Infrastructure-Based Access Policy Enforcement Using Software-Defined Networks

by

Tyler Carden Culp

Bachelor of Science
Computer Science
Florida Institute of Technology
2016

A thesis
submitted to Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
December, 2018
We the undersigned committee
hereby approve the attached thesis

Infrastructure-Based Access Policy Enforcement Using Software-Defined Networks by Tyler Carden Culp

Thomas C. Eskridge, Ph.D.
Associate Professor
Computer Engineering & Sciences
Committee Chair

Barry Webster, Ph.D.
Assistant Professor
Mechanical and Civil Engineering
Outside Committee Member

Marco Carvalho, Ph.D.
Professor
Computer Engineering & Sciences
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Computer Engineering & Sciences
ABSTRACT

Title:
Infrastructure-Based Access Policy Enforcement Using Software-Defined Networks

Author:
Tyler Carden Culp

Major Advisor:
Thomas C. Eskridge, Ph.D.

This thesis describes a method to enhance network security using software defined networks. Standard networks use perimeter-based defenses to block attackers from gaining access to internal systems. A key problem with standard networks is that once a malicious entity has gained access to the network, they are often able to freely move throughout the network and to attack internal systems with impunity. This problem can be mitigated by placing defenses such as firewalls between machines on the network, but this approach requires significant resources and constant maintenance. If the network infrastructure itself is leveraged as a defense by individualizing the visibility of the network for each user according to their roles and permissions, then the resulting network will eliminate most or all of the actions attackers would take to monitor and attack the network from the inside. This type of defense requires identifying the sources of communication, enforcing global permissions, and dynamically updating the user’s view of the network.
# Table of Contents

Abstract iii  
List of Figures ix  
List of Tables xi  
List of Listings xiii  
Abbreviations xv  
Acknowledgments xviii  
Dedication xix  

1 Introduction 1  
1.1 Demonstration of the Need 2  
1.2 Problem Statement 4  
1.3 Thesis Statement 5  
1.4 Thesis Content 6  
1.5 Original Contribution 6  

2 Background 7  
2.1 Firewalls 8
### 2.2 VLANs

2.3 Software Defined Networks

2.4 Other Related Work

### 3 Design of the Framework

3.1 Base

3.2 Identification

3.2.1 Machine Identification

3.2.2 User Identification

3.3 Level System

3.4 Multi-Factor Authentication

3.5 Encryption

### 4 Implementation

4.1 Flow of Execution

4.1.1 Standard Network is Not Using the VPE System

4.1.2 Source Machine has FullAccess to Destination

4.1.3 Source Machine has NoAccess to Destination

4.1.4 Source Machine has AuthAccess to Destination

4.1.5 Source Machine is Not Running the Agent

4.1.6 Other Variations

4.2 Component Details

4.2.1 Controller

4.2.2 Rule Engine

4.2.3 Router

4.2.4 Agent
4.2.5 Interoperability with Other Programs .................................. 68
4.2.6 Signature Validation .................................................. 70

5 Experimentation .......................................................... 72

5.1 Experiment Network Layout ............................................ 72
  5.1.1 Network Configurations ......................................... 73
  5.1.2 VPE Engine Configuration ...................................... 74
  5.1.3 Experiment Scenarios .......................................... 77

5.2 Statistical Analysis .................................................... 78

5.3 Experiments ............................................................. 79
  5.3.1 Enforcing Visibility of other Machines ....................... 79
  5.3.2 Enforcing Traffic Based On Port .............................. 82
  5.3.3 Ping Requests .................................................. 87
  5.3.4 Downloading a Web Page Locally ............................... 90
  5.3.5 Downloading Web Pages from the Internet ................... 93
  5.3.6 Supporting Long-Running Connections ....................... 96

6 Discussion and Conclusion .............................................. 99

6.1 Difficulties Identified ................................................. 99
  6.1.1 ICMP Traffic .................................................. 99
  6.1.2 ARP Traffic .................................................. 100
  6.1.3 Inter-Component Traffic ...................................... 100
  6.1.4 Broadcast Traffic ............................................. 101

6.2 Additional Work ....................................................... 101
  6.2.1 Use of Other SDN Protocols .................................. 101
  6.2.2 Inclusion of Other Network Tools ............................. 102
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3 Merging Components</td>
<td>103</td>
</tr>
<tr>
<td>6.2.4 Caching</td>
<td>104</td>
</tr>
<tr>
<td>6.2.5 Distributed Computation</td>
<td>104</td>
</tr>
<tr>
<td>6.2.6 Fault Tolerance</td>
<td>105</td>
</tr>
<tr>
<td>6.2.7 Indexing</td>
<td>106</td>
</tr>
<tr>
<td>6.2.8 MFA Strictness Options</td>
<td>106</td>
</tr>
<tr>
<td>6.2.9 Rule Aggregation</td>
<td>110</td>
</tr>
<tr>
<td>6.2.10 Proactive Agent and Pre-Authorization</td>
<td>111</td>
</tr>
<tr>
<td>6.2.11 Efficiency</td>
<td>112</td>
</tr>
<tr>
<td>6.2.12 Auditing</td>
<td>113</td>
</tr>
<tr>
<td>6.2.13 Additional Interfaces</td>
<td>113</td>
</tr>
<tr>
<td>6.3 Conclusion</td>
<td>114</td>
</tr>
<tr>
<td>References</td>
<td>115</td>
</tr>
<tr>
<td>Appendices</td>
<td>122</td>
</tr>
<tr>
<td>A OpenWrt</td>
<td>124</td>
</tr>
<tr>
<td>A.1 Compiling OpenWrt with OpenFlow</td>
<td>124</td>
</tr>
<tr>
<td>A.1.1 Initial Dependencies and Setup</td>
<td>124</td>
</tr>
<tr>
<td>A.1.2 Patching Dependency Checking Issue</td>
<td>125</td>
</tr>
<tr>
<td>A.1.3 Adding OpenFlow to OpenWrt</td>
<td>126</td>
</tr>
<tr>
<td>A.1.4 Update and Install Feeds</td>
<td>126</td>
</tr>
<tr>
<td>A.1.5 Final Settings and Running Build</td>
<td>126</td>
</tr>
<tr>
<td>A.2 Installing OpenWrt in VirtualBox</td>
<td>127</td>
</tr>
<tr>
<td>A.2.1 Configuring OpenVSwitch</td>
<td>129</td>
</tr>
</tbody>
</table>
B Installing Base Dependencies

B.1 Installing Python and Pip ............................................. 131
B.2 Installing the Ryu Controller ....................................... 132

C Installing System Components

C.1 Installing and Running the VPE Controller ....................... 134
C.2 Installing the VPE Engine ............................................. 135
C.3 Installing the VPE SwitchAgent ...................................... 135
C.4 Installing the VPE Agent .............................................. 136
   C.4.1 Kernel Module Component ...................................... 136
   C.4.2 Userspace Component ........................................... 137
List of Figures

1.1 A traditional network ................................................. 4
3.1 The three access levels a user can have with an endpoint .... 19
4.1 The components of the VPE System ............................... 21
4.2 Standard Network - Step 1 ............................................ 23
4.3 Standard Network - Step 2 ............................................ 24
4.4 Standard Network - Step 3 ............................................ 25
4.5 Standard Network - Step 4 ............................................ 26
4.6 Agent with FullAccess - Step 1 ...................................... 29
4.7 Agent with FullAccess - Step 2 ...................................... 30
4.8 Agent with FullAccess - Step 3 ...................................... 31
4.9 Agent with FullAccess - Step 4 ...................................... 32
4.10 Agent with FullAccess - Step 5 ..................................... 33
4.11 Agent with FullAccess - Step 6 ..................................... 34
4.12 Agent with FullAccess - Step 7 ..................................... 35
4.13 Agent with FullAccess - Step 8 ..................................... 36
4.14 Agent with FullAccess - Step 9 ..................................... 37
4.15 Agent with FullAccess - Step 10 ................................... 38
4.16 Agent with FullAccess - Step 11 ................................... 39
4.17 Agent with FullAccess - Step 12 ................................... 40
4.18 Agent with *NoAccess* - Step 9 ................................... 42
4.19 Agent with *NoAccess* - Step 10 ................................. 43
4.20 Agent with *NoAccess* - Step 11 ................................. 44
4.21 Agent with *NoAccess* - Step 12 ................................. 45
4.22 Agent with *AuthAccess* - Step 9 ................................ 48
4.23 Agent with *AuthAccess* - Step 10 ............................... 49
4.24 Agent with *AuthAccess* - Step 11 ............................... 50
4.25 Agent with *AuthAccess* - Step 12 ............................... 51
4.26 No Agent - Step 5 .................................................... 54
4.27 No Agent - Step 6 .................................................... 55
4.28 No Agent - Step 7 .................................................... 56
4.29 No Agent - Step 8 .................................................... 57
4.30 Signature verification process ........................................ 69

5.1 The layout of the base experiment network .......................... 73
5.2 The horizontal network scan experiment ............................. 80
5.3 The partial vertical scan experiment ................................ 83
5.4 The full vertical scan experiment .................................... 83
5.5 The ping experiment ................................................... 88
5.6 The internal web page download experiment ......................... 91
5.7 The external web page download experiment ....................... 93
5.8 The FTP download experiment ....................................... 96
List of Tables

4.1 Standard Network flow of execution ..................................... 23
4.2 Standard Network communications ........................................ 27
4.3 Agent Exists, FullAccess (AL2) flow of execution ......................... 29
4.4 Al2 Network communications .............................................. 41
4.5 Agent Exists, NoAccess (AL0) flow of execution ........................ 42
4.6 Al0 Network communications .............................................. 46
4.7 Agent Exists, AuthAccess (AL1) flow of execution ......................... 48
4.8 Al1 Network communications .............................................. 52
4.9 No Agent (NA) flow of execution ......................................... 54
4.10 No Agent Network communications ....................................... 58
4.11 Components of a Rule ................................................... 61

5.1 Defined experimentation Users ............................................. 75
5.2 Relevant experimentation Ports ............................................. 75
5.3 Defined experimentation Networks ....................................... 75
5.4 Defined experimentation Rules ............................................ 76
5.5 Horizontal scan results .................................................. 81
5.6 Partial vertical scan results .............................................. 85
5.7 Partial vertical scan significance ......................................... 86
5.8 Full vertical scan results ............................................... 86
5.9 Full vertical scan significance ............................................. 87
5.10 Ping Results ................................................................. 89
5.11 Ping significance ............................................................ 90
5.12 Internal web traffic results ................................................. 91
5.13 Internal web traffic significance ......................................... 92
5.14 External web traffic results ............................................... 94
5.15 External web traffic significance ....................................... 95
5.16 File transfer results ........................................................ 97
5.17 File transfer download significance .................................... 98

6.1 Alternative MFA strategies ................................................. 107
List of Listings

4.1 Pseudocode for validating response keys .......................... 71
5.1 Horizontal scan command ........................................... 80
5.2 Partial vertical scan command ...................................... 84
5.3 Full vertical scan command ......................................... 84
5.4 Ping experiment command ........................................... 88
5.5 Internal web page download experiment command ............... 91
5.6 External web page download experiment command ............... 94
5.7 FTP experiment command ........................................... 97
A.1 Installing OpenWrt build dependencies ......................... 124
A.2 Applying git patches to OpenWrt ................................. 125
A.3 Linked OpenFlow into the OpenWrt build ....................... 126
A.4 Updating and installing OpenWrt feeds ......................... 126
A.5 Make menuconfig command ....................................... 126
A.6 Resize virtualbox image .......................................... 127
A.7 Example /etc/config/network using DHCP ..................... 128
A.8 Example /etc/config/network using static IP .................. 129
A.9 OpenVSwitch setup commands .................................. 130
B.1 Example /etc/apt/sources.list .................................. 132
B.2 Install Python and Pip command ............................... 132
B.3 Install Pip dependencies .................................. 132
B.4 Install *Ryu* command ................................. 133
C.1 Run *VPE Controller* ................................... 134
C.2 Install Python in OpenWrt .............................. 135
C.3 Install the *VPE KernelModule* ....................... 136
C.4 Install libelf-dev ........................................ 136
List of Symbols, Nomenclature or Abbreviations

Address Resolution Protocol (ARP) a protocol used to map IP addresses to ethernet addresses on the local network

control protocol a protocol or API that allows network switches to be remotely configured

destination / target machine the machine receiving traffic from another machine

Domain Name System (DNS) a network protocol used to resolve host names with IP addresses

Dynamic Host Configuration Protocol (DHCP) a protocol used to dynamically configure machines with various information, such as IP addresses, DNS servers, and more

dynamic policies policies that determine forwarding rules that can change rapidly and often, such as allowing a certain user to access a given system only during a specific time range, restricting or allowing access until a certain condition has been met, etc
endpoint

- a subnet or machine, optionally combined with a set of ports

entity

- a user, service, process, or attacker that is connecting to endpoints on the network

File Transfer Protocol (FTP)

- a protocol used to transfer files between machines

forwarding rules / rules

- rules on network switches that define how traffic should be handled, such as redirecting the traffic to another system, dropping the traffic, etc

Internet Control Message Protocol (ICMP)

- a protocol used to send operational status information

Internet Protocol (IP) Address

- an identifier assigned to a network endpoint that uses the Internet Protocol

JavaScript Object Notation (JSON)

- a test-based data interchange format

Linux

- an open-source operating system, widely used for devices such as servers and routers

Media Access Control (MAC) Address

- a unique identifier assigned to a physical network interface controller

multi-factor authentication (MFA)

- the use of multiple methods to verify identity, such as through RSA tokens [1] or external applications like Duo [2]

network flow / flow

- a collection of packets based on shared information, such as traveling between the same source and destination
Public Key Infrastructure (PKI) a security scheme that allows for cryptographic operations through the use of mathematically linked keys [3]

software-defined network (SDN) a network consisting of switches running a common control protocol, and a controller that uses said protocol to configure switches remotely

source machine the machine sending traffic to another machine

user a human being that is expected to be present on the network
Acknowledgements

I would first like to thank my thesis advisor, Dr. Thomas Eskridge, for all the assistance he’s provided me. Throughout this process, his door was always open when I had a question, and he provided helpful insight every time I came to him. His expertise has allowed this project to flourish, and I have benefited greatly from his teachings.

Next, I would like to thank the other members of my thesis committee, Dr. Barry Webster and Dr. Marco Carvalho, for providing valuable comments on my thesis. Dr. Webster in particular assisted me with the statistical analysis, as he provided input that helped me determine which statistical tests to perform and how to interpret the results.

I would also like to thank the Florida Institute of Technology, which has been a place of home and learning for the last several years. I have greatly enjoyed my time here, and I am proud be an alumnus of this great university.

Finally, I would like to thank my family and friends, for their love, support, and willingness to listen to my brainstorming over various aspects of this project. I especially thank my girlfriend, who has helped push me to succeed ever since we met, and without whom this thesis and graduate school itself would have been much more difficult.
Dedication

To my parents, who have shaped me into the man I am today. From continuously pushing me to improve and excel, to providing support of all kinds, they have always helped me grow and enabled me to accomplish all that I have. Without them, I would not be where I am today.
Chapter 1

Introduction

On most modern networks, anyone who can log into a computer on a network is considered to be trusted and has full visibility to all of the computers and services present in the network. However, most users of a network do not require the ability to access – or even know of the existence of – most network services and should only have visibility and access to the services they need. For example, a visiting guest may need to access the Internet, but is unlikely to require the ability to communicate with other machines and devices on the network. Therefore, this guest also does not require the ability to even view these other network devices, so even their visibility represents a vulnerability.

In order for computer networks to be secure, access to the network must be customized to the needs and privileges of each user. In this thesis, I develop a system which makes these concepts of visibility and access equivalent, as there is no visibility without permitted access.

One of the most common methods to mitigate these issues is to segment the network by making multiple, smaller networks within the primary network, known
as subnets. This can often be seen with wireless networks, where there may be a regular (secure) network, as well as a guest (insecure) network. However, while this may prevent individuals from accessing the entire network, they can still fully access endpoints present on the subnet. Additionally, if anyone is able to get onto the secure network, they will again have access to all endpoints on that network.

While basic segmentation can mitigate some of this issue, it is insufficient to truly protect machines on a network from one another.

1.1 Demonstration of the Need

Most systems, especially those that communicate over a network, contain vulnerabilities. These vulnerabilities are often exploited by malicious entities in order to gain control over a system and access secure resources. When there is only a single entry point into a network, such as a lone public-facing web server or a proxy, this can be slightly mitigated through the use of a firewall between the server and the rest of the network. However, there is rarely only a single entry point on a network, and in general, many machines on a network are able to access the Internet. These can range from servers, which must access the Internet to install updates or perform other tasks, to user machines, which will be used to browse various web sites, read emails, download known and unknown applications, etc. Even if a firewall is placed between the Internet and the rest of the network, it is virtually impossible to prevent every possible exploit from providing a malicious entity access to the network.

Once an attacker gains access to a single system on a network, it is generally much easier to continue on to other network resources. This is because most
networks consider traffic originating from within the network to be trusted, and do not provide the same level of security as for traffic originating from other networks. Someone such as a company employee, with access to a secure database, may only be able to access the database from their machine. However, if their machine is compromised, such as by getting a virus from an email, an attacker can use that machine as a middle-man and access the database as well. Even if the user removes the virus, an attacker could add a backdoor to the database, either on the database server itself or another machine on the network.

For example, this exact situation may have occurred during the Yahoo breach that occurred in October of 2014 [4, 5]. A hacker gained access to a Yahoo’s network, likely through a single employee’s machine via a phishing email. The hacker then installed a backdoor on a Yahoo server, located the user database and database management tool, and copied the database onto his local machine. This gave the hacker access to personal information of users, along with security questions and encrypted credentials [6]. Yahoo also experienced a previous attack in August 2013, in which the hacker was able to use the database to compromise over a billion Yahoo accounts [7].

Because of attacks like this one, it can not be assumed that machines on the network are trustworthy, and steps must be taken to ensure that even local entities are prevented from gaining unauthorized access to sensitive resources. When setting up a system that is intended to be secure or restricted, it is common to assume that the network is already compromised by an attacker. Even with this assumption, it can be difficult to secure a system against the network. If a firewall is placed between the server and the rest of the network, then rules have to be set on it to only allow authorized machines to access it. However, such rules can
potentially be defeated by spoofed Internet Protocol (IP) or Media Access Control (MAC) addresses. Similar issues can be found when instead using a Virtual Local Area Network (VLAN). A software-defined network can also be used, but that still would generally require setting additional firewall or VLAN rules.

1.2 Problem Statement

![A traditional network](image)

Figure 1.1: A traditional network

In traditional networks, users and other entities tend to have network access to systems and services which they are not intended to ever use, enabling malicious entities to work toward exploiting these to their own benefit. For example, a payroll database may only be utilized by members of an HR department, but the login page can often be reached by anyone on the network. If a malicious entity gains access to the network, either through connecting to it directly from their machine or through exploiting remote access to a machine already on the network, they are considered trusted and can then view and access the entirety of the network. Even
if they are authorized to access secure resources, a compromised machine or set of
credentials can allow attackers to access these resources themselves.

Rather than providing full network access to any entity present, each individual
machine could be given its own network permissions and access. However, in
organizations with multi-user systems, this would not be sufficient. Since multiple
users can sign on to multi-users systems simultaneously, and these individual users
will often have different permissions on the network, providing a machine with
only a single set of permissions will either restrict users with higher permissions,
or enable users with lower permissions to access restricted endpoints, or possibly
both at once. By personalizing the network view for each user, you can prevent
some users from accessing restricted systems while allowing other users to access
the same system, even from the same machine.

1.3 Thesis Statement

My thesis is that in order for computer networks to be secure, access to the net-
work must be customized to the needs and privileges of each user. This approach
makes the concepts of visibility and access equivalent. I developed a system called
Visibility Policy Enforcement (VPE) that implements these concepts.

While such permissions would be difficult to enforce in traditional networks,
when users can log in from any number of access points, a software-defined network
would be well equipped to perform the task of updating access configurations across
the network.
1.4 Thesis Content

In Chapter 2, I discuss the background behind this thesis, such as previous work that has been done to mitigate these issues. In Chapter 3, I discuss the design of my framework, and the reasoning behind it’s capabilities. In Chapter 4, I discuss the implementation of my framework, as well as the tests performed to validate the concept behind the framework and their results. In Chapter 5, I discuss the experiments run to measure the effects of the framework on traffic and latency, compared to a standard network and a simple SDN. In Chapter 6, I discuss additional work that could be performed on the system design and implementation, difficulties and issues encountered, and my conclusions on the effectiveness and capabilities of the framework. In the Appendices, I provide instructions for setting up the VPE System and its requirements.

1.5 Original Contribution

I present a framework for allowing administrators to assign network permissions for each user and enforce these permissions globally across a network. This will result in each individual user having a personalized view of the network, even when working with other users on the same machine and at the same time. This will be done through the use of a software-defined network, where user permissions will be set on the controller.

Additionally, these permissions will allow for the integration of multi-factor authentication, which will allow administrators to help protect secure resources from unauthorized entities.
Chapter 2

Background

A machine being present on a network automatically having full access to the network is a major issue in the realm of network security. Various solutions for this problem have been proposed, such as placing firewalls between secure resources and the rest of a network, using VLANs to segment a network, and using software-defined networks to modify the logical structure of a network without changing its physical layout. While their implementations may seem quite different, they all can be considered a form of segmentation.

Segmentation, in its base form, is splitting a single network into multiple networks. This can be done in a variety of ways. A network can be physically separated by removing all links between the two networks. However, this is often infeasible, either through the difficulty of modifying the physical aspects of the network, or because machines on the networks may need to communicate with other networks.

A firewall is a machine or service placed between segments of a network that blocks traffic based on rules. These rules often prevent machines on one side from
reaching certain machines or services on the other. Firewalls may be one of the most common security devices, as they can be found on ethernet switches, home routers, and even machines themselves. With rules such as these, a firewall can be considered a form of network segmentation, by blocking or restricting traffic from traveling between the portions of the network on either side.

VLANs are a feature present on many ethernet switches. This feature allows an administrator to assign one or multiple VLANs to each port. This causes all traffic coming from that port to be tagged with the given VLAN’s IDs. In general, traffic tagged with a particular VLAN ID can only exit the switch on another port tagged with that same VLAN, though exceptions exist. Additional rules can also be set. The result of this process is often a segmented network, where each individual VLAN can be considered its own logical network.

A software-defined network (SDN) is a type of network where the logical structure can be changed by a controller on the network, often through the use of a standardized protocol. Some software-defined network implementations utilize other methods as well, such as by virtually routing all traffic through a centralized firewall, or remotely setting VLAN configurations to segment a network.

2.1 Firewalls

Firewalls can either me used to protect a single machine, or to protect a network segment [8]. When used to protect a network, they act as a barrier between a network and the rest of the internet, and are often configured to let no unsolicited traffic into the network. While this does prevent many basic intrusion attempts, they are often incapable of preventing attacks such as phishing, where a user is
somehow convinced to allow invite malware into the network, such as by clicking a link in an email, and also often do not protect against malicious traffic originating from inside the network [9].

In [10], Adão et al. designed and implemented a system for defining the path packets can take through a network. For example, it can be defined that packets coming from outside the network must pass through a packet inspection region before continuing on to the destination. This is done through the use of Mignus, a declarative specification language for firewalls. This system is essentially a form of software-defined network, specifically implemented through the use of firewalls in routers.

Beteni et al. [8] developed an algorithm for classifying incoming traffic flows, in order to detect and block attacks, specifically those of the denial-of-service variety. This algorithm takes many different factors into account, such as the number of packets in the flow, the duration of the flow, the number of SYN packets in the flow, etc. The firewall then takes the results of the algorithm and either ignores the flow, or allows it to continue onto the network.

Sobeslav et al., in [9], introduce a system for protecting endpoints using local firewalls on each endpoint, communicating with each other and a central server, such as a security appliance. This allows for multiple distinct groups within a broader network to have their own security requirements and firewall rules, while still adhering to organizational policy. As the individual endpoints may not have the necessary computational ability to perform deep-packet inspection, flows can also be send to the central server.

These systems, and many similar ones not discussed here, work to improve the ability for firewalls to dynamically protect systems, instead of using hard-coded
rules. Moreover, [10] and [9] address the issue of local and malicious entities. Both systems can be used to protect against attackers inside the network, and even support defining rules from a centralized location. This is a vast improvement over the traditional perimeter approach. However, these systems do not address the issue of multiple users being located on the same machine, and they do not verify that the traffic originating from a machine is from an authorized user.

2.2 VLANs

VLANs are used to segment a network so that machines connected to different physical switches can communicate as if they were on the same switch [11]. They can also disallow communication between machines not on the same VLAN, even when connected to the same switch [12]. However, configuring switches to use proper VLANs can be a tedious process, and in many cases multiple switches will need to be configured to support a relatively simple change in a network’s topology [11], and other connected switches may need to be configured similarly. Even relatively simple tasks, like switching a network cable from one port to another on the same switch, may require a configuration change.

Guruprasad et al. [12] discuss various security features that are built into most ethernet switches. Basic VLANs simply allow ports to be configured to be members of certain VLANs, but this also means that anyone who can physically connect to a member port can be part of its VLANs. One method of mitigating this risk is by locking a port to only allow traffic from a specific device or list of devices, often based on MAC address. This technique is known as port/address locking. Another is to force devices connected to a port to authenticate, similarly
to wireless network authentication. Another feature discussed is stacked VLANs, which allows for multiple VLAN tags in a packet. These utilized by a series of switches to support a greater number of VLANS than normal.

In *Characterizing VLAN usage in an Operational Network* [11], Garimella et al. analyzed the usage of VLANs in a large university network. They found that VLAN usage was prevalent, with "a few hundred" VLANS on the Purdue network, and that the performance of the network was greatly impacted by the VLAN configuration. While not many configurations were missing VLAN membership (5 out of 131), but most configurations had unnecessary VLAN membership (119 out of 131). This helps illustrate the difficulties in manually configuring VLAN membership on large networks, and maintaining an efficient network and minimal configurations.

Alabady [13] proposes a network security model using an authentication, authorization, and accounting (AAA) server, along with VLAN configurations. This model requires users to authenticate with the AAA server, and the server verifies all traffic flows according to policy. This server works to mitigate many of the security issues with VLANs, such as VLAN hopping attacks, as well as more general attacks, such as MAC address spoofing. However, VLANs still need to be configured manually and in accordance with the security model defined.

Li et al. [14] introduce a system for managing VLANs in a centralized, semi-automatic manner. This system, named CSS-VM, uses the network topology and a list of user groups to compute an optimal VLAN configuration across a network. This computation is split between two modules: one which handles the static configuration of VLAN group membership, and another than handles dynamic configurations based on the changing status of the network. This system was
tested on an enterprise network and was successfully able to recover switch failures by rearranging the logical network topology.

As shown in these publications, manual VLAN configuration is difficult to do efficiently, and unnecessary or invalid VLAN configurations can be extremely prevalent in large networks [11]. In addition, there are many techniques that can be used to bypass security mechanisms that only utilize basic VLANs [13]. Other systems have been designed to run on top of VLANs, such as automatic configuration servers [14] or additional traffic verifications [13]. The AAA server in [13] is similar to the Rule Engine component of my framework, but does not support multi-factor authentication or multiple users authenticating from a single machine simultaneously. Similarly, some of the functionality in the CSS-VM [14] can be related to my system’s functionality, namely the automatic configuration of switches based on permissions. However, my system is not intended to create a robust, optimal network topology, and instead focuses on policy enforcement.

2.3 Software Defined Networks

One of the main issues with enforcing security, especially in large networks, is that multiple physical switches must be configured manually. This can be made even more difficult if switches are in multiple and distant locations, such as in networks spanning multiple floors or buildings. Software-defined networks have been made to mitigate this problem by having all (or most) switches on a network configurable from the SDN controller. Some of the previously mentioned systems are an implementation of a software-defined network, such as [10], but they do not use a standardized protocol or controller.
In [15], a framework for writing algorithmic policies at a higher level than forwarding rules is presented. The user writes a function which is executed by a controller to determine a path through the network a packet must take. Maple then determines which parts of the function are predictable and repeatable, and optimizes and sets forwarding rules on the relevant switches. This reduces the load on the controller, as the defined function will only need to be run by the controller once for multiple packets that follow the same execution path through the function. This framework greatly simplifies SDN programming, as aspects such as the physical layout of the network and flow rules on the switches are abstracted and handled in the background.

Gude et al. [16], present NOX, a framework for an OpenFlow based network that is analogous to a computer operating system. This provides a programmable interface for the network, along with a centralized and generalized network view, that allows developers to create applications that can easily control the state of the network. The abstractions used, such as users and hostnames, make network configuration simpler than using lower-level concepts like MAC and IP addresses. The applications can perform tasks such as applying security policies through creating and modifying flow rules, monitoring and redirecting traffic, filtering unnecessary messages from machines, and other such functions that would be more difficult without a centralized view and conceptual abstractions.

Porras et al. [17] introduce FortNOX, an extension for the NOX [16] controller, which enforces security of the NOX controller. It contains a rule conflict detection engine, which ensures that new OpenFlow rules do not conflict with already existing rules. If there is a conflict, then the rule being set with higher authorization is given priority. Each OpenFlow application is assigned a role, which is assigned
an authorization level and verified using digital signatures. This greatly improves the security of the NOX controller, forcing applications to be authenticated with the controller before setting rules.

Software-defined networks are designed to improve network programmability by allowing for a controller to configure all switches on a network. This is especially useful in large networks, where configuring individual switches and routers can be an extremely tedious process. In addition, SDNs provide a level of granularity not generally provided by other solutions such as VLANs, by allowing rules to be set for individual flows rather than entire subnets. This essentially allows any switch to act as a firewall, thus making it a much for feasible task to secure a network against internal threats. It is even possible to require users to authenticate with the network before they are allowed to use it. However, this authentication is usually done on a per-machine basis, and many machines support multiple users. Additionally, if a malicious entity gains access to a machine, there is little guarantee that traffic coming from a given machine is being sent by the authenticated user.

2.4 Other Related Work

Zero Trust Networks refers to a concept where no traffic, even traffic originating from inside the network, is fully trusted. This means ensuring that all internal resources require authentication to access, monitoring and logging all traffic [18].

DeCusatis et al. [18] built a system that used authorized and unauthorized devices attempting to access important network resources. They blocked access for unauthorized devices at the transport layer, and prevented explicit denials from resources that would allow attackers to detect or fingerprint resources, thus
providing an individualized view of the network. However, they do not state how they authenticate a device, only mentioning the IP address of the authorized device. This system also can only totally enforce policy or never enforce policy, and does not support scenarios where someone may be able to access only certain ports of a resource, or may be able to detect a resource but not otherwise communicate. It also does not have a method of requesting access or verifying that the authorized terminal is not compromised.
Chapter 3

Design of the Framework

3.1 Base

Each individual switch will be able to allow or deny traffic based on the defined access policy. In order to facilitate this, a software-defined network will provide the base of the framework. When a new flow reaches a switch, it will contact a component of the system, called the VPE Engine, which will use the controller to set an access rule for the traffic. This will allow the entire network’s infrastructure to play a part in enforcing access policy, without the need for all traffic to travel through a centralized point. Instead, only a small amount of information about the traffic will need to be sent to the controller.

The only information sent to the controller will be that which is required to make a decision about the traffic, such as the machine and port the traffic is being sent from, the destination machine and port the traffic is going to, the user sending the traffic, etc.

The switch’s portion of the framework will be able to run on any hardware
capable of performing as an SDN switch and able to be modified to run the switch software. This is so that many hardware components will be usable by this system, rather than some existing solutions which require specialized hardware.

3.2 Identification

In this framework, machines and users need to be identified properly. While many similar systems identify machines based on IP address or MAC address, such information can be easily falsified. In order to provide a more accurate identification, the system will use a software agent that will run on many machines connected to the network. This agent will communicate with the machine’s connected switch to provide identification for the machine and the user initiating traffic.

3.2.1 Machine Identification

Machine identification will be provided through factors such as MAC address (or addresses), and verified through the use of Public Key Infrastructure (PKI) with rotating keys. Occasionally when a machine connects to a switch, the machine and the switch will generate a new set of keys to use for the next connection. Initial connections will be approved and validated by an administrator.

3.2.2 User Identification

Users will be primarily identified through their system username. This username reporting will be performed by the agent running on the system in response to a request from the switch. If the agent is not installed on a system, or is otherwise unreachable, the switch will report the user as a predefined unknown user.
3.3 Level System

There will be three levels of access each user has with each endpoint on the network, as shown in Figure 3.1.

Level 0 (NoAccess) means all traffic from the user to the endpoint will be blocked at the first switch, allowing for no communication. This will result in the user being unable to view the endpoint. In the case of a rule against ICMP communication, the machine will be unable to receive pings. In the case of a rule against communication with a certain port, it will appear as though the port on said machine is not open.

Level 1 (AuthAccess) will block the communication attempt until the user verifies that they intend to perform the action using multi-factor authentication, at which point the traffic will be permitted to proceed as normal.

Level 2 (FullAccess) will allow for full and unrestricted communication access to the endpoint, similar to a traditional network that is enforcing no policy or firewall rules.

These levels can be augmented with conditions that allow for dynamic policies. For example, a user can have FullAccess access to a system from 8:00 - 5:00, but only AuthAccess access otherwise. These three levels, combined with the flexibility provided by dynamic policies, cover the complete set of accessibility for an endpoint.

The main purpose of AuthAccess access is to mitigate the damage that can be done by an attacker pretending to be a certain user. For example, even if an attacker pretends to be a user who has access to a secure database, if the user has AuthAccess then they will still have to accept the attempt through multi-factor authentication. This would have likely prevented the attackers in the 2014 Yahoo
Figure 3.1: The three access levels a user can have with an endpoint breach from obtaining the user database.

### 3.4 Multi-Factor Authentication

Multi-factor authentication is often used in order to allow someone to identify themselves more definitively than they can with a single set of credentials. For example, someone logging into a website may have to accept a login request with their phone. This makes it more difficult for a malicious entity to pretend to be the person in question, as not only must they know the credentials of an account, but they must also have physical access to the user’s phone.

This framework utilizes multi-factor authentication in a similar way. When a user has AuthAccess access to a system and attempts to connect to said system, the network will force them to verify their identity through multi-factor authentication.
The framework will be able to utilize multiple different methods and authentication services, such as *Duo* [2], a text message reply, or some other similar tool. This way, even if an attacker gains control of the user’s machine and is present on the network, they will be blocked from reaching a secure system.

### 3.5 Encryption

All data sent across the framework will be encrypted. The agents running on machines will communicate to their switches using PKI with rotating keys, and the switches will also communicate with each other and the controller using PKI. This is to reduce the possibility of an attacker spoofing a machine, user, switch, or controller.
In order to test the hypothesis that users can operate in an environment where they have visibility and access to only those network resources that are sanctioned by their roles and permissions, we implemented a key set of functionality from the design presented in Chapter 3.

The overall system consists of several components working together across mul-
tiple machines, each of which is shown in Figure 4.1.

The VPE Controller informs switches whether to permit or deny traffic rule engine. This also acts as an SDN controller.

The VPE Engine handles all rule-based processing, such as creating or deleting rules, building representations of machines, users, or endpoints, etc.

The VPE Agent runs on host machines, and stores the user sending each packet.

The VPE SwitchAgent passes messages between the VPE Agent and VPE Controller. This component runs on routers on the network, which must be configured as members of an SDN.

The VPE AuthModule acts as a rudimentary multi-factor authentication system. It is designed to emulate a notification on a multi-factor authentication device, such as a phone, as apps that display a notification to authenticate are common, for example Duo [2] and Okta [19].

Each component is described in more detail in Section 4.2.

4.1 Flow of Execution

In this section, I will describe the flow of execution for the application, by following a packet through the network. There are several different possible execution paths, and each will be discussed separately.

4.1.1 Standard Network is Not Using the VPE System

This flow of execution represents that which occurs in many standard networks, where this system is not used and there are no internal security mechanisms in place.
Table 4.1: Standard Network flow of execution

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN-1</td>
<td>A user, service, or other process prepares a packet to be sent over the network.</td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Step #</td>
<td>Step Description</td>
<td>Image</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SN-2</td>
<td>The packet travels across the network to the nearest router.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.3: Standard Network - Step 2
Table 4.1 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN-3</td>
<td>The router forwards the packet to its destination.</td>
<td>Figure 4.4: Standard Network - Step 3</td>
</tr>
<tr>
<td>Step #</td>
<td>Step Description</td>
<td>Image</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SN-4</td>
<td>The packet arrives at its destination.</td>
<td><img src="image" alt="Figure 4.5: Standard Network - Step 4" /></td>
</tr>
</tbody>
</table>
In this scenario, two communications are sent over the network, which are summarized in Table 4.2.

Table 4.2: Standard Network communications

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>The packet traveling from the machine to the router</td>
<td>SN-2</td>
</tr>
<tr>
<td>The original packet traveling from the router to its destination</td>
<td>SN-3</td>
</tr>
</tbody>
</table>

No rules exist to be evaluated, no signatures are generated or validated, and the user needs to perform no authentication. There is also no security mechanism in place, and thus any user from any machine can access any open port on any other machine.

This flow of execution also applies to packets in the following scenarios when a rule result has been cached in the router. Once a rule has been evaluated by the VPE System, the action to take will be cached for a period of time as determined by the rule. Until the timeout passes and the action is removed, no further VPE System evaluations will occur for packets with the same parameters. This allows a stream of data to be only evaluated on the first packet, while remaining packets would proceed as in a standard network. Packets with different parameters will still go through the full VPE System flows of execution.

4.1.2 Source Machine has FullAccess to Destination

This flow of execution examines one of the most common situations where it is assumed that the source machine is running the VPE Agent, with a valid and known keypair from the user, and that it there is an active rule providing it with FullAccess to the destination. Any deviations from this flow will be outlined in
later subsections.

This flow is considered typical as users of the system will be primarily attempting to reach systems to which they have access. This will often include external networks, such as the Internet, as this security scheme is not designed to protect external networks. Additionally, traffic to external web services is extremely common, and is unlikely to require authentication with VPE System.

This is also considered the typical scenario since this scheme provides the most security when each machine is running the VPE Agent, as this is the primary method of filtering traffic by user and individualizing access permissions.
<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-1</td>
<td>A user, service, or other process prepares a packet to be sent over the network.</td>
<td>Figure 4.6: Agent with <em>FullAccess</em> - Step 1</td>
</tr>
</tbody>
</table>
The VPE Agent on the machine notices the packet before it is sent over the network, and records its identifying information locally.

### Table 4.3 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-2</td>
<td>The VPE Agent on the machine notices the packet before it is sent over the network, and records its identifying information locally.</td>
<td>Figure 4.7: Agent with FullAccess - Step 2</td>
</tr>
<tr>
<td>Step #</td>
<td>Step Description</td>
<td>Image</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| AL2-3  | The packet travels across the network to the nearest router. | ![](image)  
**Figure 4.8: Agent with FullAccess - Step 3**
Table 4.3 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-4</td>
<td>The router creates an OpenFlow message to represent the packet, and sends this message to the VPE Controller.</td>
<td><img src="image-url" alt="Diagram" /> Figure 4.9: Agent with FullAccess - Step 4</td>
</tr>
</tbody>
</table>
Table 4.3 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-5</td>
<td>The <em>VPE Controller</em> reads the OpenFlow message, extracts the packet’s identifying information, and sends a username query to the originating machine. This query contains the identifying information of the packet.</td>
<td>Figure 4.10: Agent with <em>FullAccess</em> - Step 5</td>
</tr>
</tbody>
</table>
The VPE Agent unpacks the query and checks its storage for the packet’s identifying information. If the information is found, the agent crafts a query response, which contains the username, the agent’s public key, and an RSA signature of the response. This response is then returned to the VPE Controller.

![Figure 4.11: Agent with FullAccess - Step 6](image)
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-7</td>
<td>The VPE Controller takes the query response and sends it to the VPE Engine.</td>
<td><img src="image-url" alt="Diagram" /></td>
</tr>
</tbody>
</table>

*Table 4.3 continued*

*Figure 4.12: Agent with FullAccess - Step 7*
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-8</td>
<td>The <em>VPE Engine</em> validates the response by checking the public key in the message. Assuming the public key is recognized and the signature matches, the source machine is identified. The <em>VPE Engine</em> then checks the saved rules, searching for the highest priority rule that matches the packet. If no rule is found, the level is defaulted to 0, but this description will assume that a rule providing <em>FullAccess</em> is found.</td>
<td><img src="image" alt="Figure 4.13: Agent with FullAccess - Step 8" /></td>
</tr>
</tbody>
</table>

Table 4.3 *continued*
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-9</td>
<td>The <em>VPE Engine</em> returns the evaluation of the packet to the <em>VPE Controller</em>. If the rule specifies <em>FullAccess</em>, then the response is that the packet is permitted to continue through the network.</td>
<td>Figure 4.14: Agent with <em>FullAccess</em> - Step 9</td>
</tr>
</tbody>
</table>
Table 4.3 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-10</td>
<td>The <em>VPE Controller</em> checks the response from the <em>VPE Engine</em> and, seeing that the packet is permitted, sends an OpenFlow message to the router to allow the packet to continue to its original destination.</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Table 4.3 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-11</td>
<td>The router forwards the packet to its original destination, and saves the decision from the VPE Controller. This allows future packets with the same identifying information to continue as well, without the need for the VPE Controller, VPE Agent, or VPE Engine to come to a verdict each time.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.16: Agent with FullAccess - Step 11
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL2-12</td>
<td>The packet arrives at its destination.</td>
<td><img src="image" alt="Diagram of network with labeled elements" /></td>
</tr>
</tbody>
</table>

Figure 4.17: Agent with *FullAccess* - Step 12
In this scenario, ten communications are sent over the network, which are summarized in Table 4.4.

Table 4.4: Al2 Network communications

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>The packet traveling from the machine to the router</td>
<td>AL2-3</td>
</tr>
<tr>
<td>The OpenFlow message being sent to the VPE Controller</td>
<td>AL2-4</td>
</tr>
<tr>
<td>The user query from the VPE Controller to the VPE SwitchAgent</td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user query from the VPE SwitchAgent to the VPE Agent</td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user response from the VPE Agent to the VPE SwitchAgent</td>
<td>AL2-6</td>
</tr>
<tr>
<td>The user response from the VPE SwitchAgent to the VPE Controller</td>
<td>AL2-6</td>
</tr>
<tr>
<td>The access query from the VPE Controller to the VPE Engine</td>
<td>AL2-7</td>
</tr>
<tr>
<td>The access query response from the VPE Engine to the VPE Controller</td>
<td>AL2-9</td>
</tr>
<tr>
<td>The OpenFlow message sent to the router</td>
<td>AL2-10</td>
</tr>
<tr>
<td>The original packet traveling to its destination</td>
<td>AL2-11</td>
</tr>
</tbody>
</table>

The minimum number of rule determinations is one if only a single rule is present. The maximum number of rule determinations matches the number of rules present in the VPE Engine. A single signature is generated by the VPE Agent, and a single signature verification is performed by the VPE Engine. The user does not need to authenticate in this scenario.

4.1.3 Source Machine has NoAccess to Destination

When the source machine has NoAccess to the destination, the flow of execution is entirely the same as in Section 4.1.2 through step 8. However, the following steps differ slightly, and are described in Table 4.5:
Table 4.5: Agent Exists, *NoAccess* (AL0) flow of execution

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
<th>Image</th>
</tr>
</thead>
</table>
| AL0-9  | The *VPE Engine* returns the evaluation of the packet to the *VPE Controller*. As the rule specifies *NoAccess*, the response is that the packet is not permitted to continue through the network. | ![Diagram](image.png)  
Figure 4.18: Agent with *NoAccess* - Step 9  
*The rule evaluation is sent to the controller*
Table 4.5 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL0-10</td>
<td>The <em>VPE Controller</em> checks the response from the <em>VPE Engine</em> and, seeing that the packet is not permitted, sends an OpenFlow message to the router to block the packet.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.19: Agent with *NoAccess* - Step 10
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL0-11</td>
<td>The router drops the packet, and saves the decision from the VPE Controller. This allows future packets with the same identifying information to continue as well, without the need for the VPE Controller, VPE Agent, or VPE Engine to come to a verdict each time.</td>
<td><img src="Figure_4.20.png" alt="Image" /></td>
</tr>
<tr>
<td>Step #</td>
<td>Step Description</td>
<td>Image</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>AL0-12</td>
<td>The packet fails to continue past the router. Most likely, the connection will eventually time-out.</td>
<td><img src="image" alt="Figure 4.21: Agent with NoAccess - Step 12" /></td>
</tr>
</tbody>
</table>
In this scenario, nine communications are sent over the network, which are summarized in Table 4.6.

Table 4.6: Al0 Network communications

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>The packet traveling from the machine to the router</td>
<td>AL2-3</td>
</tr>
<tr>
<td>The OpenFlow message being sent to the VPE Controller</td>
<td>AL2-4</td>
</tr>
<tr>
<td>The user query from the VPE Controller to the VPE SwitchAgent</td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user query from the VPE SwitchAgent to the VPE Agent</td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user response from the VPE Agent to the VPE SwitchAgent</td>
<td>AL2-6</td>
</tr>
<tr>
<td>The user response from the VPE SwitchAgent to the VPE Controller</td>
<td>AL2-6</td>
</tr>
<tr>
<td>The access query from the VPE Controller to the VPE Engine</td>
<td>AL2-7</td>
</tr>
<tr>
<td>The access query response from the VPE Engine the VPE Controller</td>
<td>AL0-9</td>
</tr>
<tr>
<td>The OpenFlow message sent to the router</td>
<td>AL0-10</td>
</tr>
</tbody>
</table>

The only missing communication between this scenario and AL2 is the original packet getting sent from the router to the destination. The minimum number of rule determinations is zero if no rules are present, as blocking is the default action. The maximum number of rule determinations matches the number of rules present in the VPE Engine. A single signature is generated by the VPE Agent, and a single signature verification is performed by the VPE Engine. The user does not need to authenticate in this scenario, as they only need to authenticate when they have AuthAccess to the destination.
4.1.4 Source Machine has \textit{AuthAccess} to Destination

When the source machine has \textit{AuthAccess} to the destination, the flow of execution is entirely the same through step 1, but there are an additional few steps between steps 8 and 9. After these additional steps, the flow of execution continues the same as with \textit{NoAccess} or \textit{FullAccess}.

This access level, \textit{AuthAccess}, is ideal for allowing users to access secure servers, while defending against the possibility of the \textit{VPE Agent} being compromised or spoofed by an attacker. Even if an attacker sends an incorrect or unauthorized response to a user query in order to masquerade as a more privileged user, they will not be able to access a system without also obtaining the multi-factor authentication device or credentials.

Due to the nature of this access level, rules set with this will most likely have a longer timeout than most, as the user will have to re-authenticate after each timeout if they are still using the connection.
Table 4.7: Agent Exists, *AuthAccess* (AL1) flow of execution

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL1-9</td>
<td>Since the source machine has <em>AuthAccess</em> to the destination, multi-factor authentication has to be used to determine whether the packet should be permitted to continue through the network or not. Thus, the VPE Engine sends an authentication query to the multi-factor authentication module set for the user sending the packet.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.22: Agent with *AuthAccess* - Step 9
Table 4.7 continued

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL1-10</td>
<td>The user receives a multi-factor authentication request from the module, and responds accordingly.</td>
<td><img src="image" alt="Figure 4.23: Agent with AuthAccess - Step 10" /></td>
</tr>
</tbody>
</table>

Figure 4.23: Agent with AuthAccess - Step 10
The multi-factor authentication module responds to the *VPE Engine’s* request.

Figure 4.24: Agent with *AuthAccess* - Step 11
Table 4.7 *continued*

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL1-12</td>
<td>The <em>VPE Engine</em> processes the result from the multi-factor authentication module. If the user selected that they did intend to send the packet, then their authentication level is assumed to be <em>FullAccess</em>, and the execution flow continues from the previously-described step 9. If the user did not intend to send the packet, or the multi-factor authentication module was unable to get a response, then their access level is assumed to be <em>NoAccess</em>, and the execution flow continues again continues as normal. This ensures that the <em>VPE Controller</em> only ever interacts with levels <em>NoAccess</em> and <em>FullAccess</em>, since <em>AuthAccess</em> is resolved to either <em>NoAccess</em> or <em>FullAccess</em> by the <em>VPE Engine</em>.</td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 4.25: Agent with *AuthAccess* - Step 12
In this scenario, eleven or twelve communications are sent over the network, depending on the result of the MFA request. These are summarized in Table 4.8.

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>The packet traveling from the machine to the router</td>
<td>AL2-3</td>
</tr>
<tr>
<td>The OpenFlow message being sent to the <em>VPE Controller</em></td>
<td>A2-4</td>
</tr>
<tr>
<td>The user query from the <em>VPE Controller</em> to the <em>VPE SwitchAgent</em></td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user query from the <em>VPE SwitchAgent</em> to the <em>VPE Agent</em></td>
<td>AL2-5</td>
</tr>
<tr>
<td>The user response from the <em>VPE Agent</em> to the <em>VPE SwitchAgent</em></td>
<td>AL2-6</td>
</tr>
<tr>
<td>The user response from the <em>VPE SwitchAgent</em> to the <em>VPE Controller</em></td>
<td>AL2-6</td>
</tr>
<tr>
<td>The access query from the <em>VPE Controller</em> to the <em>VPE Engine</em></td>
<td>AL2-7</td>
</tr>
<tr>
<td>The MFA request from the <em>VPE Engine</em> to the authentication provider</td>
<td>AL1-9</td>
</tr>
<tr>
<td>The MFA response from the authentication provider to the <em>VPE Engine</em></td>
<td>AL1-11</td>
</tr>
<tr>
<td>The access query response from the <em>VPE Engine</em> the <em>VPE Controller</em></td>
<td>AL2-9</td>
</tr>
<tr>
<td>The OpenFlow message sent to the router</td>
<td>AL2-10</td>
</tr>
<tr>
<td>POSSIBLY The original packet traveling to its destination</td>
<td>AL2-11</td>
</tr>
</tbody>
</table>

The minimum number of rule determinations is 1, as there must be a rule to give AuthAccess. The maximum number of rule determinations matches the number of rules present in the *VPE Engine*. A single signature is generated by the *VPE Agent*, and a single signature verification is performed by the *VPE Engine*. The user needs to authenticate in this scenario, and they will have to re-authenticate every time the rule evaluation times out. This amount of time is configurable in the rule.
4.1.5 Source Machine is Not Running the Agent

Often, a machine will be attempting to send packets without running the VPE Agent. This can be for a variety of reasons, such as the machine in question simply not having the agent installed, or perhaps because the machine is running an unsupported operating system, such as Windows or Android. In this case, the flow of execution will proceed as normal through step 4, and from step 9 onwards. However, the intermediate steps, 4 through 8, will differ.
Table 4.9: No Agent (NA) flow of execution

<table>
<thead>
<tr>
<th>Step #</th>
<th>Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA-5</td>
<td>Similarly to the previously-described step 5, the <em>VPE Controller</em> sends a query message to the machine for additional information. However, since the <em>VPE Agent</em> is not running on the source machine, or is otherwise unreachable, the <em>VPE SwitchAgent</em> will be unable to connect to the machine.</td>
<td><a href="#">Figure 4.26: No Agent - Step 5</a></td>
</tr>
</tbody>
</table>

Figure 4.26: No Agent - Step 5
<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA-6</td>
<td>The router responds to the <em>VPE Controller</em>, informing it that it was unable to connect to the source machine’s <em>VPE Agent</em>.</td>
<td>Figure 4.27: No Agent - Step 6</td>
</tr>
<tr>
<td>Step #</td>
<td>Step Description</td>
<td>Image</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NA-7</td>
<td>The <em>VPE Controller</em> takes the query response and sends it to the <em>VPE Engine</em>. This is exactly the same as in the original step 7, except the contents of the response differ.</td>
<td><img src="image" alt="Figure 4.28: No Agent - Step 7" /></td>
</tr>
</tbody>
</table>
Table 4.9 *continued*

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Description</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA-8</td>
<td>The <em>VPE Engine</em> evaluates the response. However, this is performed differently than in the original Step 8. Instead of validating the public key and signature of the source machine and using the response’s username, the username is set to a static <em>unknown</em> user, and the machine id is not set. However, the destination machine’s may still be identified through its IP address, if the IP addressed has been authenticated previously. The <em>VPE Engine</em> then searches through the list of rules like usual. After the <em>VPE Engine</em>’s evaluation, the flow of execution continues as normal from Step 9.</td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>
In this scenario, seven or eight communications are sent over the network, depending on the rule evaluation. These are summarized in Table 4.10.

Table 4.10: No Agent Network communications

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>The packet traveling from the machine to the router</td>
<td>AL2-3</td>
</tr>
<tr>
<td>The OpenFlow message being sent to the <em>VPE Controller</em></td>
<td>AL2-4</td>
</tr>
<tr>
<td>The user query from the <em>VPE Controller</em> to the <em>VPE SwitchAgent</em></td>
<td>NA-5</td>
</tr>
<tr>
<td>The response from the <em>VPE SwitchAgent</em> to the <em>VPE Controller</em></td>
<td>NA-6</td>
</tr>
<tr>
<td>The access query from the <em>VPE Controller</em> to the <em>VPE Engine</em></td>
<td>NA-7</td>
</tr>
<tr>
<td>The access query response from the <em>VPE Engine</em> the <em>VPE Controller</em></td>
<td>AL2-9</td>
</tr>
<tr>
<td>The OpenFlow message sent to the router</td>
<td>AL2-10</td>
</tr>
<tr>
<td>POSSIBLY The original packet traveling to its destination</td>
<td>AL2-11</td>
</tr>
</tbody>
</table>

The minimum number of rule determinations is zero, as there may be no rules present, in which the default access level of *NoAccess* will be applied. The maximum number of rule determinations matches the number of rules present in the *VPE Engine*. No signatures are generated, as the *VPE Agent* is not present or able to be contacted, and no signatures will be evaluated. As there is not user query response to evaluate, the effective user will be set to the built-in *unknown* user. The user does not need to authenticate in this scenario.

This case limits the security capabilities of the system. As the user sending a packet cannot be determined, the network visibility of the traffic can not be individualized for the user. However, rules can still be specifically applied to the *unknown* user, or to the machine itself without regard to the user.

These options could be useful for cases where machines without the *VPE Agent* are expected, such as allowing guest’s machines to only access the Internet, or
when using devices that do not support the VPE Agent. Another possibility is assigning MFA credentials to the unknown user and giving said user AuthAccess to a resource. This would result in guests still being allowed to access a resource, so long as the owner of the MFA credentials authorizes them. If these MFA credentials were owned by a network administrator, then the system can still provide both access and security for machines without the VPE Agent.

4.1.6 Other Variations

There are several other possible variations of these execution flows, such as when the multi-factor authentication module is unreachable, or when the message signature doesn’t match. In any of these cases, the execution flow either logs an error and continues the flow as if the source machine had NoAccess to the destination, or the flow of execution halts, which also results in the packet being unable to continue to the destination.

4.2 Component Details

4.2.1 Controller

The controller is written in Python and implements an instance of the Ryu controller [20]. When a packet is received by the controller, it is processed to determine information such as the protocol used, the source address and port, the destination address and port, and, if the packet uses the ICMP protocol, the ICMP ID. The VPE Controller then queries the VPE Agent on the source machine for the username who sent the packet. If unable to reach an agent running on the source
machine, then a built-in unknown user is used. The VPE Controller then sends this information to the VPE Engine, and receives information such as an access level and the rule that was found to apply as a response.

If the access level is NoAccess, then the controller sets a flow rule on the switch to drop packets matching the relevant information. If the access level is FullAccess, then a flow rule set to instead allow matching packets to continue on to their normal path. However, if the access level received is AuthAccess, then the controller will attempt to authorize the connection using the multi-factor authentication details received from the rule engine. If the user authorizes or denies the connection, then the controller will consider the access level to be FullAccess or NoAccess, respectively.

4.2.2 Rule Engine

The VPE Engine, the VPE Agent, and the VPE SwitchAgent components were implemented in Python. The VPE Engine is the program that stores data such as rules, users, and machines, and allows for the processing of rules based on inputs. Any logic regarding which users can contact which endpoints is handled by this component. Rather than use a pre-existing rule engine, the VPE Engine was build specifically for this implementation. While other rule engines, such as Jess [21] and Soar [22], offer increased flexibility to the VPE Engine, they also require the use and understanding of a distinct framework or language. As the VPE Engine is written in plain Python using regular classes and functions, it is simpler to understand and modify. Additionally, even if a preexisting rule engine was used, it would still require a separate component to provide necessary functionality, such as an interface which the user and VPE Controller can access, and the ability to
connect to an MFA system.

A rule is comprised of several parts: including a user, a source, a source port, a destination, a destination port, a level, a priority, and more. Many rule components are able to be grouped together or set to match anything. The full set of rule components are described briefly in Table 4.11, and in more detail in the following paragraphs.

Table 4.11: Components of a Rule

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>the user attempting to establish the connection</td>
</tr>
<tr>
<td>source</td>
<td>the source the user is attempting to connect from</td>
</tr>
<tr>
<td>source port</td>
<td>the source port the user is connecting from. This is primarily useful if a</td>
</tr>
<tr>
<td></td>
<td>service needs to be able to reach a destination</td>
</tr>
<tr>
<td>destination</td>
<td>the destination the user is trying to connect to</td>
</tr>
<tr>
<td>destination port</td>
<td>the destination port the user is trying to connect to</td>
</tr>
<tr>
<td>level</td>
<td>the level of access the user/machine combination should have with the</td>
</tr>
<tr>
<td></td>
<td>destination</td>
</tr>
<tr>
<td>priority</td>
<td>the priority of the rule, where a higher number means increased priority.</td>
</tr>
<tr>
<td></td>
<td>Only a single rule can be acted on for a given connection</td>
</tr>
<tr>
<td>condition</td>
<td>an extra condition (or set of conditions) that must be fulfilled for the</td>
</tr>
<tr>
<td></td>
<td>rule to take effect</td>
</tr>
<tr>
<td>condition combination</td>
<td>the logical operation defining how the conditions must be fulfilled (AND,</td>
</tr>
<tr>
<td></td>
<td>OR, NOT)</td>
</tr>
<tr>
<td>idle timeout</td>
<td>the idle timeout for a rule evaluation in the switch. This timeout</td>
</tr>
<tr>
<td></td>
<td>determines how long a rule can remain unused before the rule evaluation</td>
</tr>
<tr>
<td></td>
<td>is removed</td>
</tr>
<tr>
<td>hard timeout</td>
<td>the hard timeout for a rule evaluation in the switch. This timeout</td>
</tr>
<tr>
<td></td>
<td>determines the maximum lifetime of a rule evaluation in the switch,</td>
</tr>
<tr>
<td></td>
<td>regardless of whether the rule is in use or not</td>
</tr>
</tbody>
</table>

A user refers to an entity using the system. While this generally is used to refer to an actual human being, any Unix username can be used. For example, many services run under a separate user, such as Apache running under *www-data*, and
the root user is used by many system processes. This ability actually allows for more useful rules than many other systems, as these services can have networking rules applied to them just like human beings.

The source refers to the location from which the packet originates. This can be a single machine, or an entire subnet (represented by a network address and netmask).

The destination is similar to the source, in that it can be a single machine or a subnet, except that it refers to the location to which a packet it being sent.

Within the *VPE Engine*, a Port object does not necessarily correspond to a port number. These Port objects refer to a combination of protocol and port number, however the port number can be ‘Any’. This allows for a port to used to represent a protocol by itself, without any regard to actual port number. For example, the builtin *tcp* Port represents all TCP traffic on any port, and the builtin *icmp* Port represents ICMP traffic, which doesn’t have a concept of port numbers.

The source port refers to the protocol and (optionally) the port number from which the packet is being sent. This can be used to allow a specific service to access a resource, especially if the agent is not present on the server or if the service is not run by a unique user. However, in most cases this will remain unset, as most traffic is sent from a random port. This field was added to the system in order to support situations similar to that encountered in Section 5.3.6. During these experiments, the client machine connected to the server’s FTP control port and initiated a file download. However, once the file had been completely transmitted, the *ftp* command failed to exit. This issue arose because the FTP control port’s rule in the switch had timed out after not being used during the download, and the server did not have permission to initiate connections to clients. Thus another rule
had to be created which allowed traffic from the FTP control port on the server to connect to other machines.

The destination port refers to the packet protocol and (optionally) the port number to which the packet is being sent. Currently, the supported protocols are ICMP, TCP, and UDP. This field will be used much more often than the source port, since it is more common to want to protect services from clients, rather than the inverse.

The priority of a rule determines how important a rule is, and can range from 0 to 65535 [23]. Higher-priority rules are checked before lower-priority rules, which allows for more general rules to have specific exceptions, as long as the exceptions have a higher priority. For example, you can have a rule saying that everybody has AuthAccess to a secure server, while having a higher-priority rule that enforces NoAccess to unknown users.

A condition refers to an additional condition that must be satisfied in order for a rule to be applied to a given set of traffic. Currently, the only supported condition is the TimeCondition, which requires that the current time be between a given start and end times. However, additional conditions can be added, such as a planned DateCondition, which will require that the date be between a given start and end date. These can be used to create dynamic rules, such as locking company employees out of a given server after-hours, but providing them access during the company’s work hours. Rules like this may help ensure that malicious entities have even less ability to attack systems at times when system logs or alerts may be less likely to be noticed. Additionally, while many rule components can refer to a simple group, multiple conditions are supported by rules natively.

The condition combination refers to the boolean operations by which a rule’s
conditions must be fulfilled, such as OR, AND, and NOT. While groups can be
used for many of a rule's components, they are functionally restricted to an OR
boolean operation. That is, if a rule's user attribute is set to a group of users, only
one of them must be applicable for the entire user check to apply. However, since
conditions are designed to allow for the creation of dynamic rules, and multiple
conditions are supported for a rule, they have a need for multiple types of boolean
operators. While this is not too useful with the current set of supported conditions,
scenarios can be designed where multiple conditions may all need to be fulfilled
for a rule, or where a rule may require that a given condition is NOT fulfilled.

The idle timeout and hard timeout control the lifetime of a rule evaluation
within the OpenFlow switch. The idle timeout determines how long a rule evalu-
ation can remain unused before it is automatically removed. A value of 0 results
in no idle timeout, which means that a rule being unused will not cause it to be
removed. The hard timeout specifies the maximum lifetime of a rule evaluation
within the switch. This timeout will force the rule to be removed, even if currently
in use. Once again, a value of 0 results in no hard timeout. If an OpenFlow rule
has a 0 for both idle and hard timeout, then the rule will never be automatically
removed.

These timeouts options were added to the definition of a rule because of their
importance to optimizing performance of the VPE System in various situations.
In the case of ICMP traffic, where often only a few packets are being sent at a
time, a low pair of timeouts help maintain increased security. With other types of
traffic, such as traffic utilizing the TCP protocol, there is little need for a short hard
timeout, as TCP is connection-based and will continue to hit the same rule over
an extended period of time. The effects of these timeouts on system performance
in various situations is discussed in Chapter 5.

The VPE Engine itself defines many built-in entities. Some of these builtins are common entities that would often be created by the user, while others are special entities that provide additional functionality. Of the common entities, most are port entities, which have been designed to support common services, such as FTP, SSH, and HTTP. Additionally, there exist built-ins for TCP, UDP, and ICMP traffic, which apply to all traffic utilizing these protocols.

The built-ins that provide special functionality cannot be created manually. These consist of the *any*, *unknown*, and *null* entities. The *any* entity is used for declaring that a given field can apply to any value, essentially allowing it to be ignored. The *unknown* entity allows the user to define rules for users and machines that aren’t described in the VPE Engine or machines that are not running the agent, such as mobile phones and guest machines. The *null* entity is primarily used internally when fields have not been set for an entity, or when a field’s value is deleted.

### 4.2.3 Router

The implemented system is based on the Chaos Calmer version of OpenWrt [24], which was compiled with OpenFlow capabilities using openflow-openwrt [25]. The switching functionality is provided by OpenVSwitch (OVS)[26], which was able to be installed using OpenWrt’s built-in package manager (opkg). Using OVS, a bridge is created that encompasses the interfaces that will be controlled using OpenFlow, and set to connect to a controller running on a defined IP address and port. The commands that can be used to create such a bridge are shown in Section A.2.1. More information about the compilation, installation, and setup of
this customized OpenWrt image can be found in Appendix A.1.

The router also has a running *VPE SwitchAgent* that handles the connections to the controller and machines. While this switch agent is running, it ensures that it has a connection to the controller. If this connection is ever lost, the agent prioritizes reconnecting. When a message arrives from the controller, the switch agent takes the IP address of the target machine and checks that it has an open connection to that machine. If it does not have an open connection, it attempts to create one. If this creation fails, the switch agent responds to the controller with an UnableToConnect message. This failure can be for a number of reasons, such as being blocked by a firewall rule, the target machine not existing, or the agent not running on the target machine. If the switch has or is able to create a connection, then the switch agent forwards the request to the target machine. When the switch agent receives a message from one of its connected machines, the message is immediately forwarded to the controller.

### 4.2.4 Agent

The *VPE Agent* that runs on each host is composed of two parts: a kernel module written in C, and a userspace program written in Python. The kernel module is utilizes netfilter hooks [27] in order to identify packets. The specific hook used is the LOCAL\_OUT hook, which is triggered whenever a packet is sent out of an interface [28]. The packet is then inspected in order to determine the UID of the packet, the protocol used, the source IP address and port, the destination IP address and port, and the ICMP ID. This information is then sent to the userspace process for storage.

Communication between the kernel module and the userspace program is per-
formed through the use of a generic netlink socket [29], which allows for custom messages and attributes to be sent.

ICMP packets are indexed according to their ID, which also corresponds to the process ID (PID) of the process generating the traffic. This is used by the machine to separate sets of ICMP echo requests and replies, such as keeping different runs of the ping command distinct. Additionally, if the ID is not already present in the dictionary, the agent will also store the user who is running the process.

TCP and UDP packets are indexed according to their source port, which is expected to be unique for a given time period. If a source port is ever re-used, then the index in the dictionary is simply updated with the new packets.

When the agent receives a request for information from its switch, the agent looks up the information based on the details of the query, and then sends the username that generated the traffic back to the requester, along with a copy of a request and the signature.

The agent authenticates itself with the Rule Engine using rotating RSA keys. These keys are generated on startup, recorded to a file, and sent to the rule engine in the response to the first received user query. In each response to a user query, the agent includes the signature of the response, which is generated from its private key. When the rule engine receives this response and validating it, the signature is checked to determine the trustworthiness of the agent. If the signature does not match, then the user is denied access. This checking is possible since the rule engine maintains a knowledge of each agent’s public key.

On startup, the agent checks to see if it has an old set of RSA keys, and if so, loads them. The agent then generates a new set of keys to use for the current session. The first time the agent receives a user request from the controller, the

67
agent includes information about the key change. This information includes a flag that indicates, a key change is occurring, as well as the new public key to use. For this one request, the old private key is used to generate the response signature. This allows the rule engine to validate the key change. Since the old private key was trusted, and that same key was used to sign the key change message, the new keypair should also be trusted. This key rotation is important, as it allows keys to be recycled often and automatically. If an attacker manages to gain control of the keypair, then it will be invalidated if the agent reconnects to the server. Any attempt to use the old key will result in an error in the logs. If the attacker instead sends a key change message, then the agent’s responses will cease to be validated, which will quickly raise suspicion. These will also be logged as attempted uses of the old key. This process is also described visually in Figure 4.30.

If the agent does not have a saved set of keys to support a key change, then instead of adding a key change flag to the first response, it instead adds a new key flag. Currently, this simply results in an error being logged, but in the future this will result in a message being sent to a system administrator. More information about this future work can be found in Section 6.2.13.

4.2.5 Interoperability with Other Programs

Currently, usernames are represented by Unix-style usernames. However, these usernames can be created as local accounts. Ideally, this system would run alongside another system to provide login credentials, such as LDAP. This other system would be used to limit the creation of arbitrary users. However, even if a username is spoofed in order to impersonate a user with access to secure systems, any Level-1 accesses would still require the user in question to validate the connection.
Figure 4.30: Signature verification process
using multi-factor identification. Additionally, strict rules could be set to limit the access to a certain machine or set of machines.

This system does not contain any facilities to provide Dynamic Host Configuration Protocol (DHCP) or Domain Name System (DNS) services, so the system would in most cases run alongside separate DHCP and DNS servers. As long as rules are set properly to allow the DHCP and DNS traffic, the system should cause no issues. In the future, the VPE System may support additional services, as discussed in Section 6.2.2.

4.2.6 Signature Validation

At a high-level, there are seven possible results when validating a message. These consist of a successful validation, a successful validation with a key change, a new agent, and four incorrect validations. Pseudo-code that corresponds to the validation process within the Rule Engine is located in Listing 4.1.
Listing 4.1: Pseudocode for validating response keys

1. Pubkey recognized
   - Signature doesn’t match -> Error "Incorrect signature for recognized key"
   - Signature matches -> valid
2. Pubkey unrecognized
   - Not reporting keychange -> Error "Unrecognized key"
   - Reporting keychange
     - Old key is empty -> New agent
     - Old key exists
       - Old key not recognized -> Error "Attempted keychange with unrecognized original key"
       - Old key recognized
         - Signature doesn’t match -> Error "Attempted keychange with invalid signature"
         - Signature matches -> Key change successful
Chapter 5

Experimentation

Many experiments were run on the system in a basic network configuration. In addition to testing the capabilities of the system, these experiments were also used to collect data on the efficiency of the system.

Unless specified otherwise, all switches in the networks were running OpenWrt as built in Section A.1, and all other machines are running Ubuntu 18.04.1.

5.1 Experiment Network Layout

The experiment network consists of 4 virtual machines. These machines are Machine-1, Machine-2, a switch, and a controller. This is shown visually in Figure 5.1.

Machine-1 will have the VPE Agent installed and running for the tests involving the system. During the standard network and simple SDN tests, the agent will be deactivated. Machine-1 will send requests to the other machines on the network, mainly Machine-2.
The switch agent will be present on the switch, along with an OpenVSwitch bridge running OpenFlow and connected to the controller.

The controller will be running the Rule Engine, the Ryu/System controller, and the multi-factor authentication module. While these all are able to be located on separate machines, in most cases it would be ideal to have the controller and the rule engine present on the same machine. This allows for reduced latency of the system, as local connections are much faster than external connections. This is also expected to be one of the standard layouts for this system, as most networks will only require a single controller and rule engine.

5.1.1 Network Configurations

In these experiments, the switch has 5 network interfaces attached: A ”NAT Network” interface for connecting to the Internet, 3 internal network interfaces on
test-net-1, test-net-2, and test-net-3, and a host-only interface for communicating with the host machine. The NAT and internal networks are all set to promiscuous mode "Allow VMs", and are bridged together with OpenVSwitch. This bridge allows the switch and each of the attached networks to reach each other and the Internet. The host-only network allows the host machine to communicate with the switch without experiencing interference from the running system.

Each of the other machines, Machine-1, Machine-2, and Controller, have two attached network interfaces. These consist of a single internal network, test-net-1, test-net-2, and test-net-3 respectively, as well as a host-only interface. Similarly to the switch, the internal networks are used for traffic between each other and the Internet, while the host-only network supports communication with the host machine. However, each of the interfaces on these machines are set to promiscuous mode "None", as these machines do not require the capability of monitoring all traffic on their interfaces.

### 5.1.2 VPE Engine Configuration

The *VPE Engine* was configured with several users, networks, ports, and rules in order to support the experiment configurations. The users defined in in the experiment are summarized in Table 5.1, the ports used are summarized in Table 5.2, the networks defined are summarized in Table 5.3, and the rules set are defined in Table 5.4. The *Any* object was used in many fields to allow any value to apply, and are designated by table cells containing the word "Any".

Each of the ports used is built-in to the *VPE Engine*, except for the FTP passive port used (ftp_p).
<table>
<thead>
<tr>
<th>Name</th>
<th>Username</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserA</td>
<td>usera</td>
<td>Alice</td>
</tr>
<tr>
<td>UserB</td>
<td>userb</td>
<td>Bob</td>
</tr>
<tr>
<td>UserE</td>
<td>usere</td>
<td>Eve</td>
</tr>
<tr>
<td>Resolve</td>
<td>systemd-resolve</td>
<td>Systemd Resolver</td>
</tr>
</tbody>
</table>

Table 5.2: Relevant experimentation Ports

<table>
<thead>
<tr>
<th>Name</th>
<th>Protocol</th>
<th>Port Number</th>
<th>Builtin</th>
</tr>
</thead>
<tbody>
<tr>
<td>icmp</td>
<td>ICMP</td>
<td>Any</td>
<td>Yes</td>
</tr>
<tr>
<td>http</td>
<td>TCP</td>
<td>80</td>
<td>Yes</td>
</tr>
<tr>
<td>ssh</td>
<td>TCP</td>
<td>22</td>
<td>Yes</td>
</tr>
<tr>
<td>dns</td>
<td>UDP</td>
<td>53</td>
<td>Yes</td>
</tr>
<tr>
<td>ftp_c</td>
<td>TCP</td>
<td>21</td>
<td>Yes</td>
</tr>
<tr>
<td>ftp_p</td>
<td>TCP</td>
<td>10100</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.3: Defined experimentation Networks

<table>
<thead>
<tr>
<th>Name</th>
<th>CIDR</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>lan</td>
<td>10.0.2.0/24</td>
<td>The local experimentation network</td>
</tr>
<tr>
<td>github</td>
<td>github_CIDR</td>
<td>The IP range used by GitHub during experimentation</td>
</tr>
<tr>
<td>internet</td>
<td>0.0.0.0/0</td>
<td>The entire possible IP range</td>
</tr>
<tr>
<td>#</td>
<td>User</td>
<td>Access</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>UserA</td>
<td>FullAccess</td>
</tr>
<tr>
<td>2</td>
<td>UserB</td>
<td>FullAccess</td>
</tr>
<tr>
<td>3</td>
<td>UserB</td>
<td>FullAccess</td>
</tr>
<tr>
<td>4</td>
<td>UserB</td>
<td>FullAccess</td>
</tr>
<tr>
<td>5</td>
<td>UserB</td>
<td>NoAccess</td>
</tr>
<tr>
<td>6</td>
<td>UserB</td>
<td>NoAccess</td>
</tr>
<tr>
<td>7</td>
<td>UserB</td>
<td>FullAccess</td>
</tr>
<tr>
<td>8</td>
<td>Resolve</td>
<td>FullAccess</td>
</tr>
<tr>
<td>9</td>
<td>Any</td>
<td>FullAccess</td>
</tr>
</tbody>
</table>
5.1.3 Experiment Scenarios

Ideally the VPE System would have such a small effect on the network that the time for a packet with FullAccess to traverse the network would be indistinguishable from a similar packet traversing a standard network with no security mechanisms. However, this is unrealistic, as the VPE System will inevitably cause packets to take more time than they would otherwise.

The next best result would involve a packet with FullAccess being indistinguishable from a packet passing through a basic implementation of the SDN which the VPE System is based on.

Additionally, it would be useful to determine the effect of the VPE System when a packet does not have access to the destination, and when a series of requests are performed where some packets have access and others do not.

In order to collect data on each of these situations, each experiment type was run with the following scenarios:

1. Standard Network: A standard network not running any security mechanisms
2. Simple SDN: A simple SDN approving all flows with a timeout of 1 second
3. FullAccess: A network running VPE System with a user given FullAccess to all endpoints
4. NoAccess: A network running VPE System with a user given NoAccess to all endpoints
5. Mixed Access: A network running VPE System with a user given FullAccess to some endpoints and NoAccess to others
No experiments were run with AuthAccess, as the speed of the multi-factor authentication would vary greatly across individual trials. The results of such tests would be approximately equal to the tests with FullAccess and NoAccess, with the additional time loss required by the user authenticating.

5.2 Statistical Analysis

In order to make accurate comparisons of the data produced by each scenario, certain statistical analyses were performed.

The data for each experiment type was checked for normality using the Shapiro-Wilk test. For these tests, the alpha value used was 0.05, the null hypothesis was that the data came from a population that had a normal distribution, and the alternative hypothesis was that the data came from a population that did not have a normal distribution. The null hypothesis was rejected in nearly all cases, thus demonstrating that the data did not match a normal distribution. Therefore the data would be considered non-parametric [30]. The only case in which the test failed to reject the null hypothesis was for the Mixed Access scenario of the experiment described in Section 5.3.6.

As well as being non-parametric, the data samples were also considered independent. Though the same machines were used for each data measurement, the individual data values generally did not affect each other. Each trial sent approximately the same packets, and the timeouts in the OpenFlow router were tailored to ensure that trials were not contaminated by previous trials.

Given these properties of the data, the Kruskal-Wallis was chosen to determine if a significant difference could be found for each experiment. If such a difference
was found, a post-hoc test was performed between each pair of scenarios in order to determine which of the scenarios were significantly different. For the same reasons the Kruskal-Wallis test was used, the chosen post-hoc test was the Mann-Whitney test with continuity correction [30].

For both the Kruskal-Wallis and Mann-Whitney tests, the null hypothesis was that the samples came from populations with equal medians, and the alternative hypothesis was that the samples came from populations with medians that are not all equal. The alpha value used for the Kruskal-Wallis test was 0.05, however a difference alpha value had to be used for the Mann-Whitney tests. Since up to 10 post-hoc tests were run on the data for each experiment, the alpha value used must be adjusted in order to account for the accumulating Type I error rate. This adjustment method used was a Bonferroni correction [30], which was performed by dividing a total alpha value of 0.05 by the number of post-hoc tests performed, and differs per experiment.

The results of the Kruskal-Wallis test and (if performed) the Mann-Whitney post-hoc tests can be found in each experiment’s section.

5.3 Experiments

5.3.1 Enforcing Visibility of other Machines

As a security mechanism, the VPE System is designed to prevent unauthorized entities from locating machines that they are not intended to access. One way of locating machines on a network is by sending a packet to all of the valid IP addresses on a network, such as a TCP SYN packet. This type of scan is often called a ”horizontal” network scan.
In order to test the system’s capability to block this type of traffic, such a scan was run 100 times. This scan was performed on Machine-1 with the nmap[31] tool, searching for other machines on the network. The command used to perform the scan is shown in Listing 5.1.

Listing 5.1: Horizontal scan command

```
nmap -n -sn -PS21 10.0.2.0/24
```

The `-n` flag was used to disable reverse DNS lookup on the hosts, while the `-sn` flag was used to disable port scanning of discovered hosts. The `-PS21` flag instructed nmap to use a TCP SYN packet sent to port 21 on each host. This port was chosen because the rules defined in the experiments allowed UserB to reach this port on Machine-2, thus allowing discovery of two hosts during the Mixed Access experiment. This option was also set in order to disable nmap’s host discovery.
using Address Resolution Protocol (ARP) packets. As these packets are unable to be captured by this version of the VPE KernelModule, user-based rules would fail to differentiate these packets.

In each of these scans, additional addresses were detected than discussed below, consisting of the VirtualBox DHCP server, the host machine, and the switch. However, as these machines are not considered part of the experiment and traffic involving each of them has been explicitly allowed, they have been filtered out of the results.

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (s)</th>
<th>StdDev (s)</th>
<th>Machines</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>1825.62</td>
<td>533.3272</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>2023.75</td>
<td>711.3207</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>FullAccess</strong></td>
<td>100</td>
<td>1788.12</td>
<td>514.7943</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>NoAccess</strong></td>
<td>100</td>
<td>1732.60</td>
<td>279.9313</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>100</td>
<td>1816.90</td>
<td>379.7124</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Each of the experiments consisted of 100 trials. The experiment with the longest average time to complete was the Simple SDN experiment, with an average time of 2.0 seconds. Each of the other experiments had more comparable times, between 1.7 and 1.8 seconds each.

The **FullAccess** experiment was able to detect all machines on the network, consisting of Machine-1, Machine-2, and the controller, since rule 1 allowed UserA to access every port on every machine.

The Mixed Access experiment was able to detect only two machines on the network, Machine-1 and Machine-2. Machine-1 was detected as that was the machine the nmap command was being sent from, and Machine-2 was able to be detected
because rule 3 allowed the user involved to detect the scanned port on Machine-2. The NoAccess experiment was only able to discover Machine-1, as the user involved was not permitted to send packets to any other machine on the network.

The Kruskal-Wallis test performed on the data reported a p-value of 0.145, which is not significant with the alpha-value of 0.05. Therefore, we failed to reject the null hypothesis that the data came from populations with equal medians, and no post-hoc tests were performed.

5.3.2 Enforcing Traffic Based On Port

Several common protocols, such as TCP and UDP, use a specific "port" for communication. For example, TCP port 80 is the standard port used for unencrypted web traffic, while UDP port 53 is generally used for DNS requests. It is important for the VPE System to be able to filter traffic based on port, as an entity may be granted access to some ports but not others, while still being able to send ICMP requests to the machine.

In order to test whether a user can access a port, a "vertical" network scan can be used. This type of scan attempts to connect to multiple ports on a machine to see which ones can be reached. As there are 65535 possible ports in the TCP and UDP protocols, a vertical scan on all of these ports were performed. However, as the first 1024 ports are most often used for system services, a scan of only these ports is most common. As such, a scan on ports 1-1024 was also performed.

This type of test is also useful for testing the VPE System while under stress, as each scanned port will require a separate computation from the VPE System.

Machine-1 performed both port scans of Machine-2. With FullAccess, the user should see multiple open ports on Machine-2, but these should appear closed when
Figure 5.3: The partial vertical scan experiment

Figure 5.4: The full vertical scan experiment
the user has NoAccess to those ports. The command used to perform the partial and full vertical scans are shown in Listings 5.2 and 5.3, respectively.

Listing 5.2: Partial vertical scan command

```bash
nmap ${Machine_2_IP} -n -Pn -p 1-1024 --max-retries 0
```

Listing 5.3: Full vertical scan command

```bash
nmap ${Machine_2_IP} -n -Pn -p 1-65535 --max-retries 0 \ --max-rtt-timeout 200ms
```

In each command, the -n flag was used to disable reverse DNS resolution of the target IP address, as this would cause several seconds of additional latency to start the scans.

The -Pn flag was used to disable host discovery, as the some of users used in the experiments did not have access to the required endpoint. Without this flag, nmap may refuse to scan such machines, as it is unable to verify their reachability.

The --max-retries 0 option was set to ensure that a consistent number of packets was sent during each trial.

In the full vertical scan, the --max-rtt-timeout 200ms option was used to limit the maximum RTT timeout. This option was used to limit how long each trial could take. This was necessary because of the length of time the NoAccess scan would take to timeout on every port.
Table 5.6: Partial vertical scan results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (s)</th>
<th>StdDev (s)</th>
<th>Open Ports</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>0.091</td>
<td>0.029</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>2.617</td>
<td>0.925</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>100</td>
<td>6.613</td>
<td>1.038</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>100</td>
<td>104.436</td>
<td>0.098</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>100</td>
<td>9.381</td>
<td>2.256</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The Standard Network experiment was able to complete each trial with an average time of 0.09 seconds. The Simple SDN experiment involved the switch querying the controller for every single port, and this resulted in a large time increase to 2.62 seconds. The *FullAccess* experiment also resulted in a large time increase, up to 6.61 seconds. This is expected, as every single request required a pass through the VPE System to make a rule determination. This time could be improved through various techniques, such as those discussed in Sections 6.2.7 and 6.2.9.

The *NoAccess* experiment was unable to detect any open ports. However, this took a much longer time than any of the other experiments because nmap was unable to use successful connection times to adjust the timeouts. Thus the full timeout was used for every port, with a limited number of parallel attempts.

The Mixed Access experiment involved the user only having access to the FTP port, and so only this single port was detected. However, this experiment took a longer average time to complete, as nmap had a limited amount of information to use to adjust the scan variables.
Table 5.7: Partial vertical scan significance

<table>
<thead>
<tr>
<th></th>
<th>Simple SDN</th>
<th>FullAccess</th>
<th>NoAccess</th>
<th>Mixed Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>FullAccess</td>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>NoAccess</td>
<td></td>
<td></td>
<td></td>
<td>0.000</td>
</tr>
</tbody>
</table>

The Kruskal-Wallis test performed on the partial vertical scan data reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. The results of these post-hoc tests can be found in Table 5.7. All of the tests performed reported p-values that were found to be significant at the alpha value used of 0.005.

Table 5.8: Full vertical scan results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (s)</th>
<th>StdDev (s)</th>
<th>Open Ports</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>2.639</td>
<td>0.587</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>30</td>
<td>225.544</td>
<td>101.041</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>30</td>
<td>306.581</td>
<td>89.692</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>30</td>
<td>613.553</td>
<td>213.013</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>38</td>
<td>649.089</td>
<td>236.467</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The Full Vertical Scan experiments had similar results to the Partial Vertical Scan experiments, though more extreme. Both the Simple SDN and the FullAccess experiments took over 100 times as long to complete on average than the Standard Network Experiment.
The Kruskal-Wallis test performed on the full vertical scan data reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. The results of these post-hoc tests can be found in Table 5.9. Nearly all of the tests performed reported p-values that were found to be significant at the alpha value used of 0.005, with the only exception being the test between the NoAccess and Mixed Access scenarios, which was not found to be significant. This is expected, as the rules applied to the Mixed Access scenario resulted in approximately the same access levels as during the NoAccess scenario.

### 5.3.3 Ping Requests

It is important to know the latency of packets sent between machine. Ideally, a security mechanism should have as small of an impact on this latency as possible. Additionally, it is useful to be able to send only a single packet at a time.

ICMP echo and reply messages can be used for this purpose, and the `ping` utility can be used to send and receive these messages, along with recording the latency between sending each packet and receiving each reply.

Ping requests were sent by Machine-1 to Machine-2. The command used to
send the ICMP requests is shown in Listing 5.4.

Listing 5.4: Ping experiment command

```
ping -c 100 -i 1.5 ${Machine_2_IP}
```

The -i 1.5 option set the time interval between individual ping requests to 1.5 seconds. As the timeout for the rules set in the switch were equal to 1 second, this interval ensured that each request would be sent only after the previous request’s rule evaluation had expired. Thus, every single request required a full re-evaluation by the VPE System.
Table 5.10: Ping Results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg RTT (ms)</th>
<th>StdDev (ms)</th>
<th>Replies</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>0.8634</td>
<td>0.5152</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>3.6011</td>
<td>0.4245</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>100</td>
<td>7.0037</td>
<td>1.7156</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>100</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>100</td>
<td>8.464</td>
<td>0.7528</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>

In each experiment, the number of trials was set to 100. However, unlike tools used in other experiment types, the ping tool is able to send a specific number of requests and aggregate the results. Thus a single usage of the ping command per experiment was sufficient to generate 100 trials.

The standard network experiment had an average Return-Trip Time (RTT) of 0.86 milliseconds. The Simple SDN experiment had a significantly longer average RTT of 3.6 ms, which is 4.17 times as long. The FullAccess experiment had an average RTT of 7 ms, which is 1.94 times as long as the Simple SDN experiment. However, this also shows that the effect of using an SDN itself was about the same as the effect caused by VPE System.

During the NoAccess experiment, the user was unable to reach Machine-2. Thus there is effectively no RTT, as no replies were ever received.

The Mixed Access experiment involved the user having ICMP access to Machine-2, and thus the effective permissions for this experiment matched those in the FullAccess experiment.
As shown in 5.11, the significance with the \textit{NoAccess} scenario cannot be calculated, as this scenario had no return trip time.

The Kruskal-Wallis test performed on the ping experiment data reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. However the \textit{NoAccess} scenario was ignored, as the significance cannot be calculated due the lack of a return trip time that results in no data. The results of these post-hoc tests can be found in Table 5.11. All of the tests performed reported p-values that were found to be significant at the alpha value used of 0.0083.

\begin{table}[h]
\begin{center}
\begin{tabular}{|l|c|c|c|}
\hline
 & Simple SDN & \textit{FullAccess} & \textit{NoAccess} \\
\hline
\textbf{Standard Network} & 0.000 & 0.000 & N/A \\
\hline
\textbf{Simple SDN} & 0.000 & N/A & 0.000 \\
\hline
\textbf{\textit{FullAccess}} & & N/A & 0.000 \\
\hline
\textbf{\textit{NoAccess}} & & & N/A \\
\hline
\end{tabular}
\end{center}
\end{table}

### 5.3.4 Downloading a Web Page Locally

Web pages are one of the most common types of files sent over a network. However, the internet is subject to many factors that cannot be controlled for. Therefore, an experiment was run on the ability of the system to allow or block web page downloads from inside the network. This partially mitigates the issues with timing page requests that leave the local network.

A web page from an Apache [32] instance were downloaded to Machine-1 to Machine-2. The command used to request this page is shown in Listing 5.5.
Figure 5.6: The internal web page download experiment

Listing 5.5: Internal web page download experiment command

```bash
curl ${Machine_2_IP} | wc -c
```

The `curl` command was passed through `wc -c` in order to count the number of bytes received.

Table 5.12: Internal web traffic results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (ms)</th>
<th>StdDev (ms)</th>
<th>Bytes</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>15.07</td>
<td>3.3492</td>
<td>10918</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>19.67</td>
<td>4.4858</td>
<td>10918</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>100</td>
<td>21.89</td>
<td>5.9914</td>
<td>10918</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>100</td>
<td>130138.30</td>
<td>96.9206</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>100</td>
<td>130142.84</td>
<td>92.1253</td>
<td>0</td>
<td>None</td>
</tr>
</tbody>
</table>

Each of the experiments was run with 100 trials. The standard network experiment took an average time of 15.1 milliseconds, while the FullAccess experiment took a much longer time.
took an average of 21.9 milliseconds. The Simple SDN experiment took an average time of 19.7 milliseconds. Each of these experiments had a fairly large standard deviation, ranging between 22.2% of the average (for the standard network experiment) and 27.4% of the average (for the FullAccess experiment). The similarity of these experiments demonstrates that the VPE System had only a small effect on the speed of these web page downloads.

Both the NoAccess experiment and the Mixed Access experiment took approximately the same amount of time, 130.1 seconds per trial. This time corresponds to the timeout of curl when it is unable to reach the destination. The Mixed Access experiment involved the user having NoAccess access to Machine-2's HTTP port, and thus the effective permissions were the same as in the NoAccess experiment. This is also why the number of bytes downloaded in each of these experiments was 0.

Table 5.13: Internal web traffic significance

<table>
<thead>
<tr>
<th></th>
<th>Simple SDN</th>
<th>FullAccess</th>
<th>NoAccess</th>
<th>Mixed Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>0.021</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>FullAccess</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>NoAccess</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The Kruskal-Wallis test performed on the internal web traffic experiment reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. The results of these post-hoc tests can be found in Table 5.13. Nearly all of the tests performed reported p-values that were found to be significant at the alpha value used of 0.005,
with the only exception being the test between the FullAccess and Simple SDN scenarios, which was not found to be significant.

5.3.5 Downloading Web Pages from the Internet

Much of common network traffic consists of web page downloads from the Internet. Therefore, security mechanisms such as VPE System should have as little interference on this traffic as possible. If the system has an extremely noticeable effect on this type of traffic, it will be unlikely to be utilized fully, so the time to complete these requests is important. Additionally, such an experiment also results in a test of the system’s capabilities of enforcing policy using multiple traffic types in a short amount of time, as each web page request also requires a DNS request.

![Diagram](image)

Figure 5.7: The external web page download experiment

Web pages from various internet sites were downloaded to Machine-1. The
command used to request these web pages is shown in Listing 5.6.

Listing 5.6: External web page download experiment command

curl https://www.google.com \
    https://www.wikipedia.org \
    https://www.yahoo.com \
    https://stackoverflow.com \
    https://github.com \
    | wc -c

The curl command was passed through wc -c in order to count the number of bytes received.

Table 5.14: External web traffic results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (ms)</th>
<th>StdDev (ms)</th>
<th>Bytes</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>2900.22</td>
<td>877.4741</td>
<td>925000</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>3087.01</td>
<td>832.8603</td>
<td>925000</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>100</td>
<td>3222.66</td>
<td>768.7961</td>
<td>925000</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>100</td>
<td>52772.39</td>
<td>100.2269</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>50</td>
<td>217369.40</td>
<td>608.7745</td>
<td>870325</td>
<td>5, 6, 7, 8</td>
</tr>
</tbody>
</table>

Each of the experiments, besides the Mixed Access experiment, were run 100 times. Where the standard network took an average of 2.9 seconds to resolve and download all 5 web pages, the FullAccess experiment took an average of 3.2 seconds to do the same. This is a small difference, and demonstrates that the VPE System can have a minimal effect on Internet usage. The experiment involving a simple SDN had a similar average time of 3.1 seconds.

The NoAccess experiment took approximately 52.8 seconds for each trial. This was caused by the DNS lookups for each of the 5 web sites timing out after approximately 10 seconds, as the user running the requests had NoAccess access.
to everything, including the main DNS provider. This actually took less time to complete than the Mixed Access test, which took an average of 217.4 seconds to complete each trial. As the Mixed Access experiment was unable to reach GitHub, the trials could not complete until the download timed out.

The amount of bytes downloaded during each trial was variable, as certain sites, such as Google, returned slightly different web pages with each request. The Mixed Access experiment downloaded significantly fewer bytes than the other successful experiments, as this experiment involved being blocked from reaching one of the websites.

Table 5.15: External web traffic significance

<table>
<thead>
<tr>
<th></th>
<th>Simple SDN</th>
<th>FullAccess</th>
<th>NoAccess</th>
<th>Mixed Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>0.003</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Simple SDN</td>
<td></td>
<td>0.085</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>FullAccess</td>
<td></td>
<td></td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>NoAccess</td>
<td></td>
<td></td>
<td></td>
<td>0.000</td>
</tr>
</tbody>
</table>

The Kruskal-Wallis test performed on the external web traffic experiment reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. The results of these post-hoc tests can be found in Table 5.15. Nearly all of the tests performed reported p-values that were found to be significant at the alpha value used of 0.005, with the only exception being the test between the FullAccess and Simple SDN scenarios, which was not found to be significant. This is similar to the significance results of the internal web traffic test.
5.3.6 Supporting Long-Running Connections

While less common than web page requests, a significant portion of network traffic consists of large file downloads. These downloads are often performed by file-sharing programs or during program installation. Thus such activities are important to the daily activities of network traffic. The VPE System should allow for such downloads while causing minimal interference, and thus an experiment measuring the effect of the system on these downloads was performed.

Figure 5.8: The FTP download experiment

Included in this experiment is a test of the ability of the VPE System to cache rule results to avoid checking every single packet in a long-running connection.

Files were downloaded from Machine-2 to Machine-1 over file-transfer protocol (FTP). The command used to download the files is shown in Listing 5.7.
Listing 5.7: FTP experiment command

```bash
ftp -n -p ${Machine_2_IP} << EOF
user anonymous
get ubuntu-18.04.1-desktop-amd64.iso
quit
EOF
```

The `-n` flag was used to disable automatic login attempts on connection to the server, so that the anonymous user could be more easily used.

The `-p` flag was used to enable FTP’s passive connection mode, which instructs the client machine open a connection to the server’s data port, instead of the server initiating this second connection. This is a common mode used to bypass firewalls, as the server does not require permission to initiate connections with the client.

The `ubuntu-18.04.1-desktop-amd64.iso` disk image was used as it is a 1.9 GB file that is freely available, and it is similar to the disk images used to install the experiment machines.

Table 5.16: File transfer results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Trials</th>
<th>Avg Time (s)</th>
<th>StdDev (s)</th>
<th>Bytes</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>100</td>
<td>19.100</td>
<td>1.250</td>
<td>1953349632</td>
<td>N/A</td>
</tr>
<tr>
<td>Simple SDN</td>
<td>100</td>
<td>21.860</td>
<td>2.822</td>
<td>1953349632</td>
<td>N/A</td>
</tr>
<tr>
<td>FullAccess</td>
<td>100</td>
<td>22.159</td>
<td>4.631</td>
<td>1953349632</td>
<td>1</td>
</tr>
<tr>
<td>NoAccess</td>
<td>100</td>
<td>10.009</td>
<td>0.018</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Mixed Access</td>
<td>100</td>
<td>18.334</td>
<td>0.345</td>
<td>1953349632</td>
<td>3, 4, 9</td>
</tr>
</tbody>
</table>

Each of the experiments were run 100 times, and each (besides the `NoAccess` experiment) took approximately the same amount of time. This demonstrates the ability of the VPE System to avoid interfering with long-running connections, such as the download of large files. This is primarily accomplished through the use of
long timeouts, which allow each trial to only require a single set of authorizations at the start.

The NoAccess experiment took an average of 10 seconds for each trial, as this was the timeout used by the `ftp` command. Since the user did not have access to reach the FTP server, no bytes were transmitted.

The Mixed Access experiment involved the user having access to the FTP server’s ports, provided by rules 3, 4, and 9, and thus the effective permissions were approximately the same as during the FullAccess experiment.

Table 5.17: File transfer download significance

<table>
<thead>
<tr>
<th></th>
<th>Simple SDN</th>
<th>FullAccess</th>
<th>NoAccess</th>
<th>Mixed Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Network</td>
<td>0.000</td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Simple SDN</td>
<td></td>
<td>0.085</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>FullAccess</td>
<td></td>
<td></td>
<td>0.000</td>
<td>0.010</td>
</tr>
<tr>
<td>NoAccess</td>
<td></td>
<td></td>
<td></td>
<td>0.000</td>
</tr>
</tbody>
</table>

The Kruskal-Wallis test performed on the external web traffic experiment reported a p-value of 0.000, which is significant with the alpha-value of 0.05. Therefore, we reject the null hypothesis that the data came from populations with equal medians, and performed post-hoc tests on each pair of scenarios. The results of these post-hoc tests can be found in Table 5.17. Many of the tests performed reported p-values that were found to be significant at the alpha value used of 0.005, but three of the tests resulted in non-significant p-values. These consist of the comparisons between the FullAccess scenario and the Standard Network, Simple SDN, and Mixed Access scenarios. This is unsurprising, as the average times for each of these four scenarios are very similar.
Chapter 6

Discussion and Conclusion

6.1 Difficulties Identified

6.1.1 ICMP Traffic

ICMP packets do not utilize certain mechanisms present in some other protocols, such as ports in TCP and UDP traffic. Without these port attributes, it is difficult to filter ICMP traffic in a similar manner to TCP and UDP traffic, as the source port is used to differentiate streams of data with OpenFlow.

While ICMP traffic does not utilize ports, it does have an ID field. This field is unique to a ICMP sequence, and can be used to differentiate these sequences. However, OpenFlow does not have the ability to set rules based on this field, so the VPE System itself must keep track of this field. This also means that rule results cannot be cached at the switch the same way as TCP and UDP traffic, since caching these results would not allow for individualized ICMP reachability based on user.
6.1.2 ARP Traffic

ARP traffic is, in general, a separate ethernet protocol from TCP/IP. Therefore, the netfilter hooks used in the VPE KernelModule are unable to capture ARP traffic. This means that different hooks must be used, and that ARP traffic must be processed completely separately from TCP/IP traffic. In the future, the VPE KernelModule will be able to capture more types of traffic, or will actually consist of multiple separate kernel modules.

6.1.3 Inter-Component Traffic

Openvswitch automatically sets OpenFlow rules to allow traffic between the itself and the OpenFlow controller. However, this does not apply to the traffic between VPE System components. Therefore, the VPE System must define separate rules to allow this traffic, potentially without the use of the VPE Engine. The VPE Engine could be used as long as the VPE Engine and VPE Controller are present on the same machine, as this traffic will be local to the machine and avoid the SDN. If instead the VPE Engine and VPE Controller are on separate systems, and the traffic between them travels through the SDN, then the VPE Controller will be unable to communicate with the VPE Engine to determine an access level.

In the current implementation, the rules must be manually set to allow for this inter-component traffic, either through the VPE Engine or the OpenFlow rules on the switch. However, in the future the VPE System itself could maintain knowledge of the infrastructure and component locations, in order to automatically create and apply rules. This infrastructure knowledge could be built into the VPE Engine as components similar to the existing machines, users, and ports, or as a
separate component in the VPE Controller that generates rules on startup.

6.1.4 Broadcast Traffic

The TCP/IP stack provides a method of broadcasting packets to all machines on the network. This is known as broadcast traffic, and is performed by sending a packet to a certain IP address, known as a broadcast address, that is sent to all machines on a network.

The current implementation of the VPE System treats traffic to the broadcast address in the same manner as other traffic, where the ability of a user to send data to the broadcast address is assigned an access level. However, it would be useful for the VPE System to allow for more fine-grained control of broadcast traffic. For example, a user could be allowed to send broadcast traffic, but that traffic could be forwarded only to destinations to which the user has access. This could be performed by manually rewriting packets to have a destination matching individual machines, but this would likely have a significant effect on the efficiency of the VPE System. Future research would have to be performed to determine the best method of handling such traffic in the future.

6.2 Additional Work

6.2.1 Use of Other SDN Protocols

The OpenFlow SDN protocol and the Ryu controller were selected to be used in this iteration of the VPE System. However, these technologies do have limitations that make it more difficult for the VPE System to perform actions that would
fully fulfill its goals.

Once of these limitations is OpenFlow’s inability to filter packets based on ICMP IDs [23]. This limitation and the difficulties it causes are discussed in more detail in Section 6.1.1. Similarly, OpenFlow is unable to perform a match on a set of port numbers, increasing the difficulty with implementing functionality such as the rule aggregation described in Section 6.2.9.

Another limitation lies in Ryu’s speed and scalability. According to [33], the Ryu controller had no scalability across CPU cores, as well as a low flowrate overall. While this study did determine that Ryu was more secure than the other controllers tested, the inefficiencies of Ryu can be clearly seen in the experiments described in Sections 5.3.1 and 5.3.2.

In order to mitigate these limitations, future iterations of the VPE System will be able to utilize different SDN protocols, and may also be able to use a custom protocol or underlying network filter tool designed for the VPE System.

6.2.2 Inclusion of Other Network Tools

Many systems are present on a normal network, such as a DHCP server and a DNS server. Currently, the VPE System does not possess the ability to perform either of those functions, or any other capabilities, such as building routing tables. Because of this, these systems must be present on the network separately from the VPE System, and the VPE System must be configured to allow these systems to communicate with machines on the network.

It would be useful if the VPE System had the capability to integrate services that provide these functions, or if it was able to provide these functions directly. This would allow for increased synthesis between the entities defined in the VPE
and the state of the network itself. For example, an integrated DHCP server could be instructed to always provide the IP address set in a machine’s definition to that machine. Additionally, rules to allow traffic from integrated services could be automatically generated and applied, increasing the convenience and usability of the system, while not compromising security.

6.2.3 Merging Components

The VPE System is designed to allow for the various components to be located on separate machines on the network. However, this increases the number of messages that must be sent for a rule evaluation to complete, introducing latency into the system. In order to minimize the time taken to perform a rule evaluation, several components of the system could be merged into a single application. The main components that could be merged are the VPE SwitchAgent, VPE Controller, and the VPE Engine, as most of the messages sent within the VPE System are between these three components. If additional components are added to the system, such as a caching module, then they would also be candidates for inclusion in this combined system.

While this would increase the speed of the rule evaluations, such a system would most likely live on the switch itself, which may then require the switch to have more available resources than a simple switch generally possesses.

This change would require additional work to allow multiple instances of the combined system to work together on a large network, such as is described in Section 6.2.5.
6.2.4 Caching

Over the course of a given day, it is likely that each user will make multiple separate connections from the same source machine to the same destination. As of now, each connection will require the Controller to make a separate request to the *VPE Engine*. If the result of these requests were cached on the controller, then the *VPE Engine* would have to evaluate far fewer requests.

A local cache could be made on the controller, that would store the access levels associated with certain requests. For example, it could be saved that *User A* has *FullAccess* to a common server. When the *VPE Engine* receives a rule change, a message could be sent to all attached controllers to invalidate their caches and start rebuilding them from scratch.

However, caching these requests could potentially introduce some additional difficulties, and could also undermine some of the security aspects of this system. For example, *AuthAccess* results should most likely never be cached, because caching a result as permitted could allow an attacker to make a connection without the user being notified. Also, certain conditions may not work properly, such as a *TimeCondition*, as the *VPE Engine* does not check conditions until an associated rule is checked.

6.2.5 Distributed Computation

As networks grow, scale becomes a common issue with most security strategies. As more machines are present on the network, the amount of traffic on the network will also increase. This is also a potential problem with this system. As more machines make traffic, the *VPE Engine* in particular will have to evaluate many
more rules in a given time period. One way to mitigate this issue is to distribute this computation, such as by adding additional \textit{VPE Engines}.

In order for the system to run with multiple \textit{VPE Engines}, a synchronization scheme would have to be put in place. One method would be to have each \textit{VPE Engine} connect to a centralized service, and when an update is made to a \textit{VPE Engine}, a message would be sent to all of the \textit{VPE Engines} to update as well.

Multiple Controllers could also be introduced, and this would allow for a more effective use of multiple \textit{VPE Engines}. If a network is split into multiple subnets, then each subnet could have its own Controller, and each Controller could interact with a different \textit{VPE Engine}.

\section*{6.2.6 Fault Tolerance}

Many security systems, including software-defined networks, have a single point of failure. This means that if a single system goes down, such as a central firewall, the security system, or even the entire network, will also be unable to properly function. One way to mitigate this issue is to have a decentralized structure.

There are many types of software-defined networks, and there are also many different controllers [33]. While many of these controllers are centralized, some are distributed [34]. While \textit{Ryu}, the controller used in this system, is centralized, a distributed controller could be used instead. However, this system does currently have just a single \textit{VPE Engine}.

The benefits discussed in Section 6.2.5, while increasing the efficiency of the system, could also provide fault tolerance. With multiple synchronized \textit{VPE Engines} present on the network, each controller could switch to communicating with a different \textit{VPE Engine} if the main one is down.
6.2.7 Indexing

In its current implementation, the VPE Engine checks each rule from highest to lowest, until a rule is found that applies to the connection. However, this is inefficient, and as more rules are set, the speed of this process will decrease proportionally. This issue could be improved through a relatively low number of indexes.

While the VPE Engine already maintains several indexes for common information, such as an index of public keys to machines, there are few indexes involving rules. A few sets of rule indexes could be particularly effective, such as an index that stores the list of rules that applies to a given user. To reduce the number of rule checks even further, an index could be created for each user/source machine pair. Since a given user/source machine will likely only have a handful of rules, this could increase the rule scanning speed tremendously.

While generating these indexes may take a decent amount of computation, this is unlikely to have a significantly negative effect on the system when in use. This is because rules are most likely going to be fairly static, and rule updates should only be made occasionally. The only time these indexes have to be build would be during the VPE Engine’s startup and when a rule update occurs.

6.2.8 MFA Strictness Options

Currently, authorizing a connection through MFA only authorizes the specific connection in question. However, this is not the only possible behavior for such an authorization, and several alternate behaviors are described in Table 6.1.
Table 6.1: Alternative MFA strategies

<table>
<thead>
<tr>
<th>Strictness</th>
<th>User</th>
<th>Source</th>
<th>S Port</th>
<th>Destination</th>
<th>D Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>UserA</td>
<td>Machine-1</td>
<td>TCP/31985</td>
<td>Machine-2</td>
<td>ssh</td>
</tr>
<tr>
<td>3</td>
<td>UserA</td>
<td>Machine-1</td>
<td>Any</td>
<td>Machine-2</td>
<td>ssh</td>
</tr>
<tr>
<td>2</td>
<td>UserA</td>
<td>Machine-1</td>
<td>Any</td>
<td>Machine-2</td>
<td>Any</td>
</tr>
<tr>
<td>1</td>
<td>UserA</td>
<td>Machine-1</td>
<td>Any</td>
<td>Any</td>
<td>ssh</td>
</tr>
<tr>
<td>0</td>
<td>UserA</td>
<td>Machine-1</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>

This table displays several of the possible rule results that can be applied when UserA initiates a connection with AuthAccess from Machine-1 to Machine-2, with a source port of TCP/31985 and a destination port of TCP/22 (ssh). Each of these rule evaluations has been assigned a number, called the Strictness, that corresponds to a different MFA strategy. It is important to note that each of these strategies only involves connections with AuthAccess by not requiring the MFA authorization as often, and would not interfere with any rules that provide FullAccess or NoAccess.

Strictness 4 corresponds to the strategy of only authorizing the exact user, source, destination, source port, and destination port involved in the authorized connection. This is the same authorization strategy currently used in the VPE System. This provides some benefits, such as allowing long-running connections to be authorized once per rule timeout. However, this has some limitations, such as not allowing any further connections without reauthorization, even to the same destination machine and port. For example, if UserA was running single ssh commands between the same source and destination machines, they would have to authorize every individual command.
Strictness 3 corresponds to the strategy of authorizing the user, source, destination, and destination port, without regard to the source port being used. This allows a user to connect to the same destination machine and port multiple times and from different source ports, without having to authenticate multiple times within the rule evaluation’s timeout window. For example, this would allow UserA to run multiple `ssh` commands in a row, while only being required to authorize the first command, so long as these commands were run within the timeout window of the rule evaluation. However, if UserA attempted to connect to other destination ports, such as by downloading web pages with the `curl` command, they would have to separately authorize the resulting connections. This would be much more convenient for the user, though this would result in slightly worse security. An unauthorized entity, running as UserA on the same machine, would be able to connect to the same destination machine and port, until the rule evaluation expired.

Strictness 2 corresponds to the strategy of authorizing the user, source, and destination, without regard to either the source port being used, or the destination port being reached. This allows the user to initiate any number of connection between the source and destination machines, even when connecting to multiple services. For example, UserA would be able to run multiple `ssh` commands, as well as multiple `curl` commands, while still only having to authenticate a single time per timeout window. This provides even more convenience than the authorization strategy provided by Strictness 3, while also providing less security.

Strictness 1 corresponds to the strategy of authorizing the user, source machine, and destination port, without regard to the source port or the destination machine. This would allow the user to connect to the same destination port on
multiple machines, which would be useful when connecting to the same service across multiple machines on a network. For example, this would allow UserA to run `ssh` commands on multiple machines on the network any number of times, while only authenticating once. However, UserA would have to authenticate separately if they attempted to connect to any other ports, such as with the `curl` command. Depending on the situation, this strategy provides varying levels of convenience and security compared to that provided by Strictness 2.

Strictness 0 corresponds to the strategy of authorizing the user and source, without regard to destination machine, source port, or destination port. Essentially, this simply authenticates a user and source machine session, allowing them to connect to any port on any machine any number of times within the timeout windows. For example, UserA could run any number of `ssh` and `curl` commands to any machines on the network, while only having to authenticate once per timeout window. This provides the most convenience out of each of the authorization strategies described, while also providing the least amount of security. If this strategy was used, an unauthorized entity, running as UserA on the same machine, would have the maximum permissions on the network that are provided to an UserA while authenticated. Therefore, this strategy is not recommended in a production environment.

Each of these strategies has its benefits and drawbacks, and provide varying amounts of convenience and security. In the future, the VPE System will support these strategies, in order to provide users with a higher degree of flexibility.

There exist multiple possible authorization strategies besides those that have been mentioned, such as allowing each of the same sets of variables while also allowing the same user to connect from different source machines, or allowing any
user on the source machine the same levels of access. However, these each varying amounts of security compared to the discussed strategies, with many possibilities largely undermining the security mechanisms of the VPE System. Therefore, these additional strategies will not be described in detail, and may not be present in future versions of the VPE System.

6.2.9 Rule Aggregation

Every individual packet that is not cached at a switch must be evaluated by the VPE System, and each rule evaluation only specifies a specific combination of source, destination, source port, destination port, and protocol. This allows for fine-grained control of traffic within the network, and has only a small effect on most network activities. A single DNS lookup, web page download, or file download only requires a few rule evaluations, as most packets are part of the same network flow and match an existing OpenFlow rule within the switch. This has been demonstrated by the experimentation described in Sections 5.3.3, 5.3.4, 5.3.5, and 5.3.6.

However, certain activities, such as horizontal and vertical network scans, produce a large number of distinct network flows. Each of these flows requires a separate rule evaluation, sometimes for only a single packet. In these cases, the latency introduced by the VPE System takes a much more noticeable effect, as demonstrated by the experimentation described in Sections 5.3.1 and 5.3.2.

This effect could be mitigated through the use of rule aggregation, where a single rule evaluation may match multiple network flows. If the VPE System were able to detect an ongoing scan, then rule aggregation could be used to limit the number of rule evaluations that need to be performed. For example, if a user is
running a port vertical scan on a machine to which they only have access to a few of the ports, then the VPE System could detect an ongoing scan and set OpenFlow rules to allow access to those ports and block all others for a short time. This type of functionality may be explored and implemented in the future.

6.2.10 Proactive Agent and Pre-Authorization

In the current system, the VPE Agent is fully reactive. The agent records the users corresponding to packets when they are sent out of the machine, and responds to queries sent by the VPE Controller. This results in a system that can only process a packet after sending a user query from the VPE Controller to the VPE Agent and back, before evaluating the user response in the VPE Engine. As the packet is waiting within the switch during this process, and this entire process must occur for every new flow, there is a certain minimum amount of time that the rule evaluations can take.

In the future, the VPE Agent could be designed to be more proactive. Instead of waiting for a user query, the VPE Agent could send the user response as soon as the packet is sent to the switch. However, the packet information would still be stored on the agent for a time in order to keep track of recently sent user responses. This would be required to avoid sending more responses to the switch than necessary.

Another possible method to decrease authorization times, particularly with regard to network scans, would be to allow the user to inform the ruler of traffic ahead of time. For example, if a user was about to perform a vertical scan of a machine, they could use the VPE Agent to send a message to the VPE Controller about the impending scan. The VPE Controller and VPE Engine could
then determine which ports the user is allowed to access, and then set OpenFlow rules accordingly. The user could then perform the scan without waiting for the repeated rule evaluations to complete. This option would be similar to and work in conjunction with the methods described in Sections 6.2.8, 6.2.9, and 6.2.7.

6.2.11 Efficiency

Most of the components of this system are written in Python, which is an interpreted language. Changing components to be written in a compiled language might create some speed benefits. For example, while the Agent has the kernel module component written in C, the main component of the Agent is written in Python. In the future, all components could be written in C, or instead the components could both be merged into the kernel module.

Additionally, many processes used are less efficient than other options, such as messages being primarily composed of JavaScript Object Notation (JSON) objects. Other systems, such as reading pre-defined numbers of bytes from the socket, may be more efficient than encoding and decoding JSON objects.

Some aspects of the VPE System have already been designed with potential speed improvements in mind. One of these is the structure of messages. A message is composed of two parts, the message type and the message body, and username query messages have a third component that contains the source machine’s IP address. While the message body is written in JSON, the message type, as well as the source machine’s IP address in the case of the query message, is sent separately. This is useful because of the repeated forwarding these messages undergo. In particular, this splitting of the IP address from the rest of the query body allows the switch agent to determine if it can forward the message without having to
unpack the JSON object in the message body. Most likely, several other similar
improvements could be made, such as through the indexing mentioned in Section
6.2.7.

6.2.12 Auditing

A common feature of SDNs is to facilitate auditing (or monitoring) of the network
[34], because they often send information about traffic to a controller. If the
Controller or VPE Engine were modified to record this information, then this
data could be used for auditing by a separate program, or by a new feature in the
VPE System itself.

One method of this would be to send the completed username queries and the
VPE Engine’s responses to a database, such as Elasticsearch. Other tools, such
as Kibana, could then be used to analyze and graph information about the traffic.
One such visualization that could be useful would be the amount of traffic sent by
each user over time.

6.2.13 Additional Interfaces

The VPE Engine has a fairly simple interface, and is designed to allow the user
to add, delete, edit, and view the various stored objects. However, additional
capabilities could be added, such as listing all of the rules that apply to a given set
of variables. Such capabilities would make developing a set of rules much easier.

Currently, the VPE Engine can only be controlled through the command-line
interface (CLI). However, a graphical user interface (GUI) would also make config-
uration simpler. Such an interface would use the same API as the CLI, but would
have more features to be able to better organize the information for the user.
One helpful feature would be allowing the user to select variables from a list, rather than manually typing a particular name. Another feature could be to display the rules in a lost, sorted by priority, and to allow the rules to be reordered by dragging rules above and below each other.

Another important interface would be an Administrator Console, which may be integrated into the GUI described above. This could allow for features such as notifying a system administrator when a new agent is attempting to send traffic over the network, and allowing the administrator to add it to the VPE Engine. Additionally, other notifications about signature validation issues could alert the administrator to potential threats on the network.

### 6.3 Conclusion

I have designed and implemented a security mechanism that addresses several of the issues commonly found in traditional networks. This mechanism makes visibility of the network a security property that can be manipulated to ensure that users have exactly the security rights that they require, and no other. I do this by creating security rules capable of differentiating traffic based on user, rather than machine, while ensuring that malicious entities cannot access secure servers through the use of multi-factor authentication.
Bibliography


trust cloud networks with transport access control and first packet authentica-
tion,” in 2016 IEEE International Conference on Smart Cloud (SmartCloud),
Nov 2016, pp. 5–10.


[21] Jess, the rule engine for the java platform. [Online]. Available: https:
//herzberg.ca.sandia.gov/docs/

[22] Soar home - soar cognitive architecture. [Online]. Available: https:
//soar.eecs.umich.edu/

openflow-spec-v1.3.1.pdf

chaos_calmer

openflow-openwrt

[26] Production quality, multilayer open virtual switch. [Online]. Available:
https://www.openvswitch.org/


[48] D. Golle, “automake: import upstream fix for perl 5.26 · lede-project/source@92c80f3.” [Online]. Available: https://github.com/lede-project/source/commit/92c80f38c0f3c20388f9ac13d5196f2745aeaf77


Appendices
Each of these instruction sets are written with the assumption that the machines in question are running instances of Ubuntu Server 18.04.1, unless otherwise stated. While many of these commands may not work for other Linux distributions, such as *apt* [37], the system components should be able to run on most Linux distributions, provided the required dependencies are still met.

Additionally, some dependencies may have different names in package managers other than *apt*, such as *rpm* [38] or *pacman* [39], and they can most likely be installed without the use of any package manager at all.
Appendix A

OpenWrt

A.1 Compiling OpenWrt with OpenFlow

Here I will go through the process for creating an OpenWrt image with OpenFlow support, using a clean Ubuntu 18.04.1 machine. The Chaos Calmer version of OpenWrt can be found at [24] and openflow-openwrt can be found at [25]. The original build instructions can be found at [45]. Each of these are separate code repositories unaffiliated with myself and this system.

A.1.1 Initial Dependencies and Setup

Certain dependencies are required to run the compilation of OpenWrt. The commands used to obtain these are shown in Listing A.1.

Listing A.1: Installing OpenWrt build dependencies

```plaintext
# Create a build directory and enter it
mkdir ~/builder
cd ~/builder
```
# Install dependencies
```bash
sudo apt update
sudo apt install -y git make gawk build-essential \
python-minimal subversion libncurses5-dev zlib1g-dev \
libssl-dev
```

# Clone the Chaos Calmer version of {openwrt} and openflow-openwrt
```bash
git clone https://github.com/openwrt/chaos_calmer
git clone https://github.com/CPqD/openflow-openwrt
cd chaos_calmer
```

## A.1.2 Patching Dependency Checking Issue

There is an issue with the dependency detection for git, so a patch is needed to fix it before proceeding with the usual installation. The ticket can be found at [46], and the patch can be found at [47].

There is also an issue with automake for modern versions of Perl, where the ticket can be found at [49] and the patch can be found at [48].

**Listing A.2: Applying git patches to OpenWrt**

```bash
wget https://bit.ly/2Pm80iE -O gitcheck.patch
git apply gitcheck.patch
```
A.1.3 Adding OpenFlow to OpenWrt

OpenFlow needs to be placed such that the OpenWrt builder will include it in the compilation. The commands to do this are shown in Listing A.3.

Listing A.3: Linked OpenFlow into the OpenWrt build

```
1. ln -s ~/builder/openflow-openwrt/openflow-1.3/files/
2. ln -s ~/builder/openflow-openwrt/openflow-1.3/package/
```

A.1.4 Update and Install Feeds

The various feeds can be updated and installed using the commands shown in Listing A.4. If you encounter an error running these commands, then you are most likely missing dependencies.

Listing A.4: Updating and installing OpenWrt feeds

```
1. ./scripts/feeds update -a
2. ./scripts/feeds install -a
```

A.1.5 Final Settings and Running Build

Now you will run the configuration menu to select the components to compile, and then run the compilation. Make sure to save the configuration you define in the menuconfig as `.config` (the default).

Listing A.5: Make menuconfig command

```
make menuconfig
```

This will open the configuration menu. In this menu add OpenFlow and Openvswitch to the compilation by selecting these options:
• *Network ⇒ openflow.*

• *Network ⇒ openvswitch*

You will also need to select the target and subtarget for the system you are compiling for. If you are creating images to use with virtualbox, you should set:

• *Target System ⇒ x86*

• *Subtarget ⇒ x86_64*

• *Target Images ⇒ Build VirtualBox image files (VDI).*

Finally, start the build with the `make` command. If the compilation is successful, the built images should be located in `/builder/bin/x86`.

### A.2 Installing OpenWrt in VirtualBox

If VirtualBox is installed, then the `VboxManage` command should be available.

A freshly compiled OpenWrt image, such as the one produced in Section A.1, is set to have very little disk space available. This can cause issues when installing packages, such as openvswitch, which is needed to utilize the OpenFlow capabilities of the image. Thus, you should copy the image to a permanent location, and then modify the size of the image with the command shown in Listing A.6.

Listing A.6: Resize virtualbox image

```
VboxManage modifymedium {imagemname} --resize {size}
```

In my case, the imagename was `openwrt-x86-64-combined-ext4.vdi`, and the size was 4096 (which is in megabytes). However, the size could be as low as 512 MB.
Next, create a new VirtualBox machine. Select *Linux* as the type, and *Other Linux (64-bit)* as the version. For the memory, I use 512 MB, though OpenWrt can function with much less. When prompted for a hard disk, select *Use an existing hard disk file* and select the enlarged image. Continue to the next step to create the machine.

The newly-created machine should have a single network adapter with type NAT. You will need to configure OpenWrt to use this adapter to access the internet. To do this, boot up the virtual machine, and wait a short time for this process to complete. You may need to select a key to get the machine to present you with a shell prompt, though you will not be prompted to do this.

To use the NAT adapter, attached to *eth0*, modify the network configuration file located at `/etc/config/network` (such as through the *vi* editor).

There should be a section in this file with a line similar to `option 'ifname' 'eth0'`. This section represents the OpenWrt interface attached to the eth0 physical interface. Modify this section to display similar to that which is shown in Listing A.7.

**Listing A.7: Example /etc/config/network using DHCP**

```plaintext
cfgii interface 'lan0'
    option proto 'dhcp'
    option ifname 'eth0'
```

128
This tells OpenWrt to use DHCP on the eth0 address. Save the file (you may need to look up how to do this in vi), and restart the virtual machine. When the machine has booted back up, it should have an IP address for the eth0 interface, allowing the machine to access the internet. You can check the IP address with the `ifconfig` command. If it has not received an ip address, you can run the DHCP client manually with `udhcpc`.

Similar configurations will need to be added for each network interface attached to the switch. For interfaces that will be present on the internal network, the selected virtualbox network type should be 'Internal Network'. Additionally, for each interface whose traffic will be monitored and enforced by the system, the 'Promiscuous Mode' setting (located under 'Advanced') should be set to 'Allow VMs' or 'Allow All'.

If instead of using DHCP to lease IP addresses, each interface can instead be configured with a static IP address. The relevant configuration block for such a configurations would look similar to that which is shown in Listing A.8.

### Listing A.8: Example /etc/config/network using static IP

```plaintext
1  config interface 'lan0'
2    option ifname 'eth0'
3    option proto 'static'
4    option netmask '255.255.255.0'
5    option ipaddr '192.168.1.1'
```

#### A.2.1 Configuring OpenVSwitch

OpenVSwitch is used to create a network bridge in the experimentation network. The commands used to create and configure this bridge are shown in Listing A.9.
# Create the bridge
```
ovs-vsctl add-br virtualnet
```

# Set the protocol for the bridge to only use OpenFlow 1.3
```
ovs-vsctl set bridge virtualnet protocols=OpenFlow13
```

# Add the physical interfaces to the bridge
```
ovs-vsctl add-port virtualnet eth1
ovs-vsctl add-port virtualnet eth
```

```
ovs-vsctl add-port virtualnet eth3
```

# Set the bridge to connect to a controller at 192.168.1.234:6653
```
ovs-vsctl set-controller virtualnet tcp:192.168.1.234:6653
```

# Use DHCP to assign an IP address to the bridge
```
udhcpc -i virtualnet
```
Appendix B

Installing Base Dependencies

B.1 Installing Python and Pip

Most components of this system are written in Python and have Python dependencies. However, Python and Pip are not installed by default on many Linux images. Python is required to run many of the components of the system, such as the Rule Engine, the controller, the switch agent, and the machine agent. Pip, while not strictly required, is useful in order to install the dependencies of many of these components.

Add the *universe* category to the packages which will be searched by editing */etc/apt/sources.list*. You can use any text editor you wish, though you will most likely require root permissions to edit this file. In this file, ensure that *universe* is present on each line. The end result should be similar to the file shown in Listing B.1.
After this, Python 2 (through the python-minimal package) and Pip are able to be installed. To perform the installation, run the commands shown in Listing B.3.

Listing B.3: Install Pip dependencies

```
pip install --user -r requirements.txt
```

Now the various Python components of this system can be run, and Pip can be used to install their dependencies. This can be done directly, or through the use of a requirements file, as shown in Listing B.3

```
pip install --user -r requirements.txt
```

### B.2 Installing the Ryu Controller

The Ryu controller is not installed by default on any of the described Linux images. However, it can be easily installed through the use of Pip.

First, install Python and Pip as described in Section B.1. Then run the command shown in Listing B.4 to install Ryu.
Listing B.4: Install *Ryu* command

```
pip install --user ryu
```

Now, the *Ryu* controller is installed and can be used to run this system.
Appendix C

Installing System Components

C.1 Installing and Running the VPE Controller

The VPE Controller is comprised of a Python script that the Ryu controller runs. Once Python and the Ryu controller are installed using the instructions located in Sections B.1 and B.2, the VPE Controller needs only to be present on the system.

Copy the controller directory to desired location on the machine, and run the application with the command shown in Listing C.1.

Listing C.1: Run VPE Controller

```bash
ryu-manager controller/main.py
```
C.2 Installing the VPE Engine

The VPE Engine consists of a Python package, the command-line interface, and a runner script. Install Python and pip as described in Section B.1, and then add the system components to the desired location of the target machine by copying the following paths: rule_engine, cli, and engine.py.

The VPE Engine itself requires some additional Python packages as dependencies. To install these, navigate to the rule_engine directory and run Pip with the command shown in Listing B.3.

The VPE Engine can then be run using the engine.py script. Simply call python engine.py and the VPE Engine will be running locally. To send commands to the rule engine, navigate to the cli directory and run python cli.py with your desired arguments.

C.3 Installing the VPE SwitchAgent

The VPE SwitchAgent consists of a single Python package. First, install Python with the commands shown in Listing C.2.

Listing C.2: Install Python in OpenWrt

```
1 opkg update
2 opkg install python
```

Next, copy the switch_agent directory to the desired location on the switch. This can now be run with python switch_agent.
C.4 Installing the VPE Agent

C.4.1 Kernel Module Component

The kernel module component of the agent is written in C, and must be compiled for the target system. In order to do this, it requires a C compiler (such as gcc), and make is recommended. These are both installed by default on the Ubuntu Server 18.04.1 image.

First, copy the `kernel_module` directory to the desired location on the target machine. Then enter the directory and run the commands shown in Listing C.3 to compile and load the kernel module.

Listing C.3: Install the VPE Kernel Module

```
1 make all
2 sudo insmod packetreporter.ko
```

You may get a warning about not being able to use `CONFIG_STACK_VALIDATION=y`. This can be solved by running the commands shown in Listing C.4.

Listing C.4: Install libelf-dev

```
1 sudo apt update
2 sudo apt install libelf-dev
```
C.4.2 Userspace Component

The userspace component of the VPE Agent requires Python, Pip, and multiple Python libraries. Install Python and Pip as described in Section B.1, and then copy the agent directory to the target machine. Now, enter the agent directory and install the required libraries with the command shown in Listing B.3.

The userspace component can now be run from the agent directory’s parent directory with python agent.