing the developed solution is virtual local area network and bridge application. This application allows the user to configure groups of ports, independent of their physical location as a virtual LAN. The arrangement is a logical grouping and the logical group known as a virtual LAN. The next section of this work demonstrates the test results this research conducted.

5.5.1 Firewall management test results

This screen in Figure 5.9 presents the different buttons the user has and can use to manage the firewall. The status button displays the existing rules that have been configured in the firewall. The save button saves the configurations the system administrator makes on the firewall using iptables. This is because by default, iptables clears all configurations when the system restarts. The disable button clears the firewall, deletes all the rules and deactivates the firewall.

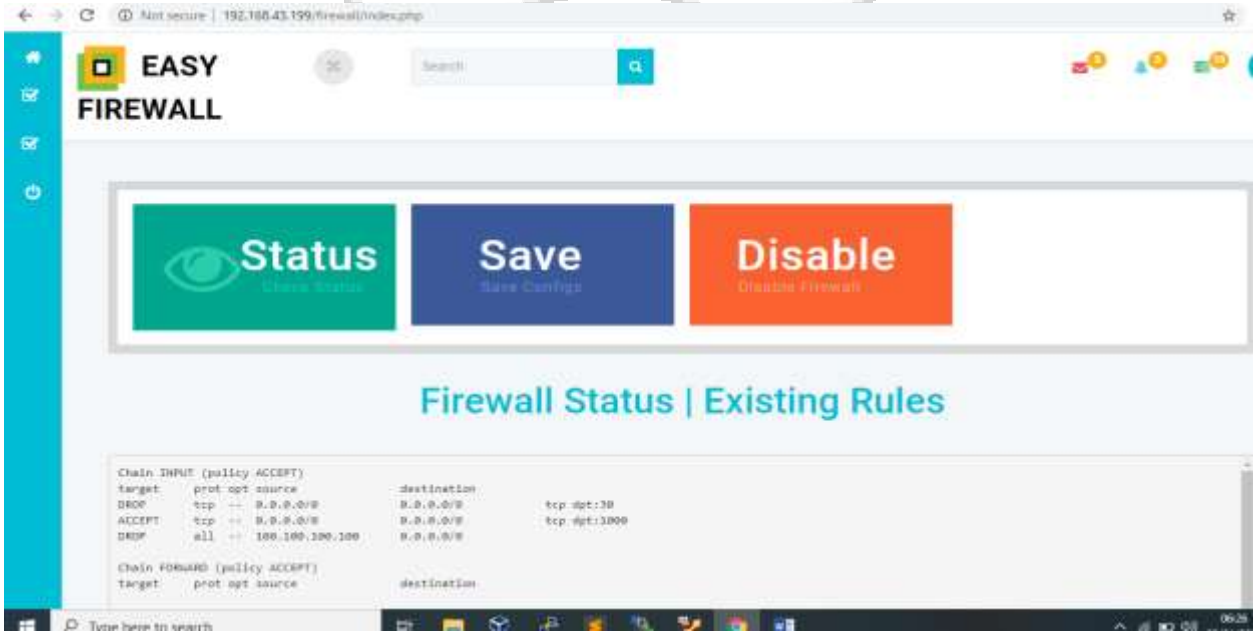


Figure 5.9: Status, save and disable options

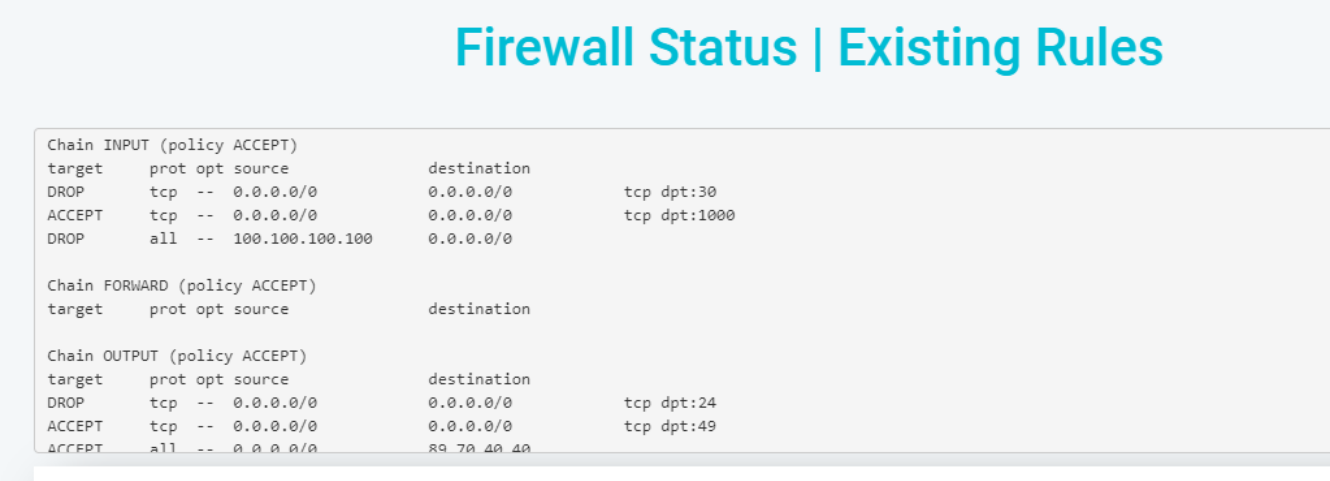


Figure 5.10: Firewall status and rules results

This screen in Figure 5.10 presents the status of the firewall. It shows the existing rules that have been configured on the firewall. Without the implementation of this research, the user or the adminstrator was supposed to type the command shown in the Figure 5.11 to get a similar output on the terminal.

```
root@rest:/var/www/html/firewall# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:30
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:1000
DROP       all  --  100.100.100.100      0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:24
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:49
ACCEPT     all  --  0.0.0.0/0            89.70.40.40
root@rest:/var/www/html/firewall#
```

Figure 5.11: Command and output to display status

In addition, in order for the user to save the configurations, they will have to type in the command shown in Figure 5.12. However, in this solution, using the button as described above,

clicking the save button allows the system to save the current configurations in a text files called save.txt. Figure 5.12 shows the output.

```
root@rest:/var/www/html/firewall/configs# cat save.txt
# Generated by iptables-save v1.6.1 on Thu Jan 23 03:38:14 2020
*filter
:INPUT ACCEPT [1317:102754]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [997:149811]
-A INPUT -p tcp -m tcp --dport 30 -j DROP
-A INPUT -p tcp -m tcp --dport 1000 -j ACCEPT
-A INPUT -s 100.100.100.100/32 -j DROP
-A INPUT -s 15.15.15.51/32 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 24 -j DROP
-A OUTPUT -p tcp -m tcp --dport 49 -j ACCEPT
-A OUTPUT -d 89.70.40.40/32 -j ACCEPT
COMMIT
# Completed on Thu Jan 23 03:38:14 2020
```

Figure 5.12: Command and output to save configurations

When the user clicks the disable button, Figure 5.13 shows the command that runs in the background. This command clears all the rules and disables the firewall.

```
root@rest:/var/www/html/firewall# iptables -F
```

Figure 5.13: Disable command

5.5.2 Firewall configuration test results

Configuration of the firewall is done using port numbers as shown in Appendix E. The incoming and outgoing port can be specified. However, in this solution, we allow the user to specify either of the ports or all. By clicking the block or allow buttons, the firewall is configured to drop or accept the connection on the specified port number respectively. Alternatively, for the user to implement this on the command line, the commands shown in Figure 5.14 will be run for each action of blocking for every source or destination port.

```
root@rest: /var/www/html/firewall
root@rest:/var/www/html/firewall# iptables -A INPUT -p tcp --dport 25 -j ACCEPT
```

Figure 5.14: destination port number accept

Figure 5.15 illustrates the action that will be run for each action of allowing for every source or destination port.

```
root@rest:/var/www/html/firewall# iptables -A OUTPUT -p tcp --dport 25 -j ALLOW
```

Figure 5.15: destination port number drop

Configuration of the firewall is done using ip addressed as shown in Appendix F. The source and destination IP addresses can be specified. However, in our solution, we allow the user to specify either of the IP addresses or all. By clicking the block or allow buttons, the firewall is configure to drop or accept the connection on the specified IP addresses ber respectively.Alternatively, for the user to implement this on the command line, the commands shown in Figure 5.16 and Figure 5.17 will be run for each action of blocking or allowing and for every source or destination ip address respectively.

```
root@rest:/var/www/html/firewall# iptables -A INPUT  -s 15.15.15.51 -j DROP
```

Figure 5.16: Src IP drop

```
root@rest:/var/www/html/firewall# iptables -A INPUT  -s 15.15.15.51 -j ACCEPT
```

Figure 5.17: Src IP accept

The status page presents configurations made either based on IP addresses or port numbers. This research has allowed a one click configuration mechanism and management solution as describe in the sections 5.5.1 and 5.5.2.

5.5.3 System Testing Classes

Table 5.1: System Testing Classes

Test Class	**Inspection Check**	**Priority**
Functional	Is there Connection and exchange of data between the web server and the firewall system?	High
Functional	Does the Web tool provide an easy to use web page with web forms and associated buttons?	High
Functional	Does the web tool get values from the user, interpret these values and variables and pass the commands to the firewall for configuration?	High
Functional	Do the commands, through the web tool successfully configure the firewall service to either block, allow or reject services based on addresses or applications?	High
Non Functional	Does the system provide an easy to use platform independent web page to do system configurations and management?	High

5.5.4 System Testing Results

Table 5.2 System Testing Results

Test Class	Test Results	Comment
Functional	PASS	The web server was able to receive and send data to and from the firewall application
Functional	PASS	The HTML web page had appropriate forms that allowed users to manipulate the firewall either by using IP addresses or Port numbers. In addition, the web page also had buttons for easy firewall management.
Functional	PASS	The web tool, through the HTML web forms and PHP API scripts were able to get values from the user, validate them, process them to appropriate BASH commands and send the down to the firewall for configuration to happen.
Functional	PASS	The BASH commands were interpreted, processed and passed down to the firewall. The firewall system was able to interpret the iptables commands and do configurations on the firewall based on specific actions and values.
Non Functional	PASS	The system was set up to run on any browser or any device that has a browser. The system is simple enough with very direct and precise actions specified for the user to manipulate the firewall service.

Chapter 6 : Discussion

6.1 Introduction

This work has presented a solution that provides an easy to manage firewall service using a web portal and APIs written in PHP. In this chapter, this work sheds more light into the inner working of the solution. The results discussed in this chapter range from why the solution is important and what makes the solution easy. This chapter discusses the nexus between the front-end HTML page and the backend PHP scripts. In detail, this chapter describes the information passed these pages when the user feeds in firewall configuration requests and a result returns. In addition, this chapter compares the results of our solution to already existing solutions in the market like CISCO ASA firewall. Finally, the researcher concludes by highlighting the challenges the researcher faced during the study.

6.2 Discussion of Results

This work has provided an easy to use web interface that manages iptables Linux firewall service. The web server firewall service all sit on the same machine, a virtual machine. This work created a HTML page that interfaces with PHP on the server side to run and execture BASH commands on the lunix box that hosts the firewall.

This solution presents a very representive and easy to use interfaces that explain and guide the user on firewall management. A task that was previously tedious because of the complex commands that one has to memorise and execute. Instead, the solution provided abstracts of all that complexity and provides a user friendly interface. Besides, what differenitates this solution from others is that it is light in nature and the web interface can easily be accessed and managed from any device that has a web browser. In our case, the solution was tested on a mobile phone's browser and worked seamlessly.

Comparing this solution to the Cisco ASA firewall described in chapter 2, the web server of choice was Aware web server (Kukan, 2019). However, to implement the solution documented by this work, Apache was the readily available web server. It easy to configure and easy to support owing to the large community online.

From the results in chapter 5, it is evident that simplicity was achieved. The web forms allowed user to interact with the system by either entering an IP address or a port number. Then

they would specify an action that they needed to be done based on the values they entered. These forms work hand in hand with API scripts written in PHP in the backend that had access to the server running iptables. Appendix G presents the simplified webpage

To compare the solution presented by this research, focus is on the simplicity configuration as shown in Appendix G, Appendix H shows the confguration pane deployed by cisco ASA firewall. From Appendix H, the web page presents the user with a lot of information. Most of it might not be useful in achieving the tasks the user is set to achieve.

Appendix H confirms the need for a simple to use configuration pane that Appendix G shows. This is very ideal when implementing firewalls for small organizations that only operate within their LAN with basic internet connectivity services.

At the same time, these results did a comparison of the tedious effort a system administrator will have to put in to achieve the same results they would have if they used the web tool implemented in chapter 5. The results have equally demonstrated that firewall management can be simplified by bundling up important tasks and actions in an easy to use interface for the system adminstrator. In addition, this research has also demonstrated that no prior knowledge of Linux administration and programming is needed to implement firewalls of a web tool that is easy to use exists.

6.3 Challenges Faced

6.3.1 System configuration challenges

The researcher faced major challenges during the test-bed setup stage. Of course, before programming and integrating the system, a proper working test bed had to be set up. Correct Precise configuration of The Linux, Apache and PHP modules in the virtual machine was important. The researcher then configured Apache to execute sudo administrator commands in the backend without compromising on the security of the system.

6.3.2 Data display and presentation challenges

Since this was an integrated system working with BASH commands, PHP and HTML, passing and presentation of data to and from these languages was a challenge this research had to overcome. First, command output from BASH comes as an array, the researcher had to fetch this data and present it on an HTML page using PHP as a table. This data conversion processes

proved to be a challenge yet very necessary to fulfil the objectives of the system. A solution that is easy and friendly to use. In addition, all of the BASH iptables commands were complex and required extra effort for the researcher to understand the syntax and finally go ahead and implement them in PHP successfully.

Chapter 7 : Conclusion and Recommendation

7.1 Conclusion

The process of configuring firewalls is tedious, complex and constrictive. An easy to use web tool written in HTML has proven to be effective. This research proposed to provide a web tool to manage and configure firewalls easily via the web tool.

This work focused on simplicity in firewall management. This is because enterprise firewalls come with complex and difficult rules to understand. Moreover, most small organizations operating small networks with fewer devices do not need these rules. In addition, as presented in chapter 2, the more the rules a firewall has, the more likely the management and configuration would encounter errors. An opportunity presented itself to abstract the complexity of the firewall management using simple and fewer rules that this work took and provided a solution.

It is important to note that this work did not re-invent the wheel; this work is cognizant of existing firewall modules. The preceding chapters did an analysis of the modules like Iptables and UFW. Iptables is the most applied owing to its robustness and efficiency. However, complexity and the likelihood to making errors when configuring Iptables, especially when doing repetitive tasks, presented another opportunity for this work to exploit.

Already, this work has demonstrated how to exploit the opportunities in chapter 4 and 5. This research has also opened up the barrier to firewall management brought about by complex BASH commands. BASH commands used to manage Iptables are embedded directly into PHP APIs and abstracting them for the user. A simple, easy to use and dynamic web interface helped in achieving this task. The languages this work used were HMTL, CSS, JavaScript and PHP.

Port numbers and IP addresses formed the basic variables, and inputs used by the front-end web page to configure the firewalls. The system then abstracted these and fed them through PHP APIs that would perform two actions, to block or allow a port number or IP addresses. Chapter 5 discusses the PHP APIs in detail.

In addition, the solution implements basic firewall management actions like restarting the firewall, saving configurations, deleting firewall rules and checking the firewall rules.

These actions are simple enough by the use of HTML buttons to run the four actions in the background in a very simple and abstracted manner.

To expound further on the simplicity, this research looked at language use when interacting with the user. Technical language that Iptables comes with was simplified and translated to basic novice technologists' language. Hence, this work has achieved the main objective of developing a web tool for firewall service management and monitoring successfully.

7.2 Recommendation

Numerous work has gone into management of firewalls using web tools before. This research has demonstrated and presented a different approach. Simplicity was the angle the researcher took to implement the solution. The integration of multiple programming systems and languages as well as abstraction was key to develop the solution. As research still goes on in this field, this work has the following recommendation to make:

i. System administrators should focus more on the analysis of network behavior and coming up with more complicated rule structures as their networks grow. They should shift their focus from learning BASH commands for different firewall programs but look at better ways to structure firewall rules. Irrespective of the language, these rules, once effective, are written and embedded in any programming language to a BASH command as demonstrated above.

ii. Security when dealing with PHP scripts and web servers that can perform administrative commands should be key. We recommend firewalls to be configured on a block all basis then use this web tool for management and subsequent configurations. Besides that, the web server hosting the PHP scripts and APIs should be hardened and secured, as it is where the variable and values for configurations pass.

7.3 Future Works

To leverage on the achievement of simplicity and abstraction to firewall management that this research has presented, the following discussion on the future works that stem from this study:

i. Creating an SMS mode for firewall management and simple configuration, specifically for quick and important tasks like disabling the firewall, checking the status and manipulating the firewall based on simple parameters like a single port number or ip address. This is because, owing to the nature of the webpage, it may grow, become heavy on the browser and take time to load. A simple SMS code could perform basic but critical operations on the go as the web page loads.

ii. This research equally proposes that in future, similar systems should implement a scenario that covers more than just the LAN. Increased area of scope will be important, as it will call for managing and manipulating more firewall services on different machines in different networks on single portal.

References

Akins, A. (2003). Agile software development [Book Review]. *IEEE Software*, *20*(1), 97-98.

Ali, W. N. A. W., Taib, A. M., & Idrus, S. Z. S. (2018). I6-FPS: Automating the ICMPv6 Filtering Rules. In *MATEC Web of Conferences* (Vol. 150). EDP Sciences.

Bärwaldt, E. (2018). Graphical tools for firewall configuration. *Linux Magazine*, (207/2018). Retrieved 24 June 2020, from https://www.linux-magazine.com/Issues/2018/207/GUI-Firewall-Tools

Bernard, H. R., & Bernard, H. R. (2013). *Social research methods: Qualitative and quantitative approaches*. Sage.

Cao, H., Falleri, J. R., & Blanc, X. (2017, November). Automated generation of REST API specification from plain HTML documentation. In International Conference on Service-Oriented Computing (pp. 453-461). Springer, Cham.

Chhawchharia, V. (2019). Test Driven Development for Improvements in Quality and Desired Results. Retrieved 9 September 2019, from http://palztechnologies.com/test-driven-development.html

De B. (2017) Designing a RESTful API Interface. In: API Management. Apress, Berkeley, CA

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis and design: An object-oriented approach with UML*. John wiley & sons.

Desai, D., Sengupta, S., Novillo, J., & Andriamanalimanana, B. (2015). Representational State Transfer as a Web Service.

Etikan, I., & Bala, K. (2017). Sampling and sampling methods. *Biometrics & Biostatistics International Journal*, 5(6), 00149.

Faisandier, A. (2013). *Systems architecture and design*. Belberaud, France: Sinergy'Com.

Fielding, R. T., & Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures* (Vol. 7). Irvine: University of California, Irvine.

Frahim, J., Santos, O., & Ossipov, A. (2014). *Cisco ASA: All-in-one Next-Generation Firewall, IPS, and VPN Services*. Cisco Press.

Haupt, F., Leymann, F., Scherer, A., & Vukojevic-Haupt, K. (2017, April). A framework for the structural analysis of REST APIs. In 2017 IEEE International Conference on Software Architecture (ICSA) (pp. 55-58). IEEE.

How to design a REST API – REST API Tutorial. (2019). Retrieved 25 September 2019, from https://restfulapi.net/rest-api-design-tutorial-with-example/

https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/25000/PD25426/en_US/NGFW_5_8_0_HW_Requirements.pdf

https://www.illumio.com/hubfs/Downloads/Illumio_White_Paper_Eliminating_Firewall_Rule_Proliferation_2019_03.pdf ILLUMIO. (2019). Retrieved 9 September 2019, from

Janert, P. (2004). Introducing Test-Driven Software Development. IEEE Software, 21(6), 100-101. doi: 10.1109/ms.2004.42

Jayathilaka, H., Krintz, C., & Wolski, R. (2014). Service-Driven Computing with APIs: Concepts, Frameworks, and Emerging Trends. In *Handbook of Research on Architectural Trends in Service-Driven Computing* (pp. 355-379). IGI Global.

Kaplinger, T. E., Moore, V. S., & Nusbickel, W. L. (2018). U.S. Patent No. 9,959,363. Washington, DC: U.S. Patent and Trademark Office.

Kothari. (2004). Research Methodology: Methods and Techniques (2nd Revised Edition). New Delhi: New Age International Ltd.

Kothari. (2014). *Research Methodology: Methods and Techniques* (second Revised Edition). New Delhi: New Age International Ltd.

Kukan, M. (2019). Cisco ASA firewall REST API. Retrieved 9 September 2019, from https://maroskukan.wordpress.com/2015/02/11/cisco-asav-firewall-rest-api/

Kumari, S. (2017). REST based API. International Journal of Trend in Scientific Research and Development, Volume-1(Issue-4), 571-575. doi: 10.31142/ijtsrd2200

Kurland, V. (2010). What's new in Firewall Builder 3.0. *Linux Journal*, *2010*(189), 4.

Lucid chart. (2019). What Is the Extreme Programming Methodology? | Lucid chart Blog. Retrieved 9 September 2019, from https://www.lucidchart.com/blog/what-is-extreme-programming

M. Fowler, "Richardson maturity model: steps toward the glory of rest", http://martinfowler.com/articles/richardsonMaturityModel.html, 2010

Marczyk, G., DeMatteo, D., & Festinger, D. (2005). Essentials of research design and methodology. John Wiley & Sons Inc.

Massé, M. (2012). REST API design rulebook. Sebastopol, CA: O'Reilly.

McAfee. (2019). Retrieved 9 September 2019.

Mugenda, O. M., & Mugenda, A. (2003). Research methods: Quantitative and qualitative Approaches. Nairobi: African Centre for Technology Studies.

Negus, C. (2015). Linux bible. Indianapolis, IN: Wiley.

Nikolaidis, I. (2000). Firewalls: a complete guide [Books]. IEEE Network, 14(2), 6-6. doi: 10.1109/mnet.2000.826359

Raja, N. K., Arulanandam, K., & Somasundaram, R. (2012). Network Packet Inspection to Identify Contraband File Sharing Using Forensic Tools. *International Journal of Computer Network and Information Security*, *4*(3), 24.

Rebahi, Y., Hohberg, S., Shi, L., Parreira, B. M., Kourtis, A., Comi, P., & Ramos, A. (2015, December). Virtual security appliances: the next generation security. In *2015 International Conference on Communications, Management and Telecommunications (ComManTel)* (pp. 103-110). IEEE.

Richardson, L., & Ruby, S. (2008). *RESTful web services*. "O'Reilly Media, Inc.".

Rodríguez, Carlos, et al. "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices." International Conference on Web Engineering, Springer, 2016

Sekaran, U., & Bougie, R. (2016). *Research methods for business: A skill building approach*. John Wiley & Sons

Services, P., & Management, S. (2020). Cisco Adaptive Security Device Manager. Retrieved 26 June 2020, from https://www.cisco.com/c/en/us/products/security/adaptive-security-device-manager/index.html

Voronkov, A., Lindskog, S., & Martucci, L. A. (2015, October). Challenges in managing firewalls. In *Nordic Conference on Secure IT Systems* (pp. 191-196). Springer, Cham.

Wool, A. (2010). Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese. IEEE Internet Computing, 14(4), 58-65. doi: 10.1109/mic.2010.29

Appendices

Appendix A: PHP Code to Allow Traffic Based on Port Numbers

```
<?php
if( $_POST["s-port"] || $_POST["d-port"]) {
        $sport=$_POST['s-port'];
        $dport=$_POST['d-port'];
        if($sport <=65535){

                echo "The source/incoming port to be allowed is ". $sport. "<br />";
                $output=shell_exec('sudo iptables -A INPUT -p tcp --dport '.escapeshellarg($sport).
 -j ACCEPT');
                echo "<br/>". $output;
        }
        else{
                echo "Invalid port number";
        }


        if($dport <=65535){

     echo "The destination/outgoing port to be allowed is ". $dport. "<br />";
                $output=shell_exec('sudo iptables -A OUTPUT -p tcp --dport '.escapeshellarg($dport)
' -j ACCEPT');
                echo "<br/>". $output;
        }

        else{
                echo "Invalid port number";
        }

     exit();
   }
?>
```

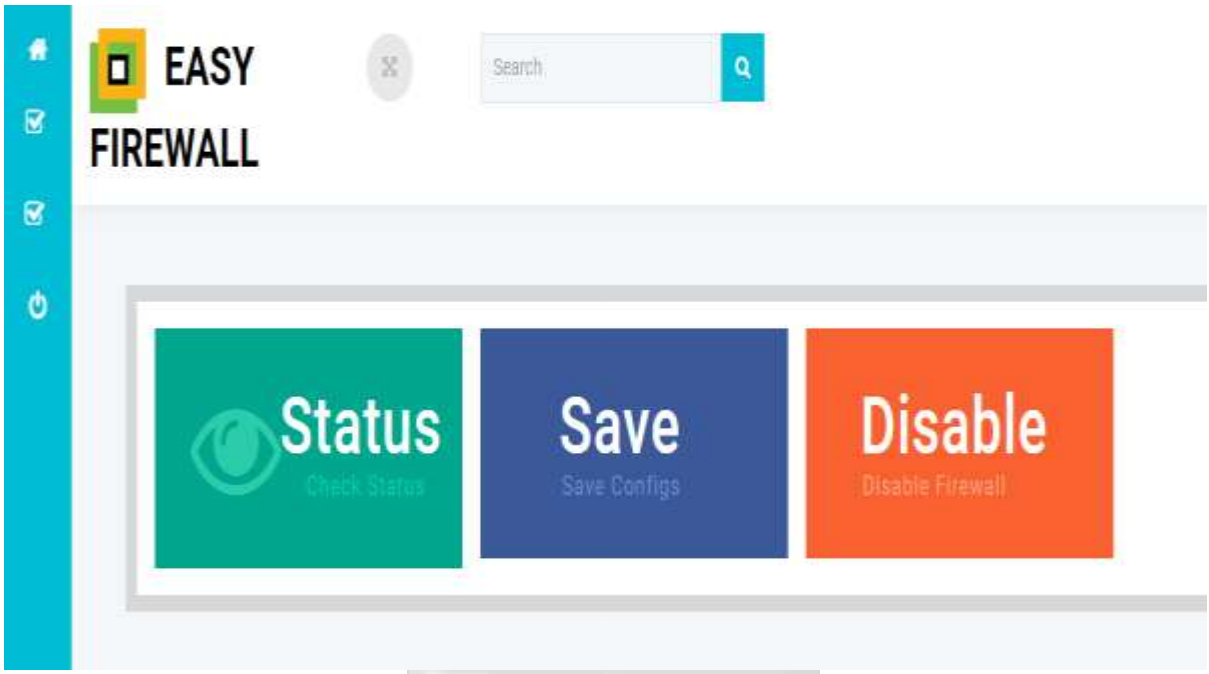
The code in appendix A shows a POST function used to allow traffic based on Port numbers. The variables s-port and d-port represent the source port number and destination port number. We implement a conditional statement to check whether the port number is within the valid range or not. When the condition is true, the Iptables modules executes the command to allow the input port on the firewall. The same happens to the destination port numbers.

Appendix B: PHP Code to Block Traffic Based on Port Numbers

```php
<?php
if( $_POST["s-port"] || $_POST["d-port"]) {
        $sport=$_POST['s-port'];
        $dport=$_POST['d-port'];
        if($sport <=65535){

                echo "The source/incoming port to be blocked is ". $sport. "<br />";
                $output=shell_exec('sudo iptables -A INPUT -p tcp --dport '.escapeshellarg($sport).'
 -j DROP');
                echo "<br/>". $output;
        }
        else{
                echo "Invalid port number";
        }


        if($dport <=65535){

     echo "The destination/outgoing port to be blocked is ". $dport. "<br />";
                $output=shell_exec('sudo iptables -A OUTPUT -p tcp --dport '.escapeshellarg($dport).
' -j DROP');
                echo "<br/>". $output;
        }

        else{
                echo "Invalid port number";
        }

     exit();
   }
?>
~
```

The code in appendix B shows a POST function used to allow traffic based on Port numbers. The variables s-port and d-port represent the source port number and destination port number. We implement a conditional statement to check whether the port number is within the valid range or not. When the condition is true, the Iptables modules executes the command to block the input port on the firewall. The same happens to the destination port numbers.

Appendix C: Simple Firewall Management Panel



Appendix C presents a web page with different buttons the user has and can use to manage the firewall.

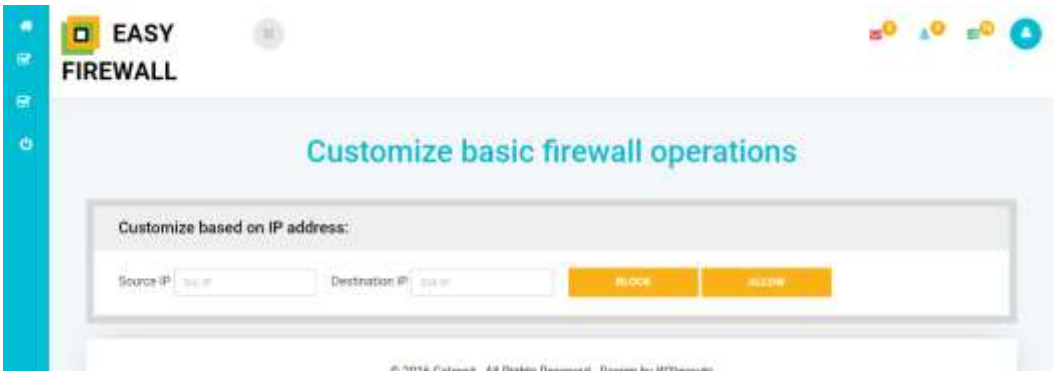
Appendix D: Simple Web Page Form

Appendix D presents a simple web page form the user can use to configure the firewall.

Appendix E: Port Numbers Firewall Management

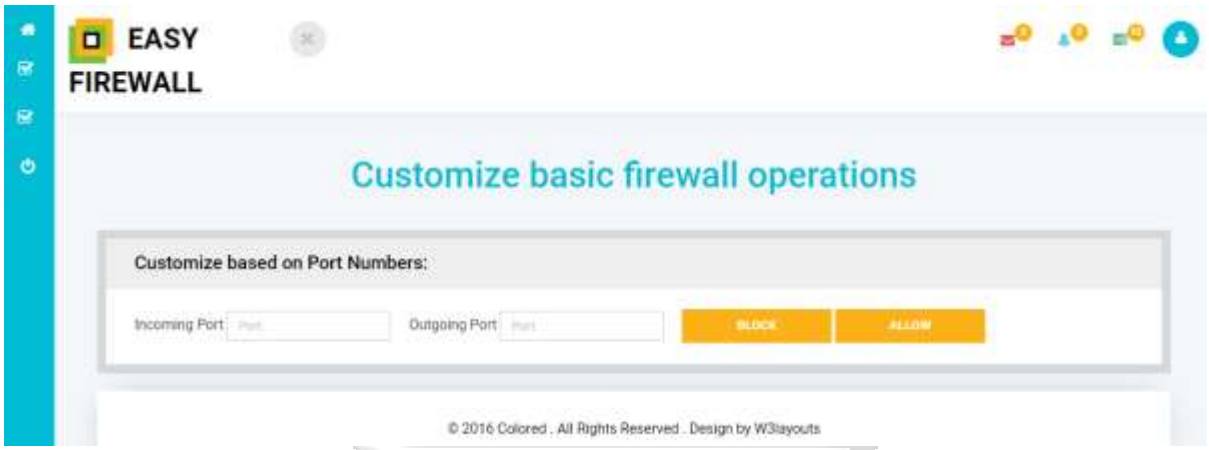
Configuration of the firewall is done using port numbers as shown in Appendix E. The incoming and outgoing port can be specified. However, in this solution, we allow the user to specify either of the ports or all. By clicking the block or allow buttons, the firewall is configured to drop or accept the connection on the specified port number respectively.

Appendix F: IP Addresses Firewall Management

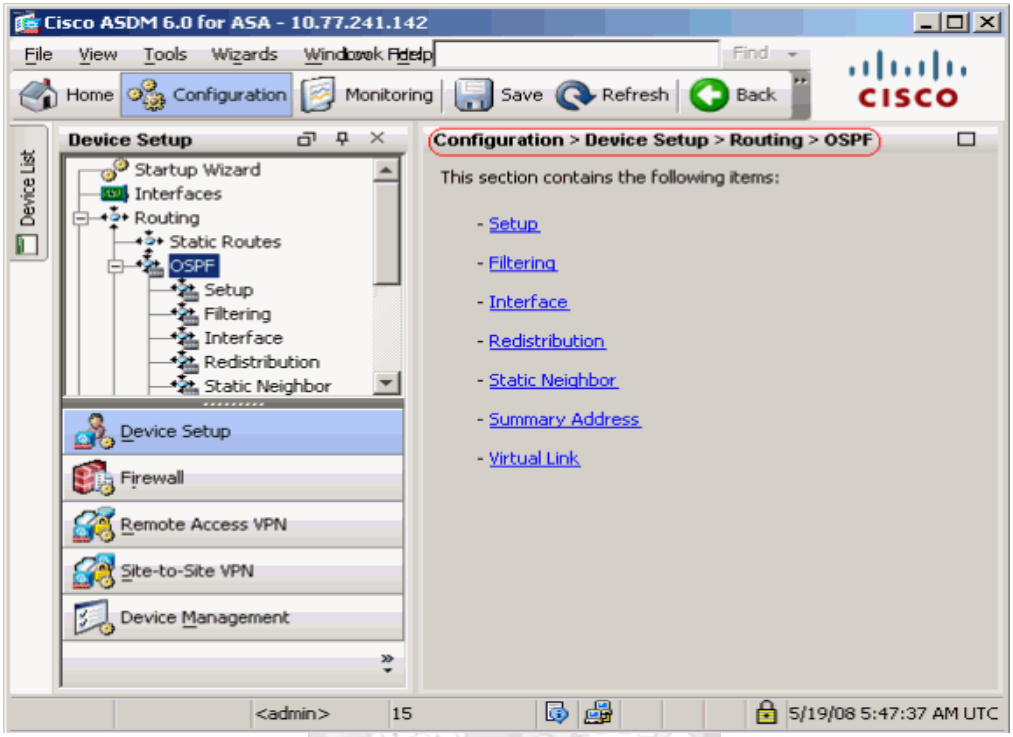


Configuration of the firewall is done using ip addressed as shown in Appendix F. The source and destination IP addresses can be specified. However, in our solution, we allow the user to specify either of the IP addresses or all. By clicking the block or allow buttons, the firewall is configure to drop or accept the connection on the specified IP addresses ber respectively.

Appendix G: Simple Web UI for Firewall Configuration



Appendix H: Cisco ASA Firewall Configuration Page



Appendix I: Originality Report