

Automated Policy Compliance and Change Detection Managed Service in Data Networks

A Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

by

Saeed M. Agbariah
Master of Science
George Mason University, 2007
Bachelor of Science
University of Phoenix, 2005

Director: Bernd-Peter Paris, Professor
Department of Electrical and Computer Engineering

Spring Semester 2015
George Mason University
Fairfax, VA



This work is licensed under a [creative commons attribution-noncommercial 3.0 unported license](https://creativecommons.org/licenses/by-nc/3.0/).

DEDICATION

This is dedicated to my father and my late mother for their endless love, support and encouragement, to my brother Ahmed who bought me my first Commodore64, to my wife and to my wonderful children. Without their help and understanding, I could never have finished my graduate studies.

ACKNOWLEDGEMENTS

I would like to thank my dissertation director, Dr. Bernd-Peter Paris, for his encouragement and support which have helped me continue moving forward through this long process, and for the opportunity to work under his supervision. A special thanks to Dr. Jeremy Allnutt who believed in me and pushed me to achieve this goal.

I would also like to thank the members of my committee, Dr. Brian Mark, Dr. Duminda Wijesekera, Dr. Shyam Prakash Pandula for their guidance throughout my Ph.D. studies.

Last but not least, I would like to thank OPNET Technologies for their inspiring NetDoctor product, and the many people who have supported and motivated me during my research, especially with the C++ programming.

TABLE OF CONTENTS

	Page
List of Tables	ix
List of Figures	x
List of Abbreviations and Symbols.....	xiii
Abstract	xv
Chapter 1: Introduction	1
1.1 Motivation and Problem Summary	1
1.2 Research Summary.....	2
1.3 Overview of Contributions.....	3
1.4 Scope and Limitations.....	4
1.5 Proposal Organization	5
Chapter 2: Background and Related Work.....	6
2.1 The Evolution of Network Management.....	6
2.1.1 Fault	8
2.1.2 Configuration.....	9
2.1.3 Accounting.....	10
2.1.4 Performance.....	10
2.1.5 Security	11
2.1.6 Consideration of the FCAPS Model.....	12
2.2 Simple Network Management Protocol (SNMP)	13
2.3 Change Management.....	16
2.4 Policy-Based Network Management.....	20
2.5 Network Configuration Protocol (NETCONF).....	23
2.6 Configuration Auditing and Policy Compliance.....	26
Chapter 3: Contributions.....	32
3.1 Policy Exchange and Management	34
3.1.1 Policy Client and Policy Decision Control Communication	36

3.2 Design Considerations.....	42
3.2.1 Masquerading	43
3.2.2 Communication Interruptions.....	44
3.2.3 Malicious Interruption	45
3.2.4 New Policy Push during Configuration Change.....	46
3.2.5 Periodic Check Detects New Policy during Configuration	47
3.3 Common Policy Language	47
3.3.1 Configuration Templates	48
3.3.2 One- and Two-Phase Commit Models	51
3.4 Common Policy Language Format	52
3.4.1 Mandatory Sections	52
3.4.2 Section Delimiters and Predefined Keywords.....	54
3.4.3 Conditional Commands	58
3.4.4 If-Then	60
3.4.5 Variables and Parameters	61
3.5 Summary	66
Chapter 4: Implementing and Testing Policy Exchange and Management.....	68
4.1 Motivation and Problem Summary	69
4.2 System Requirements and Installation	71
4.2.1 Server.....	72
4.2.2 Client	73
4.2.3 Policy Exchange Protocol.....	76
4.2.4 Message Types and Exchange	76
4.2.5 Message Format.....	81
4.3 Example Scenarios	84
4.3.1 Unknown Client Role	85
4.3.2 Known Client Role, No Policy Stored.....	92
4.3.3 Known Client Role with Existing Policy with the Same Version Number.....	98
4.3.4 Known Client Role with Older Existing Policy	104
4.3.5 Known Client Role with Newer Existing Policy.....	109
4.3.6 The Server Has a Newer Policy.....	113
4.3.7 Periodic Checks	116

4.4 Summary	124
Chapter 5: Implementaing and Testing the Runtime Compliance Manager.....	126
5.1 RCM Design, Requirements and Installation.....	127
5.1.1 System Requirements	128
5.1.2 Installing a Running RCM.....	128
5.1.3 Logs	130
5.2 Policy File Syntax	132
5.2.1 Variables.....	138
5.2.2 Printing Variables	139
5.3 Client-Monitored Features	140
5.3.1 Network Interface	140
5.3.2 NTP.....	142
5.3.3 Hostname	143
5.3.4 File System Size Limit (FSSL).....	145
5.3.5 DNS	146
5.3.6 SSH.....	147
5.3.7 Environment Variables (ENV)	149
5.3.8 Hosts	150
5.4 Example Scenarios	151
5.4.1 Section with 'Exact' Keyword.....	152
5.4.2 Section with 'Exclude' Keyword.....	163
5.4.3 Section with 'Ignore' Keyword	168
5.4.4 Conditional Block.....	171
5.4.5 If-Then Block	172
5.4.6 Variables.....	178
5.5 Summary	185
Chapter 6: Summary and Conclusions.....	187
6.1 Conclusion.....	187
6.2 Limitations and Future Work	190
Appendix A – Client/Server Policy Exchange and Management Source Code	193
Appendix B – Runtime Compliance Manager Code	196
Appendix C – Policy File Example	198

References..... 204

LIST OF TABLES

Table	Page
Table 1 The Main Objective of Each Functional Area in the FCAPS Model	8
Table 2 Change Management Study Surprising Results[21]	19
Table 3 Test Results by Test Suite[37]	26
Table 4 Message Header Fields and Values	82
Table 5 Object Header Fields and Values	83
Table 6 Section Block Parameters	135
Table 7 Predefined Keywords	136
Table 8 Conditional Block Parameters	137
Table 9 If Block Parameters	138
Table 10 Network Interface Block	141
Table 11 NTP Block	142
Table 12 Hostname Block	144
Table 13 FSSL Block	146
Table 14 DNS Block	147
Table 15 SSH Block	148
Table 16 ENV Block	150
Table 17 Hosts Block	151

LIST OF FIGURES

Figure	Page
Figure 1 TMN Reference Model Refined with FCAPS[7].....	8
Figure 2 Risk Matrix[19]	18
Figure 3 Layers of NETCONF[37].....	24
Figure 4 Configuration Audit Report.....	29
Figure 5 Process Flow.....	33
Figure 6 Policy Management System Framework.....	35
Figure 7 Policy Client and Decision Server Communication.....	39
Figure 8 Policy Update from the policy decision control server	40
Figure 9 Policy Client Status	41
Figure 10 Client Check Process	42
Figure 11 if...then Logic	50
Figure 12 Server's Flow Chart.....	73
Figure 13 Client Flow Chart	75
Figure 14 Client/Server Successful Admission	78
Figure 15 Client/Server Denied Admission.....	79
Figure 16 Client/Server Normal Cycle	80
Figure 17 Unsolicited Message Triggered.....	80
Figure 18 Message Format.....	81
Figure 19 Object Header Format	83
Figure 20 Hash Code Header.....	84
Figure 21 Starting the PolicyServer.....	86
Figure 22 Starting the PolicyClient.....	86
Figure 23 Client Is Sending an OPEN Message	87
Figure 24 Server Sends CC after Receiving Unknown Role.....	89
Figure 25 The Policy Client Receives CC Message	90
Figure 26 OPEN Message Layout	91
Figure 27 OPEN Message from Wireshark	91
Figure 28 CC Message Layout	92
Figure 29 OPEN, CAT, DEC and RPT Message Exchange.....	93
Figure 30 policy decision control server Message Exchange.....	95
Figure 31 Policy Client REQ Message Sent	96
Figure 32 policy decision control server Sending Policy File	97
Figure 33 Hash Code Matching	97
Figure 34 RPT Message Sent.....	97
Figure 35 RPT Message Received.....	98

Figure 36 Client OPEN Sent.....	99
Figure 37 CAT Message Sent.....	100
Figure 38 REQ with Version Number	101
Figure 39 Wireshark Capture REQ with Version Number.....	101
Figure 40 DEC Message Sent.....	102
Figure 41 Client Receives a DEC and Sends RPT message.....	103
Figure 42 RPT Received by the Server.....	104
Figure 43 OPEN Sent, CAT Received.....	105
Figure 44 Version Number Sent Within DEC	106
Figure 45 DEC Sent, File Pushed	107
Figure 46 DEC Received, File Download Started	107
Figure 47 File Download Completed, Sending RPT	108
Figure 48 RPT Received Confirming File Download	108
Figure 49 OPEN Sent, CAT Received.....	110
Figure 50 Awaiting DEC from Server	110
Figure 51 Client Has a Newer Version.....	111
Figure 52 DEC Received, RPT Sent.....	112
Figure 53 RPT Received, Transaction Completed	113
Figure 54 Unsolicited DEC Sent.....	114
Figure 55 Unsolicited DEC Received.....	114
Figure 56 RPT Sent to Policy Server.....	115
Figure 57 RPT Received From Client	115
Figure 58 Waiting for an Event	117
Figure 59 Client's Periodic Check.....	118
Figure 60 Client Configured with Periodic Interval	120
Figure 61 Server Replying with DEC, Waiting for an Event	121
Figure 62 Second Periodic Check, Client-Side.....	122
Figure 63 Second Periodic Check, Server-Side.....	123
Figure 64 RCM Design.....	128
Figure 65 RCM Flow Chart	130
Figure 66 New Policy File Detected.....	131
Figure 67 Machine Report	132
Figure 68 Policy File Example	133
Figure 69 Printing Block Example	139
Figure 70 FSSL Disk Usage Example	145
Figure 71 Policy File: Section with 'Exact' Keyword.....	153
Figure 72 Section with 'Exact' Report 1.....	154
Figure 73 Client's IP Address.....	155
Figure 74 Client's NTP Servers	156
Figure 75 Client's Hostname.....	156
Figure 76 Client's DNS Configuration.....	157
Figure 77 Client SSH Configuration.....	158
Figure 78 Client's ENV Variables	159
Figure 79 Client's Hosts File.....	159

Figure 80 System Interface IP Changed	160
Figure 81 Network Interface in Compliance	161
Figure 82 System NTP Servers Updated	162
Figure 83 NTP Is in Compliance	162
Figure 84 Policy File: Section with 'Exclude' Keyword.....	164
Figure 85 Section with 'Exclude' Report.....	165
Figure 86 Network Interface Mask	166
Figure 87 NTP Server Present	167
Figure 88 Hostname Is a Match	167
Figure 89 Environment Variable Section	168
Figure 90 ENV Variable Trigger	169
Figure 91 Using the 'Ignore' Keyword.....	170
Figure 92 TERM Alarm Cleared	171
Figure 93 ENV Configuration	171
Figure 94 Conditional Block.....	172
Figure 95 If-Then Policy.....	173
Figure 96 If-Then Report	174
Figure 97 Eth0 IP Configuration.....	175
Figure 98 NTP Configuration	176
Figure 99 Hostname Configuration	177
Figure 100 Hostname Changed to 'desktop'.....	177
Figure 101 DNS Warning	177
Figure 102 DNS Configuration.....	178
Figure 103 Global-Single Policy	179
Figure 104 Global-Single Report.....	179
Figure 105 SSH Configuration	180
Figure 106 Hosts File.....	181
Figure 107 Eth0 IP Address Change.....	181
Figure 108 New Line 13 Warning	181
Figure 109 New Line 18 Warning	182
Figure 110 Global-List Policy	183
Figure 111 Global-List Report.....	184
Figure 112 System Verification	185

LIST OF ABBREVIATIONS AND SYMBOLS

Client Accept Message	CAT
Close Client Message.....	CC
Cisco Internetwork Operating System	Cisco IOS
Cisco Internetwork Operating System Next Generation	Cisco IOS-XR
Common Open Policy Service.....	COPS
Common Open Policy Service - Provisioning Service.....	COPS-PR
Common Policy Language.....	CPL
Central Processing Unit	CPU
Decision	DEC
Dynamic Host Configuration Protocol	DHCP
Distributed Management Task Force.....	DMTF
Domain Name System	DNS
Fault, Configuration, Accounting, Performance, Security	FCAPS
Federal Information Security Management Act	FISMA
File System Size Limit.....	FSSL
Fully Qualified Domain Name.....	FQDN
GNU C Compiler	G++
Health Insurance Portability and Accountability Act	HIPAA
Intrusion Detection System.....	IDS
Internet Engineering Task Force.....	IETF
Internet Protocol.....	IP
Inter Process Communication.....	IPC
Intrusion Prevention System.....	IPS
Information Technology	IT
International Telecommunication Union	ITU-T
Juniper Operating System.....	JunOS
Lightweight Directory Access Protocol.....	LDAP
Message-Digest Algorithm	MD5
Management Information Base.....	MIB
Network Configuration Protocol.....	NETCONF
Network Time Protocol.....	NTP
National Institute of Standards and Technology.....	NIST
Network Management System.....	NMS
National Security Agency	NSA
OPEN Message	OPEN
Operational Support Systems.....	OSS

Policy-Based Network Management	PBNM
Payment Card Industry	PCI
Policy Decision Point.....	PDP
Policy Enforcement Point	PEP
Policy Information Base	PIB
Portable Operating System Interface	POSIX
Quality of Service	QoS
Runtime Compliance Manager	RCM
Regular Expression	REGEX
Report Message.....	RPT
Request Message.....	REQ
Request for Comments.....	RFC
Remote Procedure Call	RPC
Simple Authentication and Security Layer	SASL
Simple Network Management Protocol.....	SNMP
Service-Oriented Architecture	SOA
Simple Object Access Protocol.....	SOAP
Secure Shell	SSH
Transmission Control Protocol	TCP
Telecommunications Management Network	TMN
User Datagram Protocol.....	UDP
Extensible Markup Language	XML
Extensible Stylesheet Language	XSLT

ABSTRACT

AUTOMATED POLICY COMPLIANCE AND CHANGE DETECTION MANAGED SERVICE IN DATA NETWORKS

Saeed M. Agbariah, Ph.D.

George Mason University, 2015

Dissertation Director: Dr. Bernd-Peter Paris

As networks continue to grow in size, speed and complexity, as well as in the diversification of their services, they require many ad-hoc configuration changes. Such changes may lead to potential configuration errors, policy violations, inefficiencies and vulnerable states. Even the best administrators can make mistakes, and the cost of missing a key configuration or accidentally skipping an asset can be catastrophic. Labor-intensive manual network auditing or, more recently, products using dedicated configuration compliance scanning appliances for verifying individual system configuration can lead to the discovery of configuration errors and policy violations. However, seldom is the discovery made in real-time, which can prevent system outages, service disruptions and security risks before they occur.

The current Network Management landscape is in dire need of an automated process to prioritize and manage risk, audit configurations against internal policies and

external best practices, and provide centralized reporting for monitoring and regulatory purposes in real-time. A significant challenge for any organization is ensuring that system configurations remain compliant with internal and regulatory security and compliance policies.

The purpose of this dissertation is to define a framework for an automated configuration process with a policy compliance and change detection system, which performs automatic and intelligent network configuration audits by using pre-defined configuration templates and a library of rules that encompasses industry standards for various routing and security-related guidelines, as well as policies such as these required by FISMA, Sarbanes-Oxley, HIPAA, PCI, NIST, Cisco, and NSA.

System administrators and change initiators will have real-time feedback if any of their configuration changes violate any of the policies set for any given device. The suggested architecture achieves a high level of security and compliance, and reduces complexity in network configuration without adding any functions onto the managed entity.

CHAPTER 1: INTRODUCTION

1.1 Motivation and Problem Summary

Current networks are evolving rapidly. This rapid growth of networks and services has introduced new, complex, large networks that are made up of heterogeneous equipment from multiple vendors. As these networks continue to grow their systems and services, the task of configuration management for IP network devices is becoming more and more difficult. Not only is this heterogeneous equipment supporting different techniques in conjunction with its own configuration methods; it is a common practice to find the same device deployed in the network in multiple roles, each with its unique configuration requirements and policies governing the device within the organization, or even with policies that differ from one business unit to another within the same organization. All these complications increase the likelihood of faulty configurations, and thus increase the difficulty of anticipating what complex chain of changes may happen to the network as a result of changing one configuration parameter. As a result, network administrators dealing with the existence of a huge set of configuration parameters, and the implicit dependencies between these parameters, are confronted with the challenge of configuring these services and their network elements without committing a single mistake.

On the other hand, current networks require ad-hoc changes by network administrators to continuously conduct provisioning or performance tuning. These configuration changes are costly and error-prone, and can result in unpredictable failures and inefficiencies. Or they may lead to inefficient allocation of underlying resources, turning the active device into a traffic bottleneck. Worse, an inconsistent configuration can cause not only traffic loss, but also intermittent crashes of the network devices [1]. The study in [2] has found that 50 percent of network errors are configuration errors and 75 percent of all Time to Repair hours are due to administrator errors. Another study has revealed that 80 percent of IT budgets in enterprise networks are dedicated just to maintaining the current operating environments [3].

1.2 Research Summary

This dissertation will present a framework for an Automated Policy Compliance and Change Detection System, which performs automatic and intelligent network configuration audits by using pre-defined configuration templates and a library of rules that encompasses industry standards for various routing and security-related guidelines. It is a system that will provide real-time alerts for any configuration changes that violate any of the policies set for any given device. The suggested architecture achieves a high level of security and compliance and reduces complexity in network configuration without adding any functions onto the managed entities.

1.3 Overview of Contributions

The central research idea seeks to replace labor-intensive configuration management that is error-prone and often results in unpredictable failures and inefficiencies, with one that is automated and reduces errors and inefficiencies.

The framework seeks to define the following areas:

- 1) A Common Policy Language Language for representing device, organizational and industry best practices and any other regulatory policies or guidelines needed for any network elements; in a structured document format that can be retrieved and manipulated with ease.
- 2) A centralized repository where policies could be stored, allowing policy changes to propagate to the subjects, and allowing subjects to detect policy changes in an automated way.
- 3) A secure policy exchange procedure.
- 4) A proposed framework that permits any given device to be configured in its current native state, without any further burdens to the network administrator or system owner, with the ability to detect whether the proposed configuration violates any of the policies set in item 1.

- 5) A protocol to provide mechanisms for immediate feedback to the change initiator, and to a centralized alarm system, alerting them of any conflict.

In summary, the change initiator will access the network device, type his/her changes and immediately know whether the changes he/she committed have violated any policy set for the device. The aim is to prevent inconsistent configuration states which would result in operational failures or inefficiencies.

1.4 Scope and Limitations

There are many elements that networks comprise, including but not limited to routers, hubs, switches, bridges, firewalls, intrusion detection system (IDS), intrusion protection systems (IPS), wireless access points, servers, applications and protocols. Furthermore, because of technical innovation and vendor diversity, in practice heterogeneity exists in networks, and despite many years of standardization efforts, the number of system interface specifications seems to continue to increase. Another problem lies in the fact that the majority of network elements have a proprietary operating system under exclusive legal right of the copyright holder. Therefore, the suggested framework cannot be based on any single architecture or technology. It must instead be based on recognition of diversity and interoperability, and since proprietary software vendors regard their source code as a trade secret, implementation of the framework on any proprietary system for the scope of this research would be impracticable. To circumvent these

practical limitations we will use an open operating system, such as Linux, to implement and demonstrate our proposed framework.

1.5 Proposal Organization

This dissertation is divided into six chapters and an appendix section. The first chapter provides motivation and definition for the problem I addressed in this research. Chapter 2 provides background and context for the research problem and establishes the need for the research. It shows the current network management state of the art, how we started and where we are today. It shows that, though progress has been made, the industry still lacks a formal method for automated policy compliance and change detection management. The third chapter dives into the details of the contributions of this research. It details the specifics of each component of the framework and shows how they interact with each other, with network elements and with other network management components to provide the required automated policy compliance and change detection, of which the industry is in dire need. The fourth chapter details the implementation and testing of the policy exchange system. The fifth chapter discusses implementation and testing of the policy enforcement system. The last chapter provides a conclusion of the study, and discusses directions for future research. There is also a publications section, and an appendix section that includes any additional information that could be useful to the reader but is not an actual part of the body of the research.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 The Evolution of Network Management

Before discussing automated policy compliance and change detection, it helps to frame the conversation by describing the evolution of network management tools. As discussed earlier, a large variety of challenges in networks has produced the need to create a network management model that can enable network administrators, designers, planners, and operators to perform strategic and tactical planning of engineering, operation, and maintenance of their networks for current and further needs at minimum cost [4]. This entails functions such as initial network planning, resource allocation, predetermined traffic routing to support load balancing, access control, authorization, and a variety of other activities.

There are few reference models that have been widely established for network management. One of them is the Fault, Configuration, Accounting, Performance, and Security model, commonly referred to as FCAPS ([5], pp. 2). The FCAPS model was originally designed by the International Telecommunication Union (ITU-T). As its name indicates, it divides management functions into five categories: fault management, configuration management, accounting management, performance management, and security management.

The ITU-T organization dates back to 1865, and its original responsibility was to ensure efficient and on-time production of high-quality recommendations covering all fields of telecommunications ([5], pp. 2-3). In 1996, the ITU-T created the concept of the Telecommunications Management Framework (TMN), which was an architecture intended to describe service delivery models for telecommunication service providers based on four layers: business management, service management, fault and performance management, and element and configuration management ([5], pp. 3-7). Because TMN standards are mainly business-focused and not focused on managing IP networks, the ITU-T refined the model in 1997 to include the concept of FCAPS ([6], pp. 120-122). FCAPS, as shown in Figure 1, expanded the TMN model to focus on the five functionally different types of tasks handled by network management systems: fault management, configuration management, accounting management, performance management, and security management. Table 1 describes the main objectives of each functional area in the FCAPS model, and in the following pages will be more details on each of the functional areas provided.

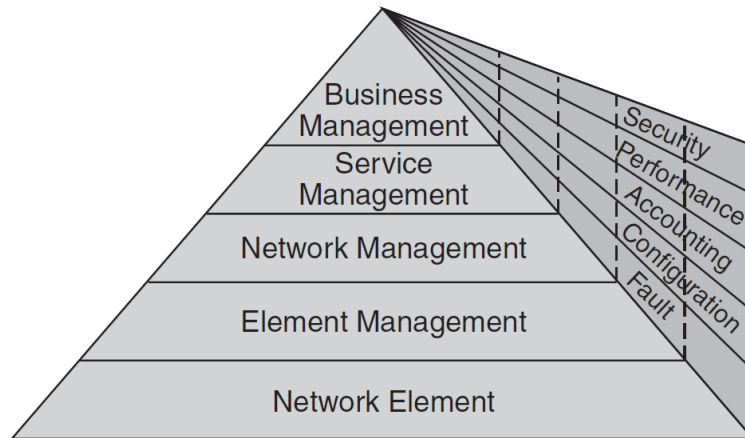


Figure 1 TMN Reference Model Refined with FCAPS[7]

Table 1 The Main Objective of Each Functional Area in the FCAPS Model

Management Functional Area (MFA)	Management Function Set Groups
Fault	Alarm surveillance, fault localization and correlation, testing, trouble administration, network recovery
Configuration	Network planning, engineering, and installation; service planning and negotiation; discovery; provisioning; status and control
Accounting	Usage measurement, collection, aggregation, and mediation; tariffing and pricing
Performance	Performance monitoring and control, performance analysis and trending, quality assurance
Security	Access control and policy; customer profiling; attack detection, prevention, containment, and recovery; security administration

2.1.1 Fault

Fault management includes functions that address alarm surveillance, testing, and fault isolation. Alarm surveillance, as the name implies, allows reporting alarms with different levels of security along with the possible cause of

the alarm. It also provides a summary of alarms that are outstanding, and permits the manager to retrieve the alarm information [8]. The objectives of doing fault management are to increase network availability, reduce network downtime and quickly restore network failures. Effective fault management is critical to ensure that users do not experience disruption of service, and that when they do, the disruption is kept to a minimum. Dealing with alarms and the large volume of events that are constantly being generated is one of the challenges that fault management addresses. However, it encompasses other functions as well, such as troubleshooting and diagnosis.

2.1.2 Configuration

Configuration of network devices is one of the most complex and error-prone network management tasks. This task in particular plays a major role in the focus of this research, because misconfiguration is one of the main sources of network unreachability and vulnerability problems. Network configuration function allows for changes to the configuration of the network devices. This functional area includes functions that allow management systems to provision resources and services and monitor and control their state and status information [8].

Configuration management can be a comprehensive set of tools encompassing collecting, storing, managing, updating and presenting data about network elements or services, and about their relationships. These tools can be vendor-neutral or vendor-specific. Vendor-neutral tools, by far the more common,

are designed for networks containing hardware and software from multiple suppliers. Vendor-specific tools usually work only with the products of a single company, and can offer enhanced performance in networks where that vendor dominates.

2.1.3 Accounting

This functional area enables charges to be established for the use of resources, and for costs to be identified for the use of those resources. Here again, depending on the service, the usage information will vary. For example, a phone service often determines the length of time the connection was used, while a packet service which collects data on the number of packets sent. Accounting management includes functions designed to do the following: inform users of costs incurred or resources consumed, enable accounting limits to be set and tariff schedules to be associated with the use of resources, and enable costs to be combined where multiple resources are invoked to achieve a given communication objective. To some organizations, accounting management tasks are potentially the least relevant. Although some network backbone architectures and organizations honoring Service-Oriented Architectures (SOAs) may incorporate chargebacks and cost-based servicing ([5], pp. 7).

2.1.4 Performance

Performance Management provides functions to evaluate and report upon the behavior of telecommunication equipment and the effectiveness of the

network or network element. Its role is to gather and analyze statistical data for the purpose of monitoring and correcting the behavior and effectiveness of the network, network elements, or other equipment, and to aid in planning, provisioning, maintenance and quality measurement[6], pp. 33-35). The performance management area includes functions to monitor performance parameters (such as errored seconds and number of bad messages), collect traffic statistics and apply control to prevent traffic congestion ([9], pp. 150-154). Monitoring the performance often allows operators to anticipate problems and take care of them before they occur. It can sometimes be useful to have the option of looking at the data later if a problem is discovered, to see if there are any indications in the data of how the problem developed or to just use the data for general analysis. In many cases, such analysis does not have to occur in real-time; it is even possible to perform the analysis offline. This means that statistical performance data need to be collected. Also, periodic snapshots need to be taken and stored somewhere in a file system or database [9], pp. 155-156).

2.1.5 Security

Two aspects need to be distinguished as part of this functional area: security of management, which ensures that the management itself is secure, and management of security, which manages the security of the network. Security management is designed to protect the services and prevent malicious, negligent and abusive behavior by authorized and non-authorized users alike. It maintains access rights, access logs, audit trails, management and governance policy

enforcement; raises security alarms, and distributes necessary security-related information. An effective security management system will provide mechanisms for security administrators to allow: access to selective resources, access to logs, data privacy, access right checking, a security audit trail log, security alarm/event reporting, and security-related information distribution ([5], pp. 8).

2.1.6 Consideration of the FCAPS Model

FCAPS addresses one of the most crucial tasks of network management by monitoring the status of the network, collecting data and avoiding undesirable states of the network. However, it also oversimplifies network management because many cases of functionality cannot be easily categorized, as they can be used for different purposes that fall under different functional categories. For example, logging and reporting events are generally categorized under fault management; however, they also can support performance, configuration management, and security management functionality [10]. Another issue to consider is that FCAPS has led to the development of stovepipe applications. The term 'stovepipe' implies that the application does not integrate with or share data or resources with other applications. Therefore, the practice of using different applications in order to manage network devices can be very challenging since each application extracts information and presents it in a different view for a given object. For example, each application is interested in a different attribute of the device interface and each uses the device interface in a different way. A device interface is usually modeled as a different object for each application.

Thus, information collected from many sources can be very useful when associated together for presenting a given network. This is because the lack of cohesion in presentation prevents the data from being associated at all, and specialized applications cannot take advantage of such data ([11], pp. 106-109).

2.2 Simple Network Management Protocol (SNMP)

Using the FCAPS model as a basis for network management architecture, the trends in network management solutions have followed two general technical directions: ITU-T's Telecommunication Management Network (TMN) for telecommunications networks and Internet Engineering Task Force's (IETF) Simple Network Management Protocol (SNMP) for IP networks ([12], pp. 71-75). For IP networks, the Simple Network Management Protocol (SNMP) has become the de facto standard in the management fields of IP networks. It is probably the best-known management protocol. SNMP is defined in a series of Internet Engineering Task Force (IETF) standards that date back to the late 1980s. The core of SNMP is a simple set of operations that gives administrators the ability to change the state of some SNMP-based devices. For example, a network administrator could use SNMP to shut down an interface on a network router or check the speed of an interface. [13] There have been several versions of SNMP. The common ones are SNMPv1, SNMPv2 and SNMPv3 ([12], pp. 55-56).

SNMP is an application layer protocol and uses the User Datagram Protocol (UDP) to exchange management information between management entities. It is based on asynchronous request-response protocol enhanced with

trap-directed polling. The qualifier 'asynchronous' refers to the fact that the protocol does not need to wait for a response before sending other messages.

'Trap-directed polling' refers to the principle that the manager polls in response to a trap message being sent to an agent, which occurs when there is an exception or after some measure has reached a certain threshold value [14]. The SNMP architecture consists of the SNMP Manager, which usually is a server running software system that can handle management tasks for a network, such as Network Management System (NMS). The Manager's key functions include: querying agents, getting responses from agents, setting variables in agents, and acknowledging events from agents. A managed device is a device or a network element that requires monitoring. An SNMP Agent is a piece of software that runs on the network device or the network element that is being managed. A Management Information Database (otherwise known as a Management Information Base, or MIB) can be thought of as a database of managed objects that the agent tracks.

Any sort of status or statistical information that can be accessed by the NMS is defined in an MIB, such as the temperature on a switch. SNMP defines a set of five management operations, which are the primitives on which all SNMP management is based. Get and get-next requests are used to retrieve management information from an MIB. Set requests are used to write to an MIB. Get responses are used by agents to respond to get, get-next, and set requests. Finally, traps are used to send event messages ([9], pp. 249-250).

While SNMP provides good network management at the macro level, it does not provide all of the network details required to solve many network issues. Its simple design means that the information it deals with is neither detailed nor organized enough to deal with expanding modern networking requirements. In an informational memo (RFC 3535, "Overview of the 2002 IAB Network Management Workshop," May 2003) various drawbacks of SNMP were summarized, for example the memo explains that because SNMP is designed as an API between management applications and devices, it cannot function without management applications. Also, SNMP does not scale well when working with large amounts of data, and the protocol lacks standardized writable MIB objects usable for configuration. It is also not easy to perform device configuration tasks using SNMP. The paper goes on to explain the fact that SNMP MIB modules (both read and read-write) are not deployed by equipment manufacturers in a timely manner [15]. The end result is that SNMP is not used for device configuration tasks. Still, SNMP does work well for periodic monitoring and, in some cases, for event reporting. SNMP will stay around for quite some time, at least for performing the types of tasks it already does well [16].

2.3 Change Management

The FCAPS model is useful for understanding the goals and requirements of Network Management, and also helps to build a foundation for understanding the significance of Network Management to compliance efforts ([12], pp. 90-95), but it does not address change, or how to handle change in an active network. Change in today's network has become inevitable. It also has become one of the most prominent sources of risk in the network, and it has a direct impact on the time, cost and quality of the services provided. To cope with changes and their impact, Change Management has become an IT Service Management discipline. It is one of the most critical processes in IT management. Some of the reasons for this are the sheer number of changes and the difficulty of evaluating the impact of changes on the network or the services it provides in real-time[17]. The main goal of change management is to ensure that the risk and business impact of each change is communicated to all impacted and implicated parties, and to coordinate the implementation of approved changes in accordance with current organizational best practices [18].

Changes in the networks may arise reactively in response to problem-solving errors and adapting to changing circumstances. Change can also arise due to externally-imposed requirements, e.g., a new policy. Or it can be caused proactively by seeking to impose greater efficiency and effectiveness or seeking business benefits such as reduced costs, improved services or new projects. However it may be, Change Management in the context of this research is to

ensure that standardized methods and procedures are used for efficient and prompt handling of all changes, in order to minimize the number and impact of any related incidents upon service. Change Management seeks to ensure that standardized methods, processes, and procedures are used for all changes, to facilitate efficient and prompt handling of all changes, and to maintain the proper balance between the need for change and the potential detrimental impact of change [19].

For the most part, changes are evaluated by stakeholders. In most organizations, a team is designated to evaluate the proposed change by trying to understand its impact on the network or the service it offers. This requires the change management team to have a good understanding of the change and its impact on the network, and to keep track of the details of the system's past and future goals. There are four major roles involved with the change management process, each with separate and distinct responsibilities: The Change Initiator, who initially perceives the need for the change; the Change Manager, who leads a team to review and accept the completed change request, the Change Advisory Board, which exists to support the authorization of changes and to assist the Change Manager in the assessment and prioritization of changes, and the Change Implementation Team (operations), which is responsible for carrying out the actual change and reporting results.

Many organizations use a simple matrix like the one shown in Figure 2 to categorize risk [18]. The Probability Axis denotes the probability of each

identified risk. For this, each risk is listed to the smallest detail possible and the probability of its occurrence is predicted. The Impact Axis assigns a percentage of impact, in the event that the risk does occur [19]. As a result, changes that have low impact in the event of failure and low probability of failure become candidates for a streamlined approval path [18].

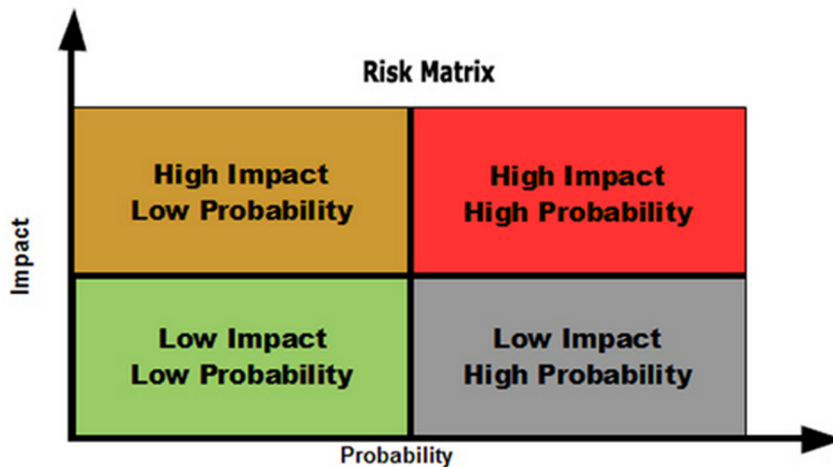


Figure 2 Risk Matrix[19]

In a recent survey conducted by author and Information Technology Service Management expert Harris Kern, he reports that of 40 corporate IT infrastructure managers, a surprising 60 percent admitted that their processes to handle change are not effective in communicating and coordinating changes occurring within their production environment. Table 2 lists the key findings of the study [20].

Table 2 Change Management Study Surprising Results[21]

Not all changes are logged	95%
Changes not thoroughly tested	90%
Lack of process enforcement	85%
Poor change communication and dissemination	65%
Lack of centralized process ownership	60%
Lack of change approval policy	50%
Frequent change notification after the fact	40%

The above statistics are not hard to imagine, particularly for IT technicians, for whom change is a constant, almost daily, occurrence. This is due sometimes to business requirements, sometimes to emergency changes in response to an incident or problem requiring immediate action to restore service or prevent service disruption, and sometimes to an expedited change that must be implemented in the shortest possible time for business or technical reasons. Unfortunately, Change Management often fails to handle changes quickly in a uniform way that has the lowest possible impact on the networks and its services. However; all changes have a disruptive potential for the business, and controlling change through an agreed change management process is critical. Change management can be even more effective in reducing service disruptions when coupled with the central thesis of this research. If change is communicated immediately to the stakeholders via a notification system and violations to any policy are immediately reported to the change initiator, then the impact of change can be both assessed and controlled.

2.4 Policy-Based Network Management

As already noted, the considerable growth of computer networks has produced significant scalability and efficiency limitations to the traditional management techniques. The tendency to use diverse management and Operational Support Systems (OSS) that are not tightly integrated together has encouraged the use of 'stovepipe' applications, which are applications that maintain their own definition of data that cannot be shared with other stovepipe applications ([11], pp. 4-6). This means that management is often fragmented and intensely human-driven. The need to manage large networks and services efficiently and with speed has given rise to the idea of Policy-Based Network Management (PBNM).

The concept of using Policy-Based Network Management (PBNM) to reduce the complexity of the management task has been researched in the Policy Framework Working Group, the Resource Allocation Protocol Working Group, the IP Security Policy Working Group of Internet Engineering Task Force (IETF), and the Distributed Management Task Force (DMTF) [22]. The PBNM concept is comprised of policies which can be processed by automated systems. The resulting policies are rules governing the choices in behavior of a set of network elements and network conditions, which trigger the policy executions [23]. Therefore, policy-based network management (PBNM) is a condition-action-response mechanism which provides automated responses to changing network or operational conditions based on pre-defined policies [24]. The expectation for

PBNM from a business point of view was to make the management task of establishing and deploying policies across a group of devices a relatively easy one, and to save on critical time and IT resources. From a marketplace perspective, vendors saw an opportunity to unite performance management, service level management, configuration management, and service provisioning in one offering, a need that has challenged IT organizations [25].

In the past decade, policy-based network management (PBNM) technology has matured to the point that it is considered as a feasible approach for the management of distributed systems and networks, and it also has seen several significant standardization efforts to define the most important policy-related concepts and languages [26]. For example, the Internet Engineering Task Force (IETF) has standardized the Common Open Policy Service (COPS) [27] protocol and Policy Information Bases (PIBs) [28], which specify policy objects manipulated by COPS. COPS has also been extended through the definition of COPS for policy provisioning (COPS-PR) [29]. Two other protocols that originally were defined outside the PBNM world have also been considered as a policy provisioning protocol [30]: The Simple Object Access Protocol (SOAP) [31] and Network Configuration Protocol (NETCONF) [32] have been standardized by the IETF to provide proper support for device configuration. The Internet Engineering Task Force (IETF) and Distributed Management Task Force (DMTF) have defined four major functional elements for a policy-based management system: A Policy Management Tool, to enable an entity to define,

update and optionally monitor the deployment of Policy Rules; a Policy Repository, to store and retrieve Policy Rules; a Policy Decision Point (PDP), which is the point at which the policy decisions are made, and the Policy Enforcement Point (PEP), which represents the component that always runs on the policy-aware node and is the point at which the policy decisions are actually enforced ([33], pp. 58). The Common Open Policy Service (COPS) allows the exchange of policy information between a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP) [34].

In essence, the policy-based networking framework allows network operators to express their business goals as a set of rules, or policies, which are then enforced throughout the network. The architecture allows such rules to be defined centrally but enforced in a distributed fashion. In addition, the goal of policy-based networking systems is to allow for the automation of manual tasks performed by network operators ([33], pp. 70-73).

For networks consisting of various network elements from different vendors and multiple systems converged into one network, Policy-Based Network Management (PBNM) is a priority in order to solve this management dilemma [34]. In particular, policy-based management provides a way to allocate network resources, primarily network bandwidth, QoS, and security, according to defined business policies. The success of management depends on the specification of unified and scalable administered policies. These policies must then map to the configuration of the multiple heterogeneous system devices,

applications and networks, for the purpose of policy enforcement [35]. However; the main challenge facing the deployment of PBNM systems is the variety of policy representation forms at different levels of the hierarchy. High-level business policies may be defined and stored in a database system, then various applications may retrieve and convert the data to different forms for processing. These conversion procedures add complexity to the internal structure of PBNM systems, leading to efficiency and interoperability concerns [36].

2.5 Network Configuration Protocol (NETCONF)

The management protocols that we have discussed thus far have their limitation in the context of configuration management of a large number of networked devices with diverse vendor-specific interface and proprietary command line interfaces (CLIs), making it costly to achieve a high level of efficiency and reliability through automation. In 2003, the Internet Engineering Task Force (IETF) started an effort to develop and standardize a network configuration management protocol, which led to the publication of the Network Configuration (NETCONF) protocol RFC4741 [32]. The NETCONF protocol provides mechanisms to install, manipulate and delete the configuration of network devices. It also can perform some monitoring functions. It uses Extensible-Markup-Language- (XML) based data encoding for the configuration data, as well as to send and receive information between managers and agents. The NETCONF protocol operations are realized on top of a simple Remote

Procedure Call (RPC) layer ([9], pp. 275). This in turn is realized on top of the transport protocol.

Figure 3 below shows the four conceptual partitions of the NETCONF protocol: The transport protocol layer provides a communication path between the client and server. The RPC layer provides a simple, transport-independent framing mechanism for encoding RPC requests and responses. The operations layer defines a set of base operations invoked as RPC methods with XML-encoded parameters for the proper handling of NETCONF operations present within requests and of reply content present within responses. The content layer provides the NETCONF protocol with a mechanism for encapsulating configuration data ([12], pp. 83-86).

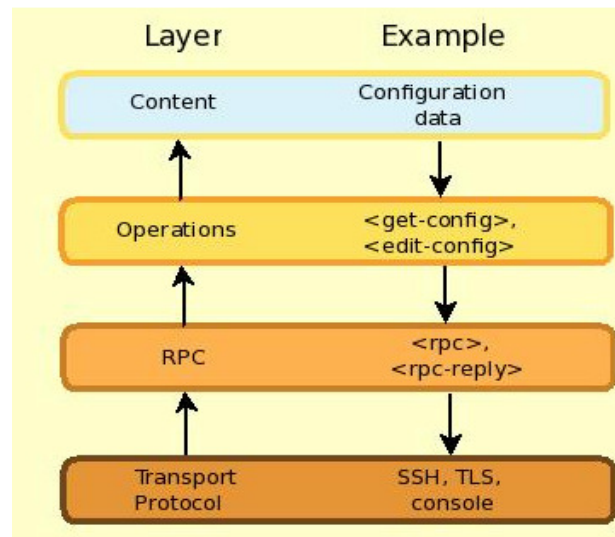


Figure 3 Layers of NETCONF[37]

In a recent NETCONF interoperability test [38] and aimed at observing the compliance of NETCONF implementations with RFC 4741, as well as at

identifying inconsistencies in the RFC, the following test suites produced the results in Table 3 [39]:

1. **GENERAL:** It includes test cases for individual operations such as lock, unlock, close-session, kill-session, discard-changes, validate, and commit.
2. **GET:** This suite aims to test the filter mechanism of the get operations.
3. **GET-CONFIG:** This suite aims to test the filter mechanism of the get-config operations.
4. **EDIT-CONFIG:** Involves tests modifying the configuration data in the datastore.

Table 3 Test Results by Test Suite[37]

Test Suite	Success	Failure	Irrelevant
GENERAL	73.6%	13.2%	13.2%
GET	29.5%	52.3%	18.2%
GET-CONFIG	48.4%	14.1%	37.5%
EDIT-CONFIG	38.3%	1.7%	60%

- Success column: Indicates the percentage of passed test cases.
- Failure column: Indicates the percentage of failed test cases.
- Irrelevant column: Indicates the percentage of test cases that cannot be applied to a specific system due to either system configuration or implementation issues [38].

In summary, NETCONF is a promising building block in network configuration automation and a promising alternative to SNMP with respect to the configuration of network devices. However; the currently open-ended format of request and response messages and their arbitrary values for attributes as specified in the RFC are leading to interoperability problems between different NETCONF implementations [38]. Furthermore, the protocol does not provide any mechanism to ensure that system and device configurations remain compliant with internal or external compliance policies [39].

2.6 Configuration Auditing and Policy Compliance

The preceding sections illustrated how current network management systems fail to help network providers face the challenge of running their

networks without service disruption in the presence of constant network change. Configuration auditing aims to verify that the configuration of any network element complies with the stated policy of the device, and that the information about the network is current. Without this function, network administrators and stakeholders would have a very hard time understanding what is happening in a network and why. The process is important because it can lead to isolating network troubles that occur due to discrepancies between what is currently configured and what should or should not have been configured on the managed entity. Obviously, manual auditing of individual device configuration for networks with a few devices is an option. However, for a large network this option simply does not scale. While this research aims to present an automated way for administrators to identify discrepancies and misconfigurations and hopefully avoid potential catastrophic service disruptions and other adverse fallouts, we will also discuss the current tools used by large service providers.

Configuration audit tools are broken down into functional and physical configuration types. A functional configuration audit ensures that functional and performance attributes of a configuration item are achieved, while a physical configuration audit ensures that a configuration item is installed in accordance with the requirements of its detailed design documentation ([12], pp. 93-94). There are many such products available today. some geared for servers and workstations, others more focused on network elements such as routers and switches. A product by OPNET called 'IT Sentinel' [40] deals specifically with

network change and configuration management. This product has its own language that allows network administrators to present their configuration in a unique format to be stored as a master template for a given device. The system then collects the configuration from the active device every predefined interval. Then the working configuration is compared to the master template, and as a result, a report detailing discrepancies is generated for authorized users to retrieve from a centralized location. Figure 4 below shows an example report that has found 435 errors, 4 warnings, and 12 notes.

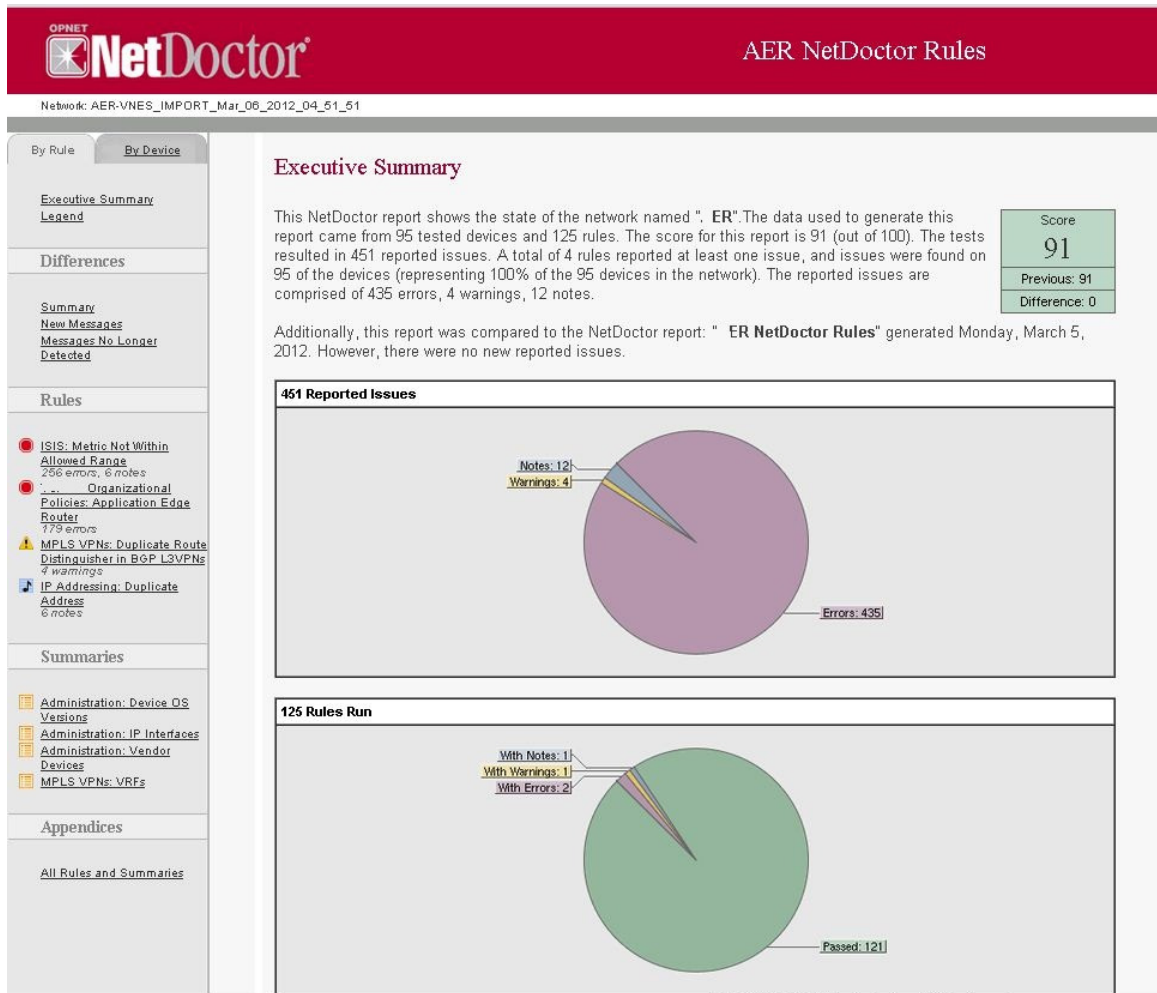


Figure 4 Configuration Audit Report

Using the navigation pane, the administrator is able to drill down to the individual errors, either by device or by rule.

Ecora's 'Configuration Audit and Audit Professional' [41] is another configuration and change reporting product for identifying and auditing configuration settings. This product can audit changes in operating systems, database management systems, applications, directories, network devices and

firewalls. It is also designed to collect configuration data and then compare it to an existing master template. There are many other tools and products that can be used by network managers to verify that the configuration of any device complies with standards defined by the organization. Some include vendor-specific tools and utilities, while others are very expensive. The limitation of such products is that they are not interactive. For example, in a network with over 1600 devices it takes OPNET 'IT Sentinel' over 10 hours to collect and parse through the configuration. This means the tool cannot be used as a reliable source to isolate network troubles in real-time, and therefore cannot be relied upon to help prevent service disruptions due to desired, undesired, accidental, malicious, intentional or unintentional configuration changes.

The preceding discussion illustrated how the frequency of network device changes could potentially be a disruptive factor in an already complex and challenging networked universe. Security issues caused by non-compliant configuration changes or regulatory noncompliance are also some of the reasons for the evolution of network configuration and management tools. These tools are helping to reshape network management towards a more process-aligned discipline that includes supporting service integrity and service performance, minimizing risk, optimizing security and compliance, managing network assets more holistically and achieving operational efficiency [42].

In addition to the tools discussed in this paper, there are many products available today with capabilities to integrate dynamic audits of network

configuration changes with service performance and infrastructure optimization, as well as compliance, security and other initiatives. However, they lack the ability to provide audits in real-time.

What the industry needs is an interactive system with real-time reporting. It is very desirable to avoid waiting hours or even minutes to discover that a change made on a network element has violated a policy and could potentially impact service. Such a system can reduce hours of troubleshooting and save companies from expensive service interruptions. The aim of this research is to describe the framework and all the pieces that are required to implement such a system.

CHAPTER 3: CONTRIBUTIONS

The ramifications of one small change to a network device, whether the change is desired or not, can be catastrophic. However, for large networks, constant change is simply a fact of life. Our proposed Automated Policy Compliance and Change Detection System can reduce these risks significantly. This chapter presents a detailed explanation of the design that will ensure that device configurations remain compliant with internal and regulatory compliance policies.

From a high-level point of view, the protocol will operate as follows: once activated on a given device, the device will check to see whether it has the most current policy by comparing its local policy version number to the server's. If the device does not have a policy, or its version is older than the server's, it initiates a connection to the policy server and requests and downloads the current policy. Once the policy is obtained, the device enters the policy enforcement state and checks to see whether the current configuration state violates any of the rules set in the policy. In the case of any inconsistencies or violations, the Runtime Compliance Manager (RCM) module generates an alarm describing the findings. If none is found, the system enters the monitoring state. In the monitoring state, the module continues to monitor if any newly-entered configurations violate the

policy. If so, the policy enforcement module triggers an alarm; otherwise it continues the monitoring. **Error! Reference source not found.** illustrates the concept.

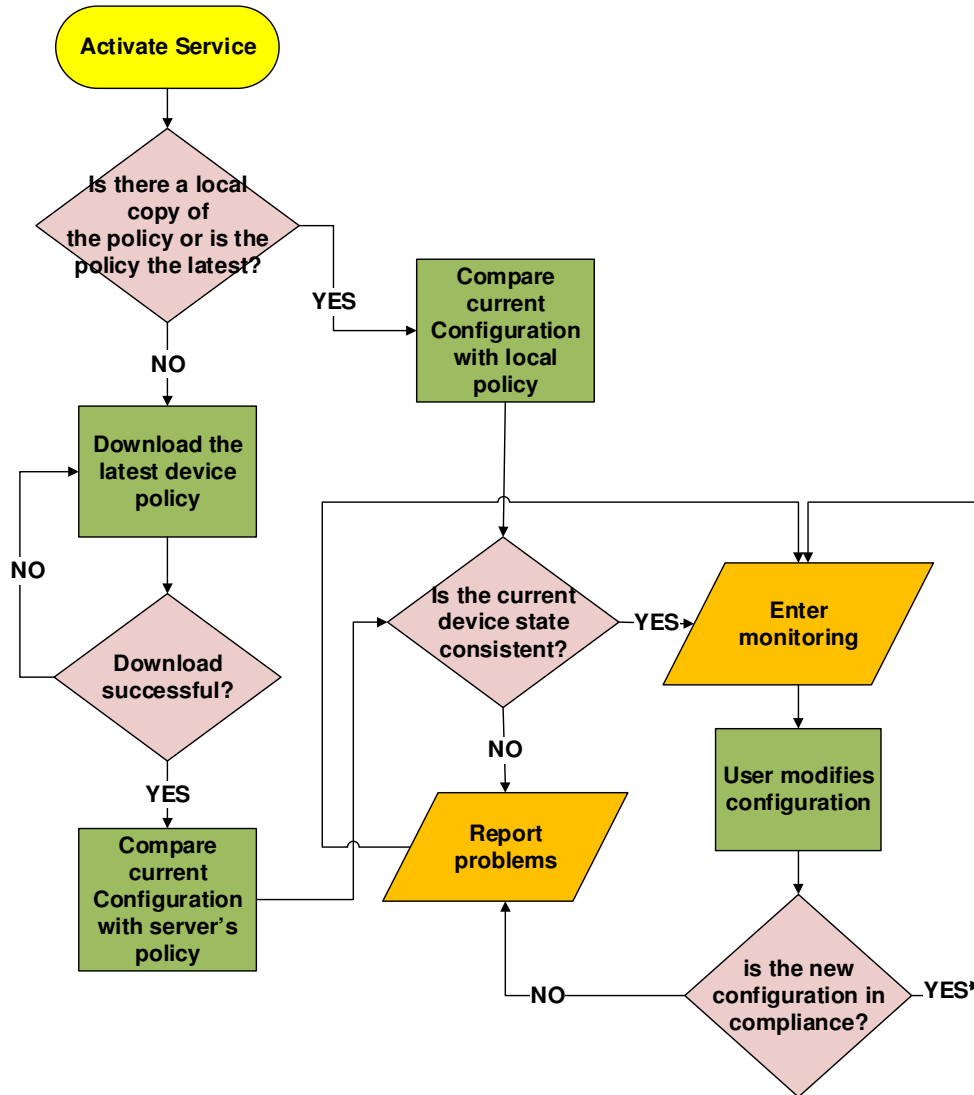


Figure 5 Process Flow

3.1 Policy Exchange and Management

For our proposed policy management system, we adopted the concept of roles from the Policy-Based Network Management (PBNM) framework proposed by the IETF (Internet Engineering Task Force) [43]. The greatest benefit of using the PBNM framework in our work is that it provides automation of network configuration by using the concept of roles. In the context of the proposed Policy Compliance and Change Detection System, a role is an administratively-specified characteristic of a managed element. It is used as a selector for policy rules to determine the applicability of the rule to a particular managed element.

The Policy Management System Framework is illustrated in Figure 6. The Policy Management System consists of the policy management server, the policy decision control server, the policy client, and the policy repository.

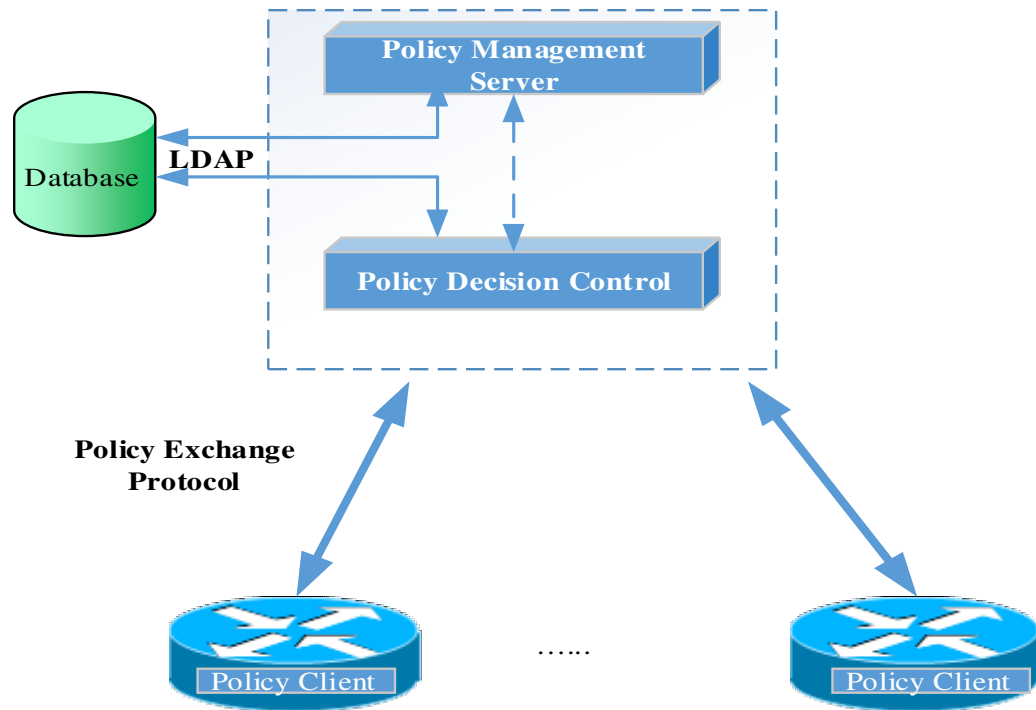


Figure 6 Policy Management System Framework

The policy management server provides a graphical user interface (GUI) for defining, changing and deleting policy information. The policy decision control server has two roles: the first is admission control for the policy clients, and the second is to make decisions based on the policy client role to retrieve the matching policy from the policy repository and distribute it to the client. The policy client is a network element subject to our policy domain. The policy repository is a specific data storage that holds policy rules, their conditions and actions, and related policy data. The architecture assumes that multiple policy systems may exist within a single policy domain, and share the same policy information. Therefore, a secured centralized repository – secured via access

authorization, such that it cannot be accessed unless someone is given direct privileges through a tightly-controlled process - can be used to store, distribute, and coordinate policy information among systems. Directories and LDAP (Lightweight Directory Access Protocol) are the IETF choice for interoperable standard policy storage [44].

The communication between the policy clients and the policy decision control is loosely based on the Common Open Policy Service – Provisioning (COPS-PR) protocol [29]. COPS-PR is an extension of COPS [27]. COPS-PR is a protocol for providing an efficient and reliable means for a policy management server to provision multiple policy clients. It has several features for efficient management, such as event-driven control (i.e., no polling) and asynchronous notification, a structured row-level access and transactional model, support for fault tolerance and security mechanisms, and reliable transport using persistent TCP connections. The protocol is discussed further in the following section.

3.1.1 Policy Client and Policy Decision Control Communication

The first role of the policy decision control server is admission control, which it fills by verifying that the requesting policy client role matches an existing defined policy on our system. A policy role is an alphanumeric string that defines the network element role in the network (e.g., Back Office Switch, Backbone Router). The second role of the policy decision control is to continuously distribute the most current policies to the policy clients. Using TCP, the policy clients stay connected with a single policy decision control server in

order to retrieve updated policies. Each policy client stores internally a policy consistent with its network role. However, since networks and client/server communication interruptions may lead to inconsistent policy states, the COPS-PR defines a set of messages to avoid the synchronization problems that may occur between the policy clients and the decision control server due to events that may occur in the network. These messages are explained next.

When a policy client boots up or after a TCP connection loss, each client opens a new connection with its configured policy decision control server and attempts to establish a policy session by sending its network role to the server, using a Client-Open (OPEN) message. Figure 7 depicts the message exchange. Upon receiving the OPEN message the policy decision control server checks to see if the client role is supported. If not, a Client-Close (CC) message is sent back to the policy client. Otherwise, the policy decision control server replies back with a Client-Accept (CAT) message. If the policy client holds an internal policy from a previous session, the client issues a Request (REQ) message informing the policy decision control server of the version number of the installed policy. The policy decision control server determines which action to take and either sends the newer policy or maintains the same one by replying back with the Decision (DEC) message. If the policy client holds no previous policy, it simply sends a Request message (REQ) and the server sends back a Decision (DEC) message initiating the download process of the policy. Finally, the policy client notifies the policy decision control server of the success or the failure of installing or

maintaining the existing policy by sending a Report (RPT) message. Figure 7 shows the common message exchange between policy clients and the policy decision control servers on boot up or recovery from TCP session loss in a COPS-PR managed network [30].

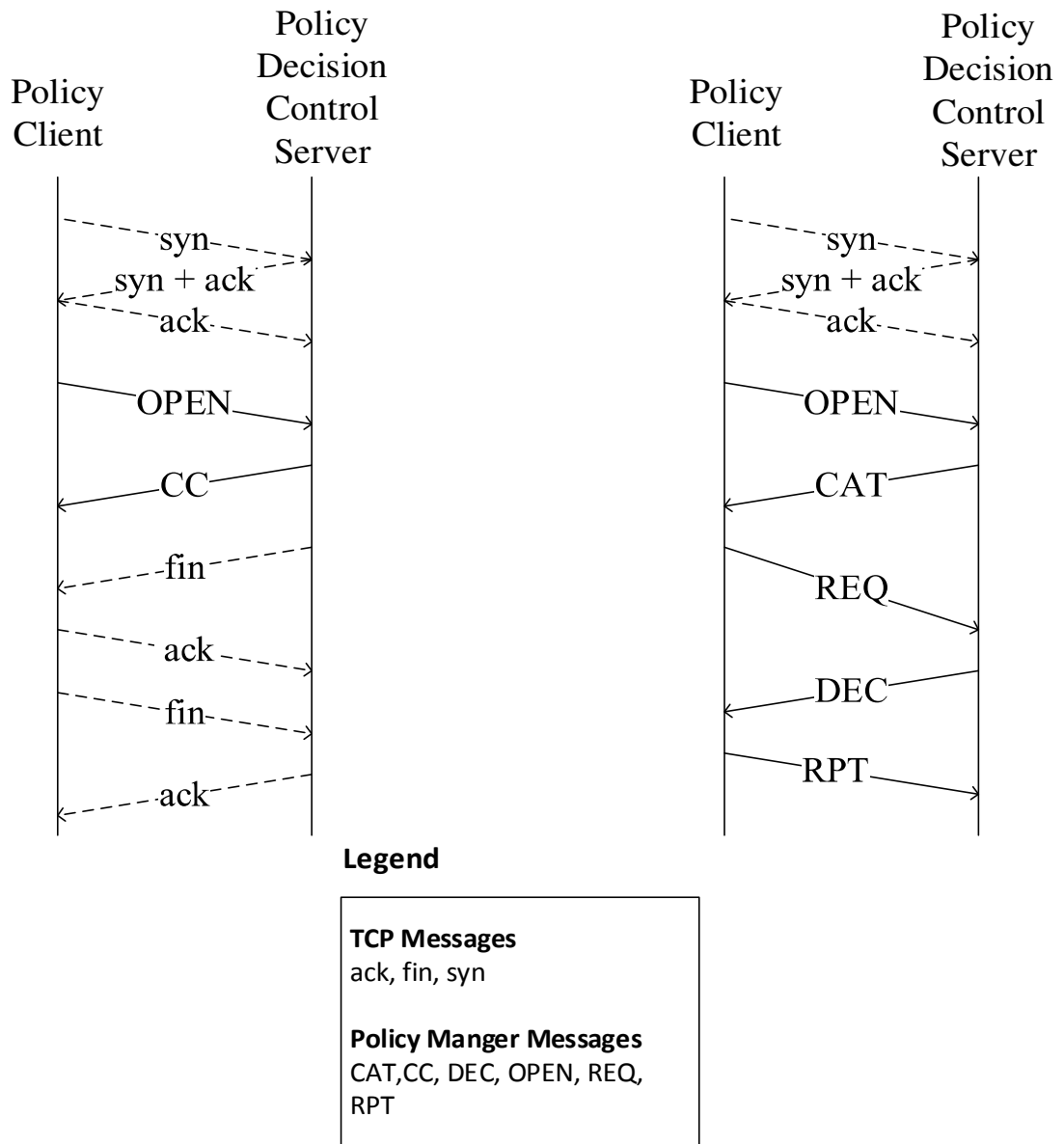


Figure 7 Policy Client and Decision Server Communication

While in the active policy session state, it may be common for the network administrator to modify, change, edit, or delete some elements of an existing policy or even to create a new one via the policy management interface. In such a scenario, the propagation of the new policy is accomplished by initiating a new

unsolicited DEC message from the policy decision control server to all the policy clients whose policies have changed [45]. This process is critical because it allows the network administrator to update one or many policies and with one stroke propagate the new policies to many network elements, even ones with different roles. Figure 8 illustrates the concept.

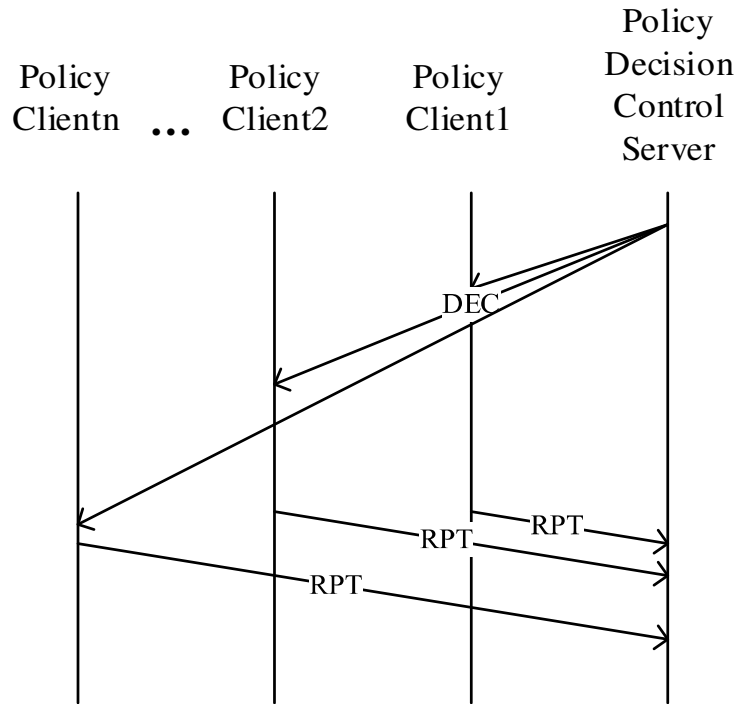


Figure 8 Policy Update from the policy decision control server

We also recognize that any active network element may require ad-hoc changes by the network administrators to continuously conduct provisioning, tune performance, change parameters, etc. These changes may cause the client policy to disagree with the server's version. Therefore, we see an application in which

the policy client should periodically check with the policy decision control server to make sure it has the latest policy. To accomplish this task we define a preconfigured interval after which the policy client sends a REQ message informing the decision server of its state. In return, the policy decision control server replies with a DEC message. Figure 9 depicts the message exchange between the policy decision control server and the policy client, while Figure 10 shows how to implement the routine in software.

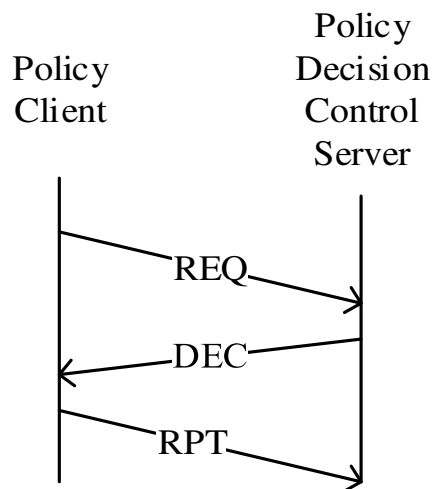


Figure 9 Policy Client Status

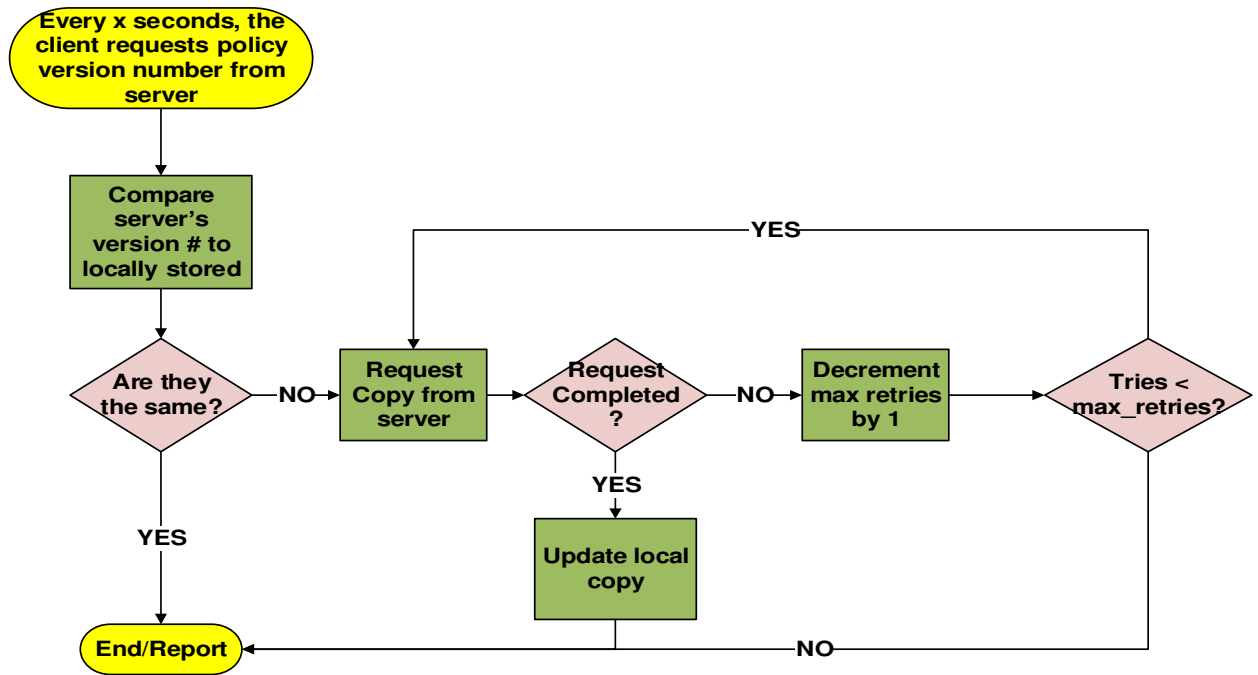


Figure 10 Client Check Process

3.2 Design Considerations

This research also takes into consideration other factors to maintain policy consistency between the policy decision control server and its policy clients, and ways to provide an optimal and reliable system even when the underlying system changes prevail.

We start by highly recommending for the LDAP directories repository to be protected and replicated using industry best practices for Authentication, Authorization, Accounting, and Auditing. As mentioned earlier, our policies will be manipulated via the network management interface and stored in LDAP repository. Therefore, the confidentiality, integrity, and availability of the policies

play a crucial role in the sustainability and success of our proposed system. A discussion on securing LDAP is outside the scope of this research. However, we mention two different authentication methods: 'simple bind' [46] and Simple Authentication and Security Layer (SASL) [47]. Both are specified by the IETF and both support the latest specification of LDAP Version 3 [48].

3.2.1 Masquerading

Masquerading is a common network attack strategy in which the attacker pretends to be someone or some network device which it is not [49]. In our system, both the policy client and the policy decision control server are susceptible to such an attack, whereby the attacker seeks to masquerade as a client in order to obtain the device policy which may include sensitive information, or to masquerade as the policy decision control server and perhaps prevent the policy clients from updating their policies. Other TCP security concerns are also in play, such as session-hijacking, man-in-the-middle, and Denial of Service (DoS) attacks. Although many industry methods have been presented to mitigate the mentioned attacks, we recommend authentication using the message-digest 5 (MD5) algorithm.

The MD5 message-digest algorithm defined in RFC 1321 takes as input a message of arbitrary length, applies some “independent and unbiased” bit-wise operations on the message blocks, and produces as output a 128-bit fingerprint or message digest of the input. With this hashing technique, the conjecture is that it is computationally infeasible to produce two messages having the same message

digest, or to produce any message having a pre-specified target message digest. MD5 is designed to be a fast and compact algorithm ([50], pp. 34-35). The MD5 message-digest algorithm is simple to implement and provides a fingerprint or message digest of a message of arbitrary length. It is estimated that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^{128} operations ([50], pp. 40). The MD5 algorithm is used in many other network security technologies in which authentication and data integrity are needed ([50], pp. 54-55).

Enabling MD5 authentication between the policy decision control server and its policy clients provides added security and protects against spoofing. MD5 authentication allows each policy client to use a secret key to generate a keyed MD5 hash that is part of the outgoing packet. The keyed hash of an incoming packet is generated on the server, and if the hash within the incoming packet does not match the generated hash, the packet is ignored. RFC4086 “Randomness Recommendations for Security” [51] describes a reasonable approach to producing a high quality random key of 96 bits or more.

3.2.2 Communication Interruptions

Earlier in the thesis, we described the recovery process after a TCP session failure, and showed how to get the client policy synchronized with the server’s latest version. However, here we consider a scenario of a network interruption during which a set of configurations has to be made on the network element, and

during which an update is made to the master policy. Our approach is to allow for the policy enforcement of the configuration using the most recent stored local policy. Then upon recovery the policy client opens a new TCP session with the policy decision control server and downloads the updated version from the server using the steps described in Section 3.1.1 Policy Client and Policy Decision Control Communication. Immediately after the policy synchronization process is completed, the RCM module running on the policy client moves to force a full system configuration check against the newly-downloaded policy, and report via syslog any discrepancies found. It is understood that the report may occur a while after the network administrator has departed the network element. In that case, the reporting is expected to be picked up by the network surveillance system, which then notifies the network administrator for further investigation.

Another approach is to lock down the device whose TCP session has been interrupted, and prevent any configurations from taking place until the session is restored. However, there may be many good reasons to allow the configuration on the system, which may outweigh the risk of being non-policy-compliant for a short period of time.

3.2.3 Malicious Interruption

This scenario is similar to what we described above in which the TCP session between the client and the server is interrupted due to a network outage or misconfiguration. Here an authorized rogue network operator maliciously severs the communication to the server by either disabling the policy exchange protocol,

changing the configuration and pointing the network element to a non-existing policy control server, or using any other means to prevent policy enforcement on the device. While we cannot stop an authorized operator from making malicious changes to the policy client, we suggest the following improvements to our system: The policy decision control server should generate an alarm and/or a syslog message whenever it loses a session to one of its clients. This way the network operators and surveillance systems are immediately notified, which should elicit further investigation. Secondly, and as suggested earlier, a full policy check and reporting should be triggered upon recovery. Finally, the system design should allow the network administrator to conduct a full policy check on any given device.

3.2.4 New Policy Push during Configuration Change

In this scenario, the policy client, while in configuration mode, receives a new policy update from the policy decision control server. Since the updated policy may have elements conflicting with the locally stored policy and currently being used for enforcement, we suggest a syslog message informing the operator of the policy update, continued use of the current policy, and, upon exiting configuration mode, a forced full policy check using the newly-updated policy, and furthermore a report of any noncompliant elements of the configuration. The network administrator also has the option to force a full system check, as described earlier.

3.2.5 Periodic Check Detects New Policy during Configuration

This differs from above scenario in that the policy client during a periodic check detects the policy update while in configuration modes. Periodic checks were described earlier in section 3.1 Policy Exchange and Management of this dissertation. The assumption here is that through the exchange of messages between the policy client and the policy decision control server, the client detects a newer policy version and the server in return makes a decision to push a new policy to the client, because the client's reported policy is older than that stored by the server. Again, the safest approach is to report the update to the operator, continue to use the locally-stored policy for compliance enforcement, and upon exiting the device configuration mode, force a full system check and report any noncompliant configurations.

3.3 Common Policy Language

One of the core requirements for our proposed system is a Common Policy Language for expressing device and organizational policies. The automated system relies on the Common Policy Language to represent device, organizational and industry best practices, and any other regulatory policies or guidelines needed for any network element. This section defines some of the building blocks of the proposed policy language in a structured document format that can be retrieved and manipulated with ease. A Common Policy Language will ease the enforcement of policies in all components of the network. Furthermore, the proposed Common Policy Language will bring numerous practical advantages,

such as lowering implementation overhead and the possibility of using the same or at least similar tools to maintain the policies.

3.3.1 Configuration Templates

Given the complexities and challenges of network configuration, an effective policy compliance system does not only define a model to unify all data but also provides a mechanism to support coordinated multi-device configuration. The use of a high level of abstraction to describe the behavior of the network, network-wide configuration and policy management rules, and the ability to map high-level language into low-level, device-local configuration and vice versa, are some of the most desirable features for any management system [52].

The Policy Compliance System relies on the Common Policy Language to represent device parameters and settings in a template format. The common policy configuration template allows for a very straightforward implementation of policy configuration backup and for restoring functionality. Configuration files also make it simple to maintain different configuration versions by simply copying configuration files back and forth. This approach is also well-suited to many similar configurations across the network. The same configuration template can essentially be applied to different devices of the same type across the network, with only minimal processing required to customize the template. For example, the same basic template can be used for all office switches where only the hostname, IP addresses, or VLANs, and so on, are unique.

Altering individual device configurations across a large number of devices can be tedious and time-consuming, and templates save network administrators time by applying the necessary configurations and by ensuring consistency across devices [53]. The configuration template will also be used to maintain the internal and regulatory compliance policies, to be enforced by our automated compliance system.

From a high-level point of view, once the template is activated on a given device, the automated compliance system will check to see whether the current configuration state violates any of the rules set in the policy. If any inconsistencies or violations are found, an alarm is generated describing the findings. If no inconsistencies are found, the system enters the monitoring state. In the monitoring state the protocol checks to see if any of the newly-entered configurations violate the policy, and if so, a report/alarm is generated. Otherwise, monitoring continues.

The term 'template' can have different meanings depending on which programming environment, language, or framework is being used. In this context, the template is a string that can be combined with configuration and policy requirements to produce a working vendor configuration file. In the proposed system, the configuration templates are stored as files on the server, and downloaded to the client via secure file exchange. The templates contain placeholders where client-specific data will go. When the automated policy compliance system engine renders the templates, these placeholders are examined

against the actual existing configurations, or the would-be configuration entered by the system administrator. The system then alerts the user of any policy violation, thus allowing for real-time feedback.

In order to accomplish basic output logic, such as if...then logic, some logic code needs to exist in the template files. To illustrate the concept, Figure 11 expects different values for the name-server depending on whether the logic is evaluated TRUE or False. It is evaluate TRUE if the device hostname is set to R1 and false if set otherwise, and therefore if TRUE the name-server is set to 8.4.4.4 and to 8.8.8.8 if false.

```
1 %condition-start Host
2 hostname R1
3 %condition-end
4
5 %if-start Host
6 ip name-server 8.4.4.4
7 %if-end
8
9 %if-start Not_Host
10 ip name-server 8.8.8.8
11 %if-end
```

Figure 11 if...then Logic

The syntax of the template language is intentionally clean, simple and elegant. With minimal understanding of programming concepts, a system administrator and other stakeholders can make powerful and flexible templates to

represent their organizational policies or an acceptable baseline device configuration.

3.3.2 One- and Two-Phase Commit Models

Considering the landscape of networking devices and vendors in the networking field, we found that many of the network devices adhere either to the one-phase commit model or to the two-phase commit model.

In the one-phase commit model, each line of configuration that enters the networking device takes effect immediately. In contrast, the two-phase commit model breaks the process into two distinct stages. In the first stage, the system administrator builds the targeted configuration on the given device, then checks the configuration for both syntax and transport errors, ensuring that the configuration entered the device successfully. In the second stage, the user is able to commit the configuration into the networking device, making it part of the working configuration [54]. The discussion of the benefits and disadvantages of either model is outside the scope of this paper. However, for our paper we examined solutions from two of the leading enterprise networking vendors, Cisco Systems and Juniper Networks [55].

Cisco Systems networking devices operate on Cisco IOS (originally called Internetwork Operating System) or Cisco IOS-XR. The Cisco IOS operating system uses a one-phase commit model, and has monolithic architecture, which means that it runs as a single image and all processes share the same memory space. The Cisco IOS XR operating system uses the two-phase commit model and

has a micro-kernel architecture, which provides basic operating system functionalities including memory management, task scheduling, synchronization services, context switching, and interprocess communication (IPC) [56].

Juniper Networks uses a modular software architecture that provides highly available and scalable software. JunOS is a FreeBSD-based operating system that runs a single code base across most of Juniper's routing, switching and security devices. JunOS uses a two-phase commit model [57].

In the following sections, we will use both Cisco's and Juniper's operating systems to represent our Common Policy Language.

3.4 Common Policy Language Format

This section describes the common policy language format for expressing device and organizational policies. Using a common language brings numerous practical advantages, such as lower implementation overhead, as well as the ability to reuse sections of existing policies with other devices, and even the ability to link multiple policies together. The language format is written in ASCII text, and is essentially the same, except for few minor differences between the OS types of the supported vendors. We will use Cisco's IOS and Juniper's JunOS to illustrate the Common Policy Language format.

3.4.1 Mandatory Sections

The template will start with two mandatory sections. First we define the template version number. This number is used by the policy decision control

server to compare the device version to that stored on the server, and the higher version of the two is used to enforce the configuration template. If the device has a lower number, a secure file fetch procedure is executed and the newer file replaces the existing device template file. The version is a context variable that holds the version value. We use the percent sign (%), preceded by the keyword version, followed by a numerical value of positive integers or decimal numbers, to express the version value. Example:

```
%version 2.1
```

We also recognize that the same type of hardware could be used for different roles within the same organization. For example, a layer 3 (L3) switch could be used as layer 2 (L2) device only in the access layer, and as router (L3) in the distribution layer. Therefore, the chassis should not dictate the configuration template, rather the device role itself should be the indicator of what the template should and should not contain. Hence, we use the variable device to determine to which device role the template applies.

```
%device <fixed/Regular expression>
```

The value of the device variable could be a fixed value, such as a device role defined by the network administrator, or the value could be expressed using a regular expression. Below are two examples:

```
%device office-switch
```

```
%device regex(office-switch1[0-9])
```

In the example, office-switch is a fixed value name, and regex (office-switch1[0-9]) matches any device whose name falls between 'office-switch10' and 'office-switch19,' inclusive.

```
%device regex([A-Z]+?-SWITCH-[\d]+)
```

The above example matches any single character in the range between A and Z (case sensitive), and repeated between one and unlimited times, followed by an exact match of -SWITCH- and followed by a one-to-unlimited match of digits between 0 and 9. WASH-SWITCH-01 and TAMPA-SWITCH-99 are examples of string matching the regular expression match.

3.4.2 Section Delimiters and Predefined Keywords

We use the keyword section as a delimiter of a configuration section. The section may match part of the configuration stanza, such as system, interface, routing, or syslog settings. The section could also be a user-defined. The section starts with the keyword `section-start`, and ends with `section-end`. Furthermore, our predefined keywords: `ignore`, `exclude`, and `exact` are used as follows:

- `Ignore`: specifies commands or sections that should be ignored when our policy compliance system compares the device configurations against the template. The command is useful for writing a comprehensive device template while allowing certain sections of the policy to be ignored by the Runtime Compliance Manager.

- **Exclude** : specifies that the sections or the commands should not exist on the device. This is useful when the operator wants to guarantee that a certain block of configuration does not exist on the examined device.
- **Exact** : specifies commands or sections that should be identical between the template and the policy client configuration. This is useful when we expect to see an exact configuration match between the device template and the actual configuration.

To illustrate the above concepts, we give the following JunOS example:

```
%section-start ignore
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 192.168.1.1/32 {
                    primary;
                }
                address 127.0.0.1/32;
            }
        }
    }
}
%section-end
```

In the above example, the entire loopback0 configuration is ignored by the system when enforcing the template to the device. We extend another example, using Cisco IOS this time, to illustrate the use of the keywords 'exclude' and 'exact:'

```
router bgp 1008
  no synchronization
  %section-start exact
  bgp router-id 8.8.0.1
  %section-end

  bgp log-neighbor-changes
  network 8.8.0.1 mask 255.255.255.255
  neighbor 8.8.0.6 remote-as 1008
  %section-start exclude
  route-map STATIC-TO-OSPF permit 10
  match ip address prefix-list STATIC-TO-OSPF
  %section-end
```

In the above example, the router-id was specified with the 'exact ' keyword, therefore when enforcing the policy, the system must have a matching router-id. However; the example excludes the route-map command. Thus, if the command exists on the client, the system would consider it a violation of the policy.

Additionally, there are two kinds of comments we can use in our templates: single-line and multi-line. Writing comments could often be as important as writing the system policy itself. Even though the comments left in the policy will be ignored upon execution, it is important to let others know what you are doing, because even the best policy may need to be maintained or updated by someone else. The comments let other administrators understand what the author intended in each step. This makes it much easier for them to work with it, and to edit it if needed. Comments can also remind the policy author of what he or she did when it is time to edit the policy a year or two later. For our Common Policy Language, single-line comments are identified by a right slash and an asterisk (/ *):

```
/* this is an example of a line comment
```

Multi-line comments are implemented by using the right slash and the asterisk (/ *) and end with the asterisk followed by the left slash (* \). Anything that falls between the delimiters is considered a comment:

```
/* this is an example of a multiline comment.
```

```
It is important for the compliance and change
detection portion of the system to provide sufficient
information about differences between the active
configuration, or the newly-entered configuration, and
that specified in our policy for that device. * \
```

In this example, the template engine ignores everything between the '/'* and '*\' tags. In addition to providing for explanations in the template section, this technique could also be used to troubleshoot and debug a section of template that is not behaving properly.

Here is another practical example of the use of comments: the multiline comment gives the reader an indication of the purpose of the template and of the latest changes and the previous changes, while the one-line comment is a specific instruction about the NTP command.

```
/* Application Ethernet Switch
   version cisco WS-C3750G-24TS 2.6
   Modified: 11/25/2014
   Modified By: Saeed Agbariah

Latest Change(s) :
1. Updated NTP servers
2. Updated DNS servers

Previous Change(s) :
1. changed VTY password
2. changed syslog buffer size
*\

/* DO NOT TYPE ntp clock period <seconds>;

ntp source FastEthernet0
ntp server 172.30.127.1
```

3.4.3 Conditional Commands

With the conditional commands, we can create an expression to be considered conditionally during the comparison process. Conditional expressions perform different computations or actions depending on whether a specified Boolean condition evaluates to true or false. This allows our compliance system

to respond differently to different values and inputs, which allows for manipulation that is far more advanced and for a far wider range of possible behaviors. In its basic operation, the system checks to see if the expression is TRUE and then performs comparison within the policies configured within the defined configuration block. If the expression is not TRUE, it skips the condition.

To build the conditional statement we first define the condition. This can be done anywhere in the template, although it may be better practice to define the condition in the configuration stanza where it will be used.

The condition is defined by using the `condition-start` followed by a unique name within the template, and ends with `condition-end`. The condition is expressed with a string, or by using regular expression. It can have a single value, or it can have multiple values, each on a separate line. For the condition to be true, all expressions must match the device settings:

```
%condition-start Ethernet
    interface ethernet 0/0
    ip address 192.168.1.1 255.255.255.0
%condition-end
```

In the above example, our condition name is Ethernet, and for the condition to be true, the examined device must have interface Ethernet 0/0 and it also must have a 192.168.1.1/24 IP address configured. Next, we will illustrate the use of regular expression:

```
%condition-start MyDevice
```

```
snmp-serverlocation regex(Fairfax|Washington)
%condition-end
```

The MyDevice condition is true if the examined device SNMP location is either Fairfax or Washington.

3.4.4 If-Then

When the Runtime Compliance Manager (RCM) module finds an *If* that matches the condition from the previous section, it expects a Boolean condition. If the evaluated condition returns true, then policies defined within the *if* block are compared to the device's configuration. The conditional section is identified by *if-start* and the terminator is identified by *if-end*. The negation operator ('Not') returns the opposite of the given Boolean expression. The following example shows the use of the *if* statement:

```
%condition-start Ethernet
    interface ethernet 0/0
        ip address 192.168.1.1 255.255.255.0
%condition-end

%if-start Ethernet
    NTP SERVER 192.168.1.253
%if-end

..

%if-start Not Ethernet
    NTP SERVER 192.168.1.254
```

```
%if-end
```

In the above example, if the condition Ethernet is true, then the policy compliance system expects the policy client in evaluation to have the NTP server set to 192.168.1.253. If false, then the NTP server should match 192.168.1.254.

3.4.5 Variables and Parameters

A router offers many levels of configuration modes, allowing the configuration to be changed for a variety of router resources. Global configuration mode, for example, allows commands that affect the router as a whole, while interface configuration mode allows commands that configure router interfaces. There are many other configuration modes, depending on what is being configured ([58], pp. 25-30). In this section, we address the need for variables to ensure device-specific command variables are presented and compared properly by our automated compliance system against the device of interest.

For example, our devices may be configured with a Loopback0 interface, each with its own unique Loopback0 address. Loopback interfaces are virtual in nature and can be used as termination points for protocols such as BGP. Loopback interfaces can also be used to provide a known and stable ID for the OSPF routing protocol, or whenever data needs an intermediate output interface, such as for address translation ([58], pp. 70-77). Our Common Policy Language addresses variables such as the Loopback, by defining a global parameter with a unique name across the configuration template, which has a single value. The global

parameter is contained between percentage signs (%) and starts with the keyword `global-single`, followed by a unique name. Here is an example:

```
interface loopback0
    ip address %global-single LoopBack%
255.255.255.255
router ospf 8
    router-id %LoopBack%
    network %LoopBack% 0.0.0.0 area 0
```

When the automated system compares the template to the device of interest, in the example above, the `LoopBack` variable uses the value of the configured IP on the device. Then the policy checks to see if the same variable is configured for the `router-id` under the OSPF process, and also for area 0. Next, we show a similar example for a Juniper device to illustrate that the template is essentially the same, except for a few minor differences between the OS types of the supported vendors.

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address %global-single LoopBack%/32
            }
        }
    }
}
```

```

}

routing-options {
    router-id %LoopBack%
}

protocols {
    bgp {
        Group ebgp
        local-address %LoopBack%
    }
}

```

The same variable Loopback is used for loopback0, router-id, and BGP source address.

A mismatch is declared and the user is informed if, in the example above, the configured subnet for the Loopback0 interface is not 32 bits.

The second parameter type that our Common Policy Language defines is the global-list parameter. The global-list parameter allows for a set of variables to match against the device configuration. We define the parameter with global-list contained between percent signs (%) followed by a unique name. We illustrate the use of the list variable by the following example:

```

snmp {
    clients {
        %global-list snmp_server%/32;
    }
}

```

```
    }  
}
```

In this example, the global-list variable accepts a value of at least one client. If the actual device being examined has the configuration below, which has three SNMP clients, then the configuration is considered correct.

```
snmp {  
    clients {  
        192.168.10.12/32;  
        192.168.10.13/32;  
        192.168.10.14/32;  
    }  
}
```

As another example, using Cisco IOS, the device below is configured for TACACS authentication and the administrator has configured a primary and a secondary address. First we show the configuration template and then the actual device configuration:

```
tacacs-server host %global-list tacacs-server%
```

Actual device configuration:

```
tacacs-server host 192.168.10.66  
tacacs-server host 192.168.11.66
```

Our third parameter type is called 'related.' The related variable name does not need to be unique across the configuration template but should be unique

across the same parameter type in the configuration command. In addition, the related parameter is always associated with the first global parameter that follows. The related parameter is contained within the percent signs (%), and again we use an example to illustrate the concept:

```
interface loopback0
    ip address %global-single LoopBack% %related
subnetmask%
```

Using the earlier Juniper example, we change the 32-bit subnet mask to the related variable subnet mask:

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address %global-single
LoopBack%/%related subnetmask%
            }
        }
    }
}
```

In both above examples, the related variable subnet mask is associated with a global-single parameter named Loopback. Our system check would not fail if the subnet mask is not 32 bits long, as it did in the earlier example.

3.5 Summary

This chapter first presented a methodology to design and implement a mechanism to maintain and exchange network element policies based on their network roles. The policy exchange is based on the client/server model. The policy decision control server represents the server and is responsible for client admission and the distribution of policies. The client is represented by any data network element, whose policies will be used by the Runtime Compliance Manager (RCM) to enforce against the actual device configuration or any new configuration, and to report noncompliance to network surveillance systems in real-time.

The chapter also showed various types of messages exchanged between the Policy decision control server and the policy client. It explained the admission control process by the policy server and presented how in normal operations the policy file is exchanged and maintained between the policy decision control server and the policy client. We also considered a few scenarios where the locally-stored policies on the individual network elements could be rendered outdated, and described best practices to handle such anomalies. We also described the Runtime Compliance Manager operation that continuously monitors and analyzes the state of a device, in order to keep it in compliance.

The chapter also presented some of the building blocks for a Common Policy Language (CPL) to be used with our automated compliance system. The system is capable of providing real-time auditing and ensures a consistent

configuration state. It can guarantee compliance and reduce outage minutes. The language format is written in ASCII text, is portable and powerful, is remarkably easy to manage and manipulate, and is essentially the same for many supported vendors. The Common Policy Language (CPL) is easy to understand, and is very similar to your device configuration structure, but with more powerful programming language functions, statements and operations.

In designing the Common Policy Language (CPL), our focus was on readability, coherence and ease of use in modules. The language is intended to be readable, and hence reusable and maintainable. The uniformity of the Common Policy Language code makes it easy to understand, even if the administrator did not write it. The ability to customize the Common Policy Language enables improvements to best practices. Further work may include defining more parameters and roles to address a wider range of configuration diversity between vendors. The flexibility in the language format leaves little danger of being locked in by a vendor.

CHAPTER 4: IMPLEMENTING AND TESTING POLICY EXCHANGE AND MANAGEMENT

The purpose of this research is to describe a method for an automated change detection system in which a policy-based management system is able to prioritize and manage risks, audit configurations against internal policies or external best practices, and provide centralized reporting for monitoring and regulatory purposes in real-time. The goal is to avoid any potentially-disruptive factors in an already complex and challenging networked universe where changes may lead to configuration errors, policy violations, inefficiencies, vulnerable states and security threats through faulty or non-compliant configurations.

One of the core components of our proposed system is the policy exchange system. The policy exchange system allows a network administrator to create a policy or manipulate an existing policy from a centralized location and then propagate the policy to a device or set of devices in the domain, based on their role in the network, in a convenient way without needing to manage each device manually. The policy exchange system also addresses methods for keeping stale policies out of the network and ensures that the clients always have an up-to-date replica of the policy. Finally, the proposed system addresses security concerns in the policy exchange process between the client and the management server.

In design of the client/server protocol for the policy exchange system, we had Cisco's Internetwork Operating System (IOS) and Juniper's Operating System (JunOS) in mind. In this research, we used many examples in the Common Policy Language (CPL) section utilizing both Cisco's IOS and Juniper's JunOS. But unfortunately, because the operating system is proprietary, as it is for many other vendors, we chose an open system - Linux-based - to implement and test our code. Linux is a Unix-like and mostly POSIX-compliant operating system assembled under the model of free and open-source software development and distribution [59]. One of the main advantages of using open source software like Linux for our research is customizability. Since the code is open, the code can be modified to fit the functionality we want for our policy exchange system. Interoperability is another advantage, because the open source software tends to be much better at adhering to open standards than proprietary software [60].

4.1 Motivation and Problem Summary

The policy client/server is a lightweight but very powerful system that is used to centralize the policies of all the clients. It uses a modified version of COPS-PR (RFC3084).

Putting this system in simple words, there is a server running with several text readable policy files (.txt files); and each time a new client establishes a TCP connection with the server, it reports its client role to the server. The server then finds the suitable policy file for the client and makes sure that the client receives

the latest version of the policy. Or it rejects the connection if the policy client role is not defined on the policy decision control server.

The client is kept up to date by keeping the TCP connection alive, and for any update made on the server side that affects the behavior of a client, the server sends an unsolicited message with the new policy file to the client.

Furthermore, the client has the option of a timer configuration, which allows the client to send periodic and customizable requests to check with the server whether the policy it has is still the latest.

Additionally, since the information contained in the policy file may contain mission critical information, our proposed system is equipped with an MD5 hash function. Thus, every packet exchanged has an MD5 hash attached to it, to avoid undesired tampering and unauthorized policy exchanges. The system is designed for administrators to manage the clients' policies from a centralized location and distribute the policies in a secure manner.

4.2 System Requirements and Installation

The system is written in C/C++ for UNIX-like environments. Since it uses only standard C++ library, the requirements are just a few:

- **Unix-like system:** a multitasking, multiuser computer operating system that exists in many variants [61]
- **G++:** a compiler, a program that will take your C++ source code and compile it into a binary file that can be executed to actually run your program [62][62].
- **Bash script:** a Unix shell, a command processor that typically runs in a text window where the user types commands that cause actions. Bash can also read commands from a file, called a script [63].
- **Make:** a utility that automatically builds executable programs and libraries from source code by reading files called makefiles, which specify how to derive the target program [64].
- **Tar command:** (tape archive command) used to rip a collection of files and directories into highly compressed archive files commonly called tarball or tar, gzip and bzip in Linux [65]
- **md5sum Unix command:** a computer program that calculates and verifies 128-bit MD5 hashes, as described in RFC1321 [66]. A high level description of the source code is provided in Appendix A of this dissertation.

4.2.1 Server

Once it is compiled, the server can be started by from the server's folder by executing the following command:

```
./PolicyServer <Port> <Secret Key>
```

Example: `./PolicyServer 12345 SeCrEtKeY`

The server software requires two input parameters, and they are explained next:

<Port>: TCP port in which the server is going to start listening to new clients.

<Secret Key>: The secret key used in the MD5 hash function to secure the server-client communication.

After the server has sent the Client-Accept (CAT) message to the client, in the event of policy modification, the policy decision control server sends an unsolicited Decision (DEC) message and pushes the modified or new policy to the client. Alternatively, if the client is configured with a periodic check timer, it waits for a Request (REQ) message from the client. Upon the receipt of the Request (REQ) message, it checks to see if the reported client version number of the policy matches the server's, and either sends a Decision (DEC) message with the newer policy file, or sends a Decision (DEC) message without the file. Figure 12 depicts the process.

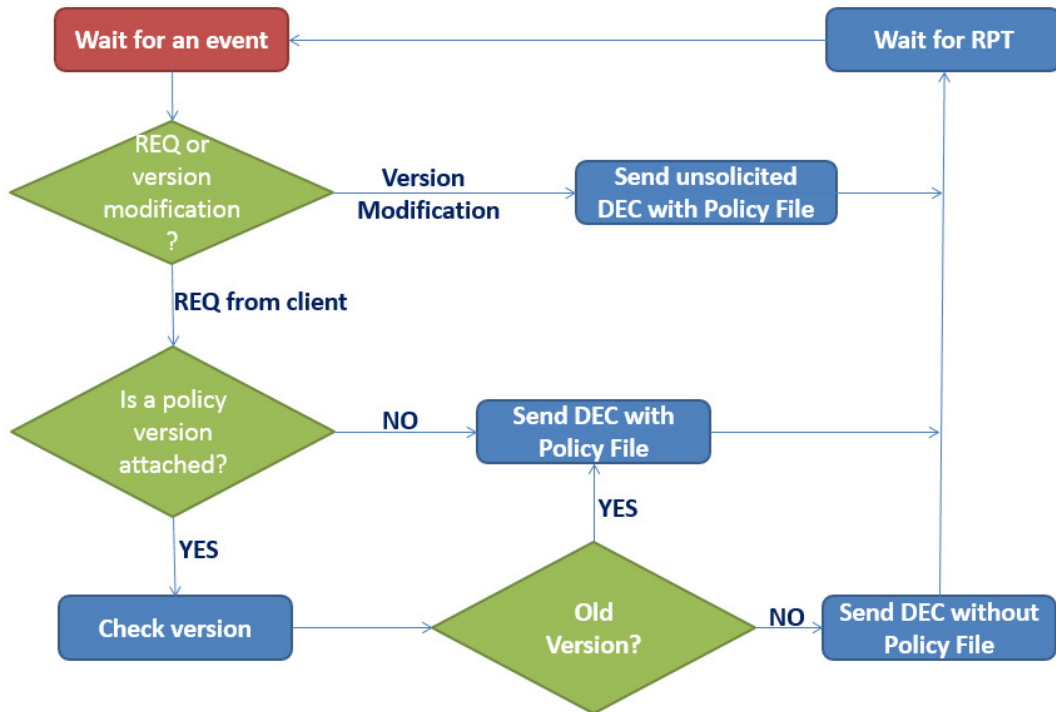


Figure 12 Server's Flow Chart

4.2.2 Client

Once the server is up and running, the client can be started by executing the PolicyClient program from the client folder, as illustrated by the example below:

```
./PolicyClient <Server-IP> <Server-Port> <Client-ID> <Secret Key> [Timer]
```

Example: `./PolicyClient 10.168.255.144 12345 router3 SeCrEtKeY 3600`

The PolicyClient program expects the following parameters:

<Server-IP>: The IP of the server with which the client is going to create the connection. This is the IP address of the server from the previous section.

<Server-Port>: The TCP port on the server where the connection should be created. This is also the port number specified in the previous section.

<Client-ID>: This is the client role set by the device operator; it should be a string of letters with an optional ending number. It is used by the server for admission control, and for distributing the matching policy to the client.

<Secret Key>: This is the secret key used in the MD5 hash function to secure the communication between the client and the server. It must be the same as the Secret Key of the server.

[Timer]: This is an optional parameter; it is an integer between 0 and 2147483647, measured in seconds. If set, the client will wait this amount of seconds to check with the server to see if there are any updates to the policy file. If it is not set, the client will only listen for the server updates. This mechanism is a second assurance that the client will always have the most recent copy of the device role policy.

After receiving a Client Accept (CAT) message from the policy decision control server, the policy client then sends a Request (REQ) message to the server. The server then sends a Decision (DEC) message with or without the policy file, depending on whether the REQ message contained a policy version or not. Otherwise, the policy client waits for an unsolicited Decision (DEC) message from the server. Or if a periodic timer is configured, it reports the existing policy

version number to the server. If an unsolicited Decision (DEC) message is received, the client saves the policy files and logs the previous version to the logs file, then confirms the receipt of the policy by sending a Report (RPT) message to the server. During the configured periodic check, the client checks to see whether it has a policy or not. If not, it sends a Request (REQ) message to the server and the server replies by sending a Decision (DEC) message with the policy file. If a policy file already exists on the policy client then the version number is reported to the decision control server. Figure 13 depicts the client's message flow.

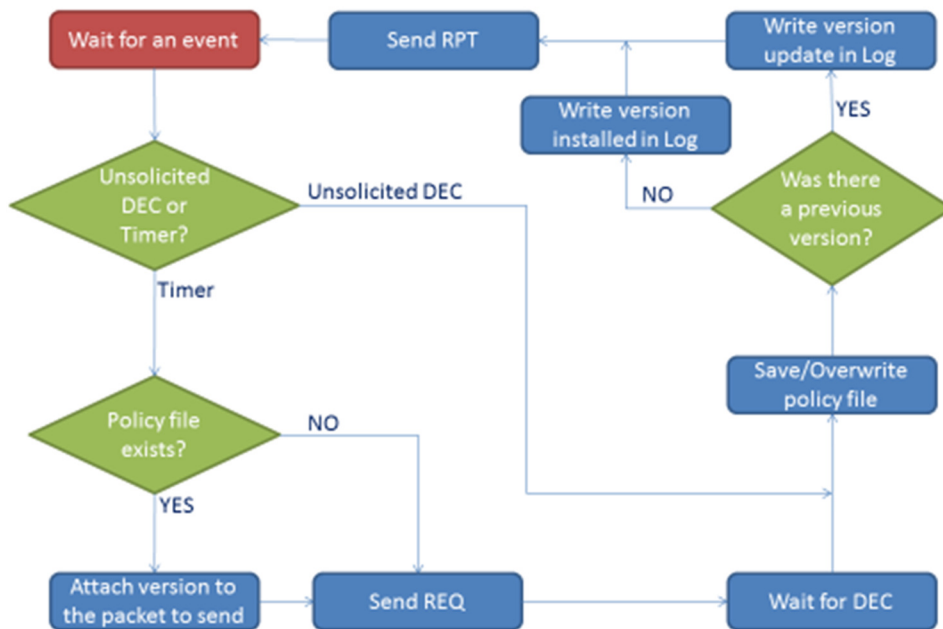


Figure 13 Client Flow Chart

4.2.3 Policy Exchange Protocol

In the real world, a protocol often refers to code of conduct or procedure, or to a system of rules to be followed in formal certain situations. In diplomatic exchange, for example, diplomats must follow certain rules of ceremony and form to ensure that they communicate effectively and without coming into conflict [67].

In networking, protocols define how communication is accomplished between two or more devices. They describe the format for transmitting data between the devices and ensuring that all the devices on a network or internetwork agree about how various actions must be performed in the total communication process.

In the context of our research, the client/server approach allowed us to prototype the client and the server components of the protocol in parallel. The independent nature of the model is especially evident when a need arises to change, modify or upgrade either side of the code. What follows is an explanation of the message format and message types and their associated codes:

The protocol is based on the following types of messages: OPEN, CAT, CC, REQ, DEC and RPT.

4.2.4 Message Types and Exchange

OPEN: When the client establishes its connection with the policy server, either for the first time or after a communication interruption, it sends an OPEN message to the policy server. The client role is encoded within this message, and

the policy decision control server performs admission control by either accepting or rejecting the connection. The connection is accepted if the client role is defined on the server, and rejected if no such role is found.

CAT: This message is sent from server to the client in response to the OPEN message. The Client Accept (CAT) is sent if the server finds a matching role for the client encoded in the OPEN message.

CC: The Client Close message is sent from the server to the client, and is part of the admission control performed by the policy decision control server. It is also sent in response to the OPEN message, but only if the server does not find a matching policy.

REQ: The Request message is sent from the policy client to the policy decision control server. It includes an encoded version number if a policy already exists on the client.

DEC: The Decision message is always sent from the server to the client. There are two types: a regular DEC in response to a REQ message from the client, and an unsolicited DEC when a policy change occurs on the server.

The REQ message received from the client may or may not include a policy version number. If it does, then the server performs a comparison to its own policy version number and pushes the newer policy if it has one. If not, it informs the client that the policy is the same. In addition, whenever a change occurs to the version number on the server, the server immediately initiates an

unsolicited DEC message to the client and pushes the modified policy, thus ensuring policy consistency.

RPT: the client uses the Report message to acknowledge that the policy file was received from the server.

Figure 14 below shows a successful admission after the client sends an OPEN message and receives a Client Accept (CAT) message from the server.

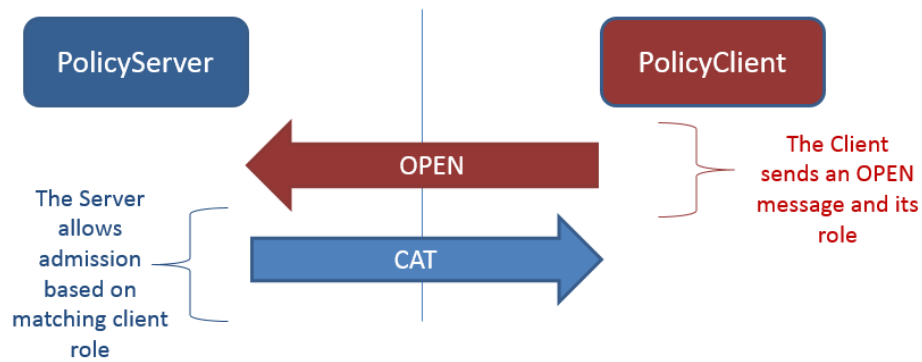


Figure 14 Client/Server Successful Admission

In Figure 15, we show a server denying a client admission by sending a CC message to the client in response to an unmatched policy role on the server.

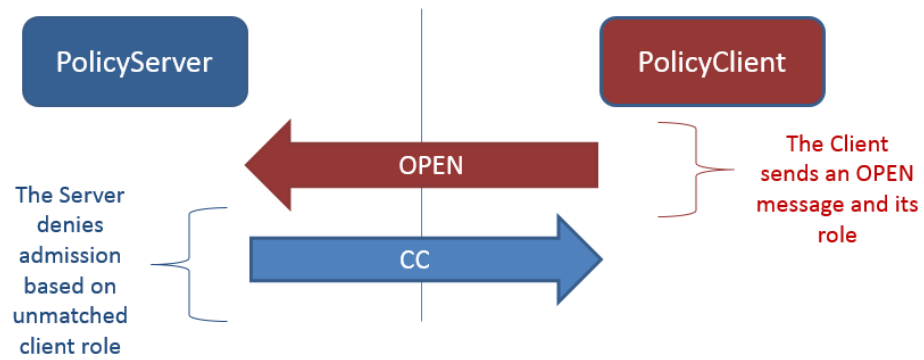


Figure 15 Client/Server Denied Admission

Figure 16 represents a DEC message normal communication cycle for a successfully admitted client.

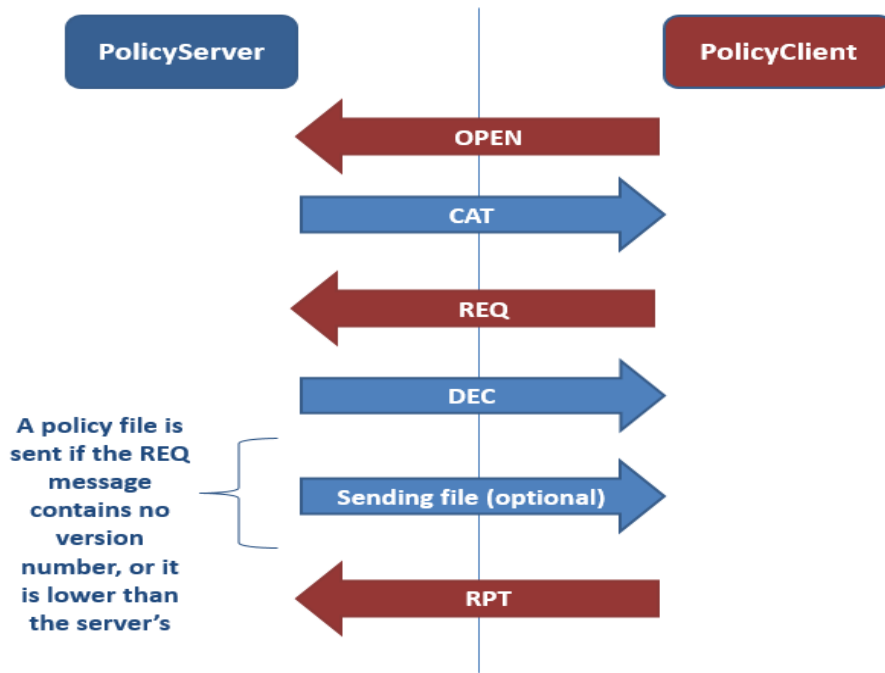


Figure 16 Client/Server Normal Cycle

Figure 17 illustrates the concept of Unsolicited DEC, where the policy is changed and the server synchronizes the policy with all affected clients.

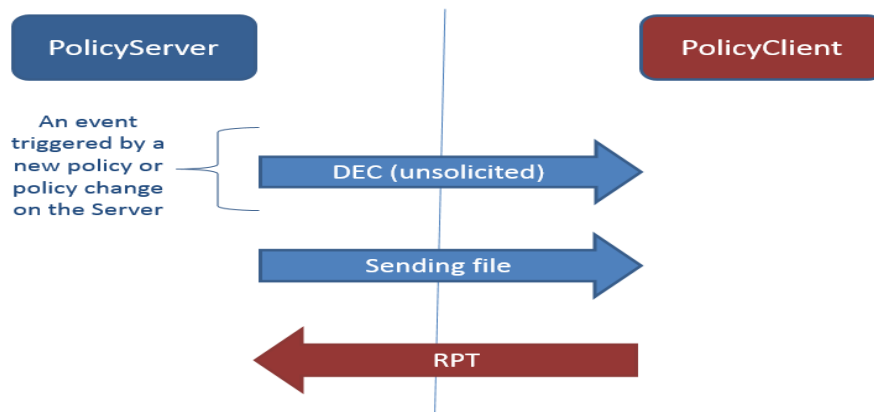


Figure 17 Unsolicited Message Triggered

4.2.5 Message Format

The messages described next are the fundamental unit of information responsible for transporting the message types described in the former section. They can be a fixed size or variable sizes, depending on the message and the data being transported. Regardless of size, each message consists of four main parts: version, flag, OP code, and client type.

A message length is added when an object is being transported. In addition, all messages have an MD5 hash code attached to them for added security. Figure 18 below illustrates the concept.

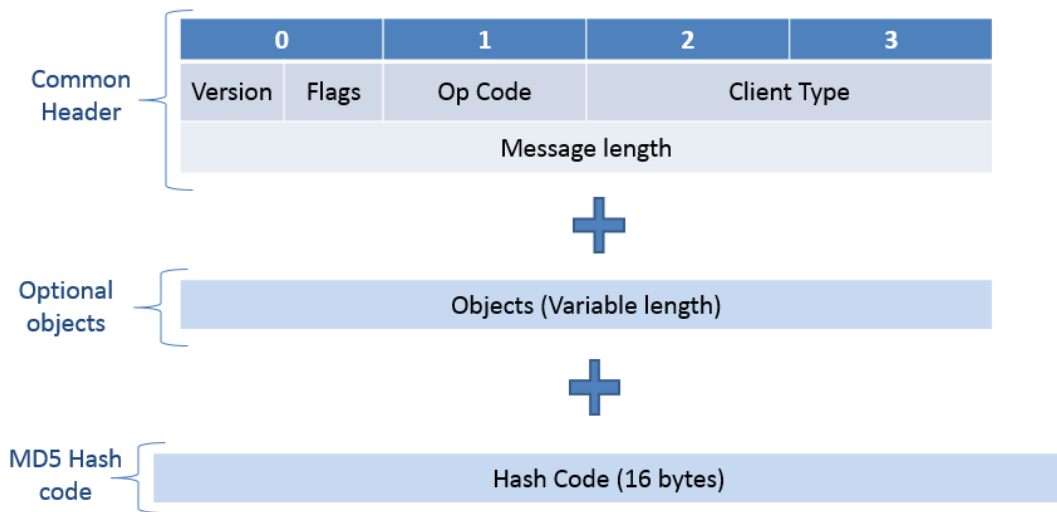


Figure 18 Message Format

The Version in the message header always has a value of 1 assigned to it, and the Flag field can be either 0x0 for a solicited message or 0x1 for an

unsolicited message, as explained earlier in the previous section. The OP Code represents the message type. The client type has two values assigned for now, 0x0000 for a server and 0x0001 for a UNIX client, However, with a 16-bit field we have the option to code up to 2^{16} , or 65,536, different types of networking devices. Table 4 lists the message header fields and their values.

Table 4 Message Header Fields and Values

Field	Value	
Version (4 bits)		
	0x1	Always 1
Flag (4 bits)		
	0x0	For solicited messages
	0x1	For unsolicited messages
OP Code (1 byte)		
	0x01	Request message (REQ)
	0x02	Decision message (DEC)
	0x03	Report message (RPT)
	0x06	OPEN message (OPEN)
	0x07	Accept Client message (CAT)
	0x08	Close Client message (CC)
Client Type (2 bytes)		
	0x0000	For Server
	0x0001	For Unix Client

The message length field is indicated when one or more objects are present. It represents the length of all the objects plus 4 bytes of the length of itself. For each object inside the common header, we have the following fields specified in Figure 19.

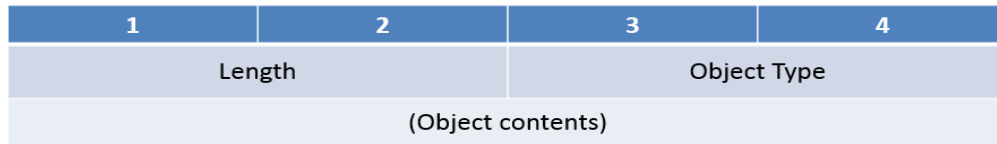


Figure 19 Object Header Format

The length field is a two-bytes field and represents the total object length, and the object field is a two-byte field representing the type of the object.

Currently, we have three different objects defined, but with a two-byte field we have an open window for further development and enhancement of this protocol.

Table 5 lists the different object types currently defined.

Table 5 Object Header Fields and Values

Field	Value	
Object Type (2 bytes)		
	0x0001	Policy Version
	0x0002	Policy File
	0x0003	RPT status

The hash key is added to all message exchanges to ensure the integrity of the transmitted messages. The client and the server share a common configurable secret key that allows them to calculate the hash. To ascertain integrity, the receiver calculates the hash of the received message and compares it to the received hash. If the hash is the same, the message is accepted. Otherwise, a mismatch indicates tampering.

The MD5 message-digest algorithm used in these messages is a widely-used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32 digit hexadecimal number. MD5 was designed by Ron Rivest in 1991 to replace an earlier hash function, and it has been utilized in a wide variety of cryptographic applications to verify data integrity [66]. Figure 20 depicts the hash code header attached to the message.



Figure 20 Hash Code Header

4.3 Example Scenarios

The purpose of this research is to present a model for automated policy compliance and change detection in data networks. One of the principal components of the proposed model is the policy exchange protocol between the policy decision control server and the policy client. One of the main goals of the policy exchange process is the distribution and maintainability of the policy. We need to guarantee not only secure policy delivery by the policy decision control server to the policy clients, but also that any further changes in the policy will be immediately detected and propagated to all affected policy clients.

In this section, we present plausible scenarios for the client/server exchange, from connection initiation to admission control by the policy decision

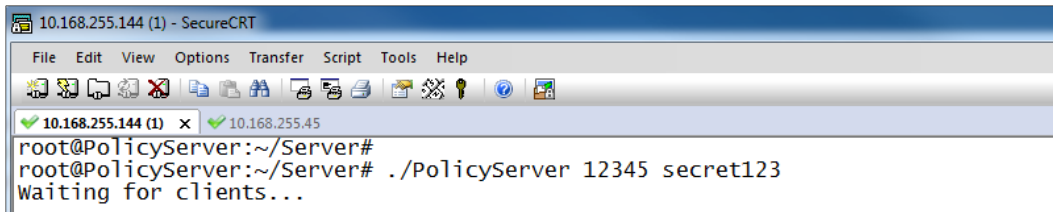
control server to policy exchange and reporting. This research specifically covers the following examples:

1. Unknown client role
2. Known client role, no policy stored
3. Known client role with existing policy with the same version number
4. Know client role with older existing policy
5. Know client role with newer existing policy
6. The server has a newer policy
7. Periodic checks

4.3.1 Unknown Client Role

For this test scenario, we will demonstrate the exchange of the OPEN and the Client Close (CC) message. Recall that the CC message is sent from the policy decision control server to the policy client in the event of admission rejection due to an unknown policy client role.

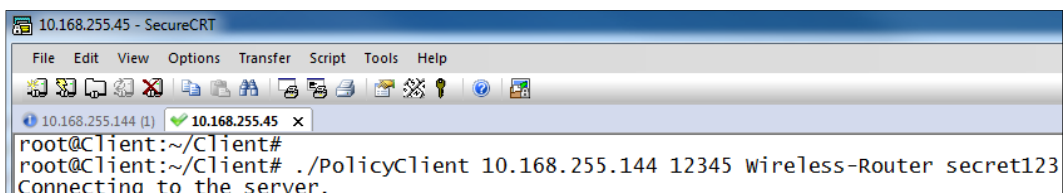
We start by running the PolicyServer, then by starting the client's PolicyClient software. The PolicyServer startup process is shown in Figure 21. The PolicyServer command is entered, followed by a port number (12345) and a secret key for the MD5 hash calculation (secret123).



```
10.168.255.144 (1) - SecureCRT
File Edit View Options Transfer Script Tools Help
10.168.255.144 (1) x 10.168.255.144
root@PolicyServer:~/Server#
root@PolicyServer:~/Server# ./PolicyServer 12345 secret123
Waiting for clients...
```

Figure 21 Starting the PolicyServer

The PolicyClient is started as captured in figure 22 by entering the PolicyClient command, followed by the IP address of the PolicyServer (10.168.255.144), then by the port number (12345), the device role type (Wireless-Router), and finally the MD5 secret key (secret123).



```
10.168.255.45 - SecureCRT
File Edit View Options Transfer Script Tools Help
10.168.255.144 (1) x 10.168.255.45
root@Client:~/Client#
root@Client:~/Client# ./PolicyClient 10.168.255.144 12345 wireless-Router secret123
Connecting to the server.
```

Figure 22 Starting the PolicyClient

To establish the TCP session between the policy client and the policy decision control server, first the server checks for a matching port and secret key parameters. In the event of a mismatch in either of the two parameters, the connection is terminated. However, if successful, the policy client starts the policy exchange process by sending an OPEN message to the policy decision control

server. The OPEN message contains the policy client role. In this test scenario the rule is set to “Wireless-Router.”

Figure 23, Portrays a scenario in which, after the policy client has made the connection, it sends an OPEN message to the server and waits for a Client Close (CC) or a Client Accept (CAT) message in return.

```
root@Client:~/Client# ./PolicyClient 10.168.255.144 12345 Wireless-Router
secret123
Connecting to the server.
hashBufferKey::buffer: <106010001801401576972656c6573732d526f757465720>
hashBufferKey::buffer_temp:
<106010001801401576972656c6573732d526f757465720736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: a8759d210a4a2e9861e8e3948f50f692
Sending OPEN request: <10 6 0 1 0 0 0 18 0 14 0 1 57 69 72 65 6c 65 73 73
2d 52 6f 75 74 65 72 0 61 38 37 35 39 64 32 31 30 61 34 61 32 65 39 38 36
31 65 38 65 33 39 34 38 66 35 30 66 36 39 32 >
Waiting for reply. Expecting CAT or CC.
```

Figure 23 Client Is Sending an OPEN Message

The corresponding states expected on the policy decision control server side are as follows: The server receives the connection request from the client and waits for an OPEN message. The policy decision control server examines the reported policy client's role in the OPEN message and checks for a matching policy.

In Figure 24, the server has received Client-ID 'Wireless-Router' and checks for a matching policy, which in this test scenario is not defined on the server. Therefore, the policy decision control server sends a Client Close (CC) message to the policy client.


```

New Client!
Waiting for query. Expecting OPEN.
read_policyPacket::Message-length: 18
hashBufferKey::buffer: <106010001801401576972656c6573732d526f757465720>
hashBufferKey::buffer_temp:
<106010001801401576972656c6573732d526f757465720736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: a8759d210a4a2e9861e8e3948f50f692
ParsePolicy::isAuthentic::hashGenerated:
<a8759d210a4a2e9861e8e3948f50f692>
ParsePolicy::isAuthentic::hashReceived :
<a8759d210a4a2e9861e8e3948f50f692>
Successfully authenticated.
OPEN expected: <10 6 0 1 0 0 0 18 0 14 0 1 57 69 72 65 6c 65 73 73 2d 52
6f 75 74 65 72 0 61 38 37 35 39 64 32 31 30 61 34 61 32 65 39 38 36 31 65
38 65 33 39 34 38 66 35 30 66 36 39 32 >
Client ID: Wireless-Router
Looking in file: modem.txt
Finding Device type in: modem.txt
Looking in file: router.txt
Finding Device type in: router.txt
Looking in file: office.txt
Finding Device type in: office.txt
Policy file for Wireless-Router not found.
hashBufferKey::buffer: <108010000>
hashBufferKey::buffer_temp: <108010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: fb4cb66f307e45e940d1b608190b6761
Sending CC: <10 8 0 1 0 0 0 0 66 62 34 63 62 36 36 66 33 30 37 65 34 35
65 39 34 30 64 31 62 36 30 38 31 39 30 62 36 37 36 31 >

```

Figure 24 Server Sends CC after Receiving Unknown Role

Figure 25 illustrates the policy client's receiving the Client Close (CC) message and immediately terminating its TCP session to the policy decision control server.

```

read_policyPacket::Message-length: 0
hashBufferKey::buffer: <108010000>
hashBufferKey::buffer_temp: <108010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: fb4cb66f307e45e940d1b608190b6761
ParsePolicy::isAuthentic::hashGenerated:
<fb4cb66f307e45e940d1b608190b6761>
ParsePolicy::isAuthentic::hashReceived :
<fb4cb66f307e45e940d1b608190b6761>
Successfully authenticated.
Packet received: <10 8 0 1 0 0 0 0 66 62 34 63 62 36 36 66 33 30 37 65 34
35 65 39 34 30 64 31 62 36 30 38 31 39 30 62 36 37 36 31 >
hashBufferKey::buffer: <108010000>
hashBufferKey::buffer_temp: <108010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: fb4cb66f307e45e940d1b608190b6761
ParsePolicy::isAuthentic::hashGenerated:
<fb4cb66f307e45e940d1b608190b6761>
ParsePolicy::isAuthentic::hashReceived :
<fb4cb66f307e45e940d1b608190b6761>
SUCCESSFULLY AUTHENTICATED!!!!
CC received. The client is shutting down...

```

Figure 25 The Policy Client Receives CC Message

Now let us examine and analyze the data in the message and see how they line up with the message header format discussed in section 4.2.5 Message Format.

From Figure 23, we see that the OPEN message contains the following:

```

Sending OPEN request: <10 6 0 1 0 0 0 18 0 14 0 1 57
69 72 65 6c 65 73 73 2d 52 6f 75 74 65 72 0 61 38 37
35 39 64 32 31 30 61 34 61 32 65 39 38 36 31 65 38 65
33 39 34 38 66 35 30 66 36 39 32 >

```

Figure 26, shows the bits layout based on the message format discussed earlier.

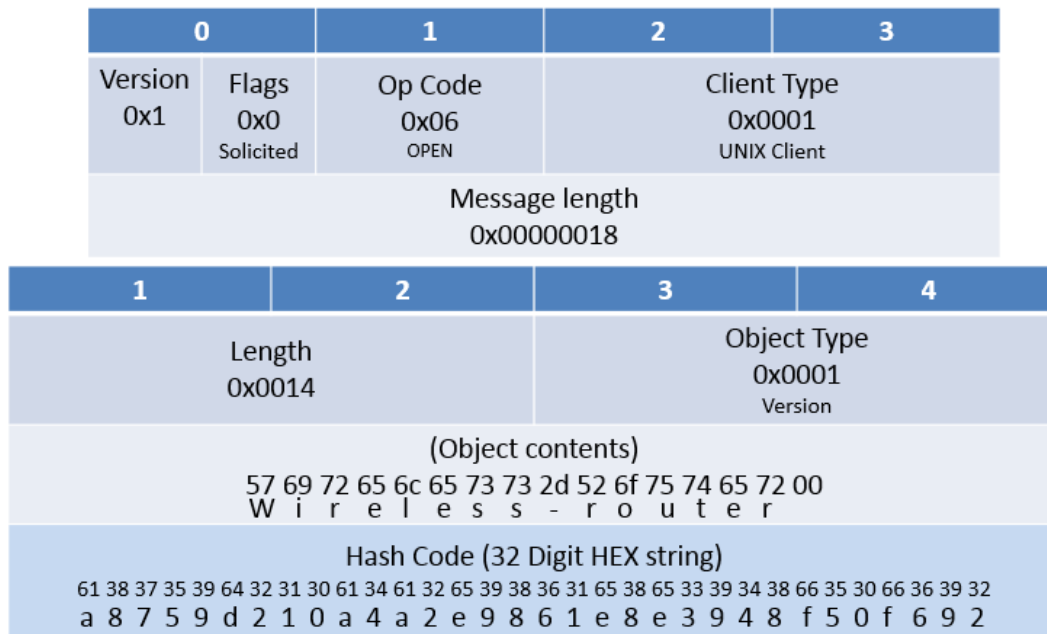


Figure 26 OPEN Message Layout

We also captured in Figure 27 the same message format using the message capture utility Wireshark.

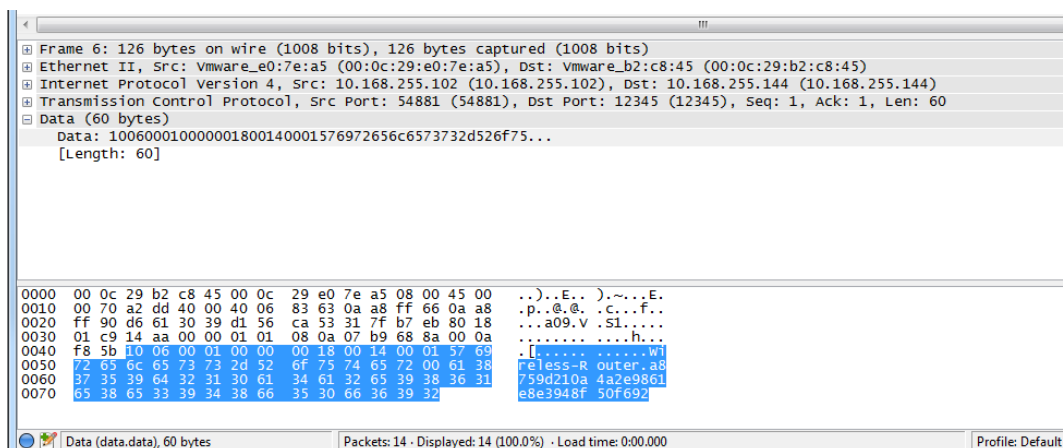


Figure 27 OPEN Message from Wireshark

Next, we explore the captured message from the server side. Figure

24, has the following data:

```

Sending CC: <10 8 0 1 0 0 0 0 66 62 34 63 62 36 36 66
33 30 37 65 34 35 65 39 34 30 64 31 62 36 30 38 31 39
30 62 36 37 36 31 >
    
```

Figure 28 lays out the bits for the Client Close (CC) message captured from the server side. It is also worth noting that the calculated hash key is identical on both client and server.

0		1		2		3	
Version 0x1	Flags 0x0 Solicited	Op Code 0x08 CC		Client Type 0x0001 UNIX Client			
Message length 0x00000000							
Hash Code (32 Digit HEX string) 66 62 34 63 62 36 36 66 33 30 37 65 34 35 65 39 34 30 64 31 62 36 30 38 31 39 30 62 36 37 36 31 f b 4 c b 6 6 f 3 0 7 e 4 5 e 9 4 0 d 1 b 6 0 8 1 9 0 b 6 7 6 1							

Figure 28 CC Message Layout

4.3.2 Known Client Role, No Policy Stored

This example scenario is different from the last in that the policy client role is known to the policy decision control server. Therefore, this time we expect the policy decision control server’s admission control process to send a Client

Accept (CAT) message to the policy client, then initiate a policy file download after sending a DEC message. Finally, the policy client should confirm the receipt of the policy file by sending an RPT message back to the policy decision control server.

```

root@Client:~/Client# ./PolicyClient 10.168.255.144 12345 office-client1
secret123
Connecting to the server.
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f666666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
Sending OPEN request: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63
6c 69 65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31
38 37 33 63 36 30 36 32 32 37 35 39 63 33 >
Waiting for reply. Expecting CAT or CC.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
Successfully authenticated.
Packet received: <10 7 0 1 0 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38
37 31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
SUCCESSFULLY AUTHENTICATED!!!!
CAT received.

```

Figure 29 OPEN, CAT, DEC and RPT Message Exchange

From Figure 29, we see that the policy client sent an OPEN message to the policy decision control server using the client role 'office-client1' and waited for a

reply from the server, expecting a Client Close (CC) message or a Client Accept (CAT) message. After receiving the OPEN message, the policy decision control server checks to see if it has a matching policy for the client role “office-client1.” In this test case, it finds a policy called “office.txt”, and sends the policy client a Client Accept (CAT) message.

Figure 30, also shows that the policy client has received a CAT from the policy decision control server.

The message exchange from the policy decision control server side is depicted in Figure 30.

```
New Client!
Waiting for query. Expecting OPEN.
read_policyPacket::Message-length: 17
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f666666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
ParsePolicy::isAuthentic::hashGenerated:
<e39b1526ed57cd5051873c60622759c3>
ParsePolicy::isAuthentic::hashReceived :
<e39b1526ed57cd5051873c60622759c3>
Successfully authenticated.
OPEN expected: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63 6c 69
65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31 38 37
33 63 36 30 36 32 32 37 35 39 63 33 >
Client ID: office-client1
Looking in file: modem.txt
Finding Device type in: modem.txt
Looking in file: router.txt
Finding Device type in: router.txt
Looking in file: office.txt
Finding Device type in: office.txt
Policy file for this client: office.txt
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
Sending CAT: <10 7 0 1 0 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38 37
31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
```

```
Waiting for an event...
```

Figure 30 policy decision control server Message Exchange

As detailed in chapter 3 and 4 of this dissertation, once the policy client receives the Client Accept (CAT) message from the policy decision control server, it is ready to send the Request (REQ) message. In this scenario, our policy client does not hold a previous policy, so it sends the Request (REQ) message without attaching a policy version number and expects a Decision (DEC) message from the server, which will initiate the policy file download process.

Figure 31, shows the policy client sending a Request (REQ) message to the policy decision control server and receiving a Decision (DEC) message policy decision control server in reply, followed by a secure policy file download from the server using MD5 for file integrity.

```
--- Timer ---
hashBufferKey::buffer: <101010000>
hashBufferKey::buffer_temp: <101010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 3905545f31ac0da0add88f3967db92d7
Sending REQ: <10 1 0 1 0 0 0 0 33 39 30 35 35 34 35 66 33 31 61 63 30 64
61 30 61 64 64 38 38 66 33 39 36 37 64 62 39 32 64 37 >
Waiting for reply. Expecting DEC.
read_policyPacket::Message-length: 9
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
Successfully authenticated.
DEC received: <10 2 0 0 0 0 0 9 0 5 0 2 0 61 66 36 39 66 64 38 63 32 64
61 63 66 35 65 33 33 63 36 37 61 65 30 64 37 32 35 38 63 62 62 30 >
hashBufferKey::buffer: <10200000905020>
```

```

hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
SUCCESSFULLY AUTHENTICATED!!!!
ReceiveFile::Starting to download file...
ReceiveFile::Size of file: 1032
ReceiveFile::rcv_Hashcode: 0e5355d9ba55a26169c859b849c15282
ReceiveFile::FILE SUCCESSFULLY AUTHENTICATED
ReceiveFile::Download completed
Writing log

```

Figure 31 Policy Client REQ Message Sent

Now let us examine the message exchange from the server side. We notice in Figure 32 that the policy decision control server has received a Request (REQ) message from the policy client and that the message does not contain a policy version number. Therefore the policy decision control server sends the matching policy file, office.txt, to the policy client.

```

--- Client Request ---
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <101010000>
hashBufferKey::buffer_temp: <101010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 3905545f31ac0da0add88f3967db92d7
ParsePolicy::isAuthentic::hashGenerated:
<3905545f31ac0da0add88f3967db92d7>
ParsePolicy::isAuthentic::hashReceived :
<3905545f31ac0da0add88f3967db92d7>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 0 33 39 30 35 35 34 35 66 33 31 61 63 30 64
61 30 61 64 64 38 38 66 33 39 36 37 64 62 39 32 64 37 >
No policy received. Sending policy file.
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
PolicyServer::DownloadPolicy::Sending file...
Checking file: <office.txt>
SendFile::Size of file: 1032 bytes
SendFile::snd_Hashcode: 0e5355d9ba55a26169c859b849c15282 bytes

```



```
Waiting for query. Expecting RPT.
```

Figure 32 policy decision control server Sending Policy File

Once again, we observe in the message exchange that the calculated hash code used to ensure file integrity is the same on both the sender and receiver sides, as depicted in Figure 33.

```
policy decision control server side  
Checking file: <office.txt>  
SendFile::Size of file: 1032 bytes  
SendFile::snd_Hashcode: 0e5355d9ba55a26169c859b849c15282 bytes  
  
Policy Client side  
ReceiveFile::Starting to download file...  
ReceiveFile::Size of file: 1032  
ReceiveFile::rcv_Hashcode: 0e5355d9ba55a26169c859b849c15282  
ReceiveFile::FILE SUCCESSFULLY AUTHENTICATED  
ReceiveFile::Download completed
```

Figure 33 Hash Code Matching

Finally, we witness the policy client sending a Report (RPT) message to the policy decision control server, confirming the receipt of the policy file. Figure 34 depicts the RPT message sent to the server.

```
hashBufferKey::buffer: <103010000>  
hashBufferKey::buffer_temp: <103010000736563726574313233>  
hashBufferKey::password: secret123  
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121  
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30  
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >  
Sleeping.
```

Figure 34 RPT Message Sent

We also confirm on the policy decision control server the receipt of the RPT message from the policy client as show in Figure 35.

```
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...
```

Figure 35 RPT Message Received

4.3.3 Known Client Role with Existing Policy with the Same Version Number

In this example case scenario, we will examine the message exchange between the client whose role already has been defined on the policy decision control server, and which already has an existing policy with the same version number as the policy decision control server's. This scenario could occur if the policy client had a previous session with the policy decision control server and, for reasons discussed in Chapter 3, the TCP session was severed. It could also occur if the policy was copied manually to the policy client.

After establishing the TCP connection with the policy decision control server, the policy client sends an OPEN message containing its device role and waits for a Client Close (CC) or Client Accept (CAT) message from the policy

decision control server. For this example, the device role is 'office-client1.' Figure 36 shows the captured OPEN message exchange.

```
root@Client:~/Client# ./PolicyClient 10.168.255.144 12345 office-client1
secret123
Connecting to the server.
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f666666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
Sending OPEN request: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63
6c 69 65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31
38 37 33 63 36 30 36 32 32 37 35 39 63 33 >
Waiting for reply. Expecting CAT or CC.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
Successfully authenticated.
Packet received: <10 7 0 1 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38
37 31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
SUCCESSFULLY AUTHENTICATED!!!!
CAT received.
```

Figure 36 Client OPEN Sent

Upon receiving the OPEN message from the policy client, the policy decision control server performs the admission control function by comparing the reported client role in the OPEN message to the known policy decision control server client roles.

In Figure 37, the policy decision control server finds a match, sends a Client Accept (CAT) message and waits for a Request (REQ) message from the policy client.

```
New Client!  
Waiting for query. Expecting OPEN.  
read_policyPacket::Message-length: 17  
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>  
hashBufferKey::buffer_temp:  
<1060100017013016f666666963652d636c69656e74310736563726574313233>  
hashBufferKey::password: secret123  
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3  
ParsePolicy::isAuthentic::hashGenerated:  
<e39b1526ed57cd5051873c60622759c3>  
ParsePolicy::isAuthentic::hashReceived :  
<e39b1526ed57cd5051873c60622759c3>  
Successfully authenticated.  
OPEN expected: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63 6c 69  
65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31 38 37  
33 63 36 30 36 32 32 37 35 39 63 33 >  
Client ID: office-client1  
Looking in file: modem.txt  
Finding Device type in: modem.txt  
Looking in file: router.txt  
Finding Device type in: router.txt  
Looking in file: office.txt  
Finding Device type in: office.txt  
Policy file for this client: office.txt  
hashBufferKey::buffer: <107010000>  
hashBufferKey::buffer_temp: <107010000736563726574313233>  
hashBufferKey::password: secret123  
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c  
Sending CAT: <10 7 0 1 0 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38 37  
31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >  
Waiting for an event...
```

Figure 37 CAT Message Sent

After receiving the Client Accept (CAT) message, the policy client is now ready to send its Request (REQ) message. Since it already has an existing client policy, it encodes the policy file version number inside the REQ message and awaits a Decision (DEC) message from the policy decision control server.

Figure 38, shows the REQ message with the reported version number, and Figure 39 shows the same information using the message capture utility Wireshark, including the MD5 hash “f43d317bc91f81eb552780c35be11227”

```
Policy file exists. Version: %version 3.394m
hashBufferKey::buffer: <10101000180014012576657273696f6e20332e3339346d0>
hashBufferKey::buffer_temp:
<10101000180014012576657273696f6e20332e3339346d0736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: f43d317bc91f81eb552780c35be11227
Sending REQ: <10 1 0 1 0 0 0 18 0 14 0 1 25 76 65 72 73 69 6f 6e 20 33 2e
33 39 34 6d 0 66 34 33 64 33 31 37 62 63 39 31 66 38 31 65 62 35 35 32 37
38 30 63 33 35 62 65 31 31 32 32 37 >
Waiting for reply. Expecting DEC
```

Figure 38 REQ with Version Number

The image shows a Wireshark capture of a network packet. The packet list pane shows a single packet of 126 bytes. The packet details pane shows the following structure:

- Ethernet II, Src: Vmware_e0:7e:a5 (00:0c:29:e0:7e:a5), Dst: Vmware_b2:c8:45 (00:0c:29:b2:c8:45)
- Internet Protocol Version 4, Src: 10.168.255.102 (10.168.255.102), Dst: 10.168.255.144 (10.168.255.144)
- Transmission Control Protocol, Src Port: 54911 (54911), Dst Port: 12345 (12345), Seq: 60, Ack: 41, Len: 60
- Data (60 bytes)

The data field is expanded to show the following hex and ASCII representation:

```
0000 00 0c 29 b2 c8 45 00 0c 29 e0 7e a5 08 00 45 00 ..)..E.. )~...E.
0010 00 70 1e 34 40 00 40 06 08 0d 0a a8 ff 66 0a a8 .p.4@.@. ....f..
0020 ff 90 d6 7f 30 39 f0 24 0e 2a 52 21 28 3a 80 18 ....09.$.*R!(...
0030 01 c9 14 aa 00 00 01 01 08 0a 10 a6 13 de 08 f7 .....
0040 a3 b0 10 01 00 01 00 00 00 18 00 14 00 01 25 76 .....%v
0050 65 72 73 69 6f 6e 20 33 2e 33 39 34 6d 00 66 34 ersion 3 .394m.f4
0060 33 64 33 31 37 62 63 39 31 66 38 31 65 62 35 35 3d317bc9 1f81eb55
0070 32 37 38 30 63 33 35 62 65 31 31 32 32 37 2780c35b e11227
```

Figure 39 Wireshark Capture REQ with Version Number

The policy decision control server receives the Request (REQ) message with the client's policy version number and compares it to its stored policy for that device role. In our test case, the policy version number is the same, therefore the server sends a Decision (DEC) message to the policy client and expects to see a confirmation from the policy client by receiving a Report (RPT) message. This exchange is illustrated in Figure 40.

```

read_policyPacket::Message-length: 18
hashBufferKey::buffer: <1010100018014012576657273696f6e20332e3339346d0>
hashBufferKey::buffer_temp:
<1010100018014012576657273696f6e20332e3339346d0736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: f43d317bc91f81eb552780c35be11227
ParsePolicy::isAuthentic::hashGenerated:
<f43d317bc91f81eb552780c35be11227>
ParsePolicy::isAuthentic::hashReceived :
<f43d317bc91f81eb552780c35be11227>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 18 0 14 0 1 25 76 65 72 73 69 6f 6e 20 33
2e 33 39 34 6d 0 66 34 33 64 33 31 37 62 63 39 31 66 38 31 65 62 35 35 32
37 38 30 63 33 35 62 65 31 31 32 32 37 >
The client has an updated version.
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
Sending DEC: <10 2 0 1 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
Waiting for query. Expecting RPT.

```

Figure 40 DEC Message Sent

The fact that the server sent the Decision (DEC) message without initiating a download process informs the policy client that it has the most recent policy file. Therefore, the policy client completes the exchange by sending a

Report (RPT) message to the policy decision control server confirming the receipt of the DEC message. Figure 41 details the exchange.

```
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>
Successfully authenticated.
DEC received: <10 2 0 1 0 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>
SUCCESSFULLY AUTHENTICATED!!!!
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.
```

Figure 41 Client Receives a DEC and Sends RPT message

Finally, Figure 42 shows the RPT message received by the policy decision control server.

```
Waiting for query. Expecting RPT.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
```

```
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...
```

Figure 42 RPT Received by the Server

4.3.4 Known Client Role with Older Existing Policy

Now let us explore what happens if the policy client has a policy older than what is stored on the policy decision control server. This condition could occur if the policy client was offline during the last policy file update, or due to a failed update attempt by the server. The newer policy detection may occur on reconnect, or during the configured client periodic check as described in Chapter 3. In either case, the message exchange process between the policy client and the policy decision control server is the same.

For our example, the client office-client1 has a policy file with version 3.4, and the policy decision control server has version 3.5 for the office-client device role. As described earlier, the policy client starts by sending an OPEN message with its device role office-client1 encoded within the message, and waits for the server to initiate its admission control process.

In Figure 43, we observe that the client has sent an OPEN message and in return has received a Client Accept (CAT) message from the policy decision control server.


```

Connecting to the server.
hashBufferKey::buffer: <1060100017013016f66666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f66666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
Sending OPEN request: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63
6c 69 65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31
38 37 33 63 36 30 36 32 32 37 35 39 63 33 >
Waiting for reply. Expecting CAT or CC.

read_policyPacket::Message-length: 0
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
Successfully authenticated.
Packet received: <10 7 0 1 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38
37 31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
SUCCESSFULLY AUTHENTICATED!!!!
CAT received.

```

Figure 43 OPEN Sent, CAT Received

The next step is for the policy client to send a Request (REQ) message encoded with the existing policy file version number. Again, the version number is read from the policy file's mandatory variable '%version' in the file currently located on the policy client.

In Figure 44, we see that the policy client has detected version 3.4, has sent a Request (REQ) message, and is awaiting the policy decision control server's Decision (DEC) message.

```

Policy file exists. Version: %version 3.4
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e340>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e340736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: f1d0274816956d0dae1863f366428513
Sending REQ: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33 2e
34 0 66 31 64 30 32 37 34 38 31 36 39 35 36 64 30 64 61 65 31 38 36 33 66
33 36 36 34 32 38 35 31 33 >
Waiting for reply. Expecting DEC.

```

Figure 44 Version Number Sent Within DEC

The server in Figure 45 receives the Request (REQ) message and checks the policy file associated with the office-client1 role, which happens to be office.txt, and compares the file version number to that reported by the client in the Request (REQ) message. Because the server's version number is 3.5, which higher than the policy client's 3.4, the server sends the Decision (DEC) message and pushes the policy file to the policy client, then awaits the Report (RPT) message for confirmation.

```

read_policyPacket::Message-length: 15
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e340>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e340736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: f1d0274816956d0dae1863f366428513
ParsePolicy::isAuthentic::hashGenerated:
<f1d0274816956d0dae1863f366428513>
ParsePolicy::isAuthentic::hashReceived :
<f1d0274816956d0dae1863f366428513>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33
2e 34 0 66 31 64 30 32 37 34 38 31 36 39 35 36 64 30 64 61 65 31 38 36 33
66 33 36 36 34 32 38 35 31 33 >
The client has an older version. Sending updated one
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
PolicyServer::DownloadPolicy::Sending file...
Checking file: <office.txt>

```

```
SendFile::Size of file: 1029 bytes
SendFile::snd_Hashcode: 9b5406a463d031d322e79b241754799f bytes
Waiting for query. Expecting RPT.
```

Figure 45 DEC Sent, File Pushed

The policy client in Figure 46 as depicted has received the Decision (DEC) message from the policy decision control server and started the file download process.

```
read_policyPacket::Message-length: 9
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
Successfully authenticated.
DEC received: <10 2 0 0 0 0 0 9 0 5 0 2 0 61 66 36 39 66 64 38 63 32 64
61 63 66 35 65 33 33 63 36 37 61 65 30 64 37 32 35 38 63 62 62 30 >
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
SUCCESSFULLY AUTHENTICATED!!!!
ReceiveFile::Starting to download file...
ReceiveFile::Size of file: 1029
ReceiveFile::rcv_Hashcode: 9b5406a463d031d322e79b241754799f
ReceiveFile::FILE SUCCESSFULLY AUTHENTICATED
ReceiveFile::Download completed
Writing log
```

Figure 46 DEC Received, File Download Started

Once the file is downloaded successfully, the policy client sends a Report (RPT) message informing the policy decision control server of the successful completion. Otherwise, the server reinitiates the file upload. *Sleeping.*

Figure 47 shows the policy client sending an RPT message confirming the successful file download.

```
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.
```

Figure 47 File Download Completed, Sending RPT

Finally, we show in Figure 48 the server's receipt of a Report (RPT) message from the policy client, confirming the successful file download operation.

```
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...
```

Figure 48 RPT Received Confirming File Download

4.3.5 Known Client Role with Newer Existing Policy

In this section, we explore an example case scenario where the policy client has a policy file newer than that stored on the policy decision control server. One cause for this could be that the network administrator made changes to the policy file locally instead of in a centralized location, such as the policy decision control server or the repository.

Earlier in the Design Consideration section of this dissertation, we discussed such a scenario and recommended a course of action for keeping the newer policy file on the policy client, instead of overwriting it by the server's version. The assumption here is that the network administrator is aware of the existence of the Automated Change Detection System in the network, but for some reason has a compelling motivation to make changes locally. Hence, we expect the server to honor the policy client's version of the policy by allowing it to use the locally-stored version instead of the master version. This state will remain as is until the policy decision control server's version of the policy is higher than the local policy.

From Figure 49, we see that the policy client has successfully received a Client Accept (CAT) message.

```
Connecting to the server.  
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>  
hashBufferKey::buffer_temp:  
<1060100017013016f666666963652d636c69656e74310736563726574313233>  
hashBufferKey::password: secret123  
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
```

```

Sending OPEN request: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63
6c 69 65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31
38 37 33 63 36 30 36 32 32 37 35 39 63 33 >
Waiting for reply. Expecting CAT or CC.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
Successfully authenticated.
Packet received: <10 7 0 1 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38
37 31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
SUCCESSFULLY AUTHENTICATED!!!!
CAT received.

```

Figure 49 OPEN Sent, CAT Received

After receiving the Client Accept (CAT) message from the policy decision control server, the client sends a Request (REQ) message containing the version number of the policy it currently has. This time version number is 3.6. Then it waits for the Decision (DEC) message from the server, as seen in Figure 50.

```

Policy file exists. Version: %version 3.6
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e360>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e360736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5bac5090eb291fb71cf13960f322b005
Sending REQ: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33 2e
36 0 35 62 61 63 35 30 39 30 65 62 32 39 31 66 62 37 31 63 66 31 33 39 36
30 66 33 32 32 62 30 30 35 >
Waiting for reply. Expecting DEC.

```

Figure 50 Awaiting DEC from Server

The policy decision control server receives the Request (REQ) message and compares the reported policy version number with the stored version number. It determines that the client has a newer version and sends a DEC message. Then, since no file download action is required, the server simply waits for a Report (RPT) message from the client, as seen in Figure 51.

```
read_policyPacket::Message-length: 15
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e360>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e360736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5bac5090eb291fb71cf13960f322b005
ParsePolicy::isAuthentic::hashGenerated:
<5bac5090eb291fb71cf13960f322b005>
ParsePolicy::isAuthentic::hashReceived :
<5bac5090eb291fb71cf13960f322b005>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33
2e 36 0 35 62 61 63 35 30 39 30 65 62 32 39 31 66 62 37 31 63 66 31 33 39
36 30 66 33 32 32 62 30 30 35 >
WARNING: The client has a newer version.
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
Sending DEC: <10 2 0 1 0 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
Waiting for query. Expecting RPT.
```

Figure 51 Client Has a Newer Version

The policy client receives the Decision (DEC) message from the policy decision control server and sends a Report (RPT) message in reply, as seen in Figure 52.

```

read_policyPacket::Message-length: 0
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>
Successfully authenticated.
DEC received: <10 2 0 1 0 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>
SUCCESSFULLY AUTHENTICATED!!!!
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.

```

Figure 52 DEC Received, RPT Sent

Lastly, in Figure 53, the policy decision control server receives the Report (RPT) message from the policy client, confirming the completion of the transaction. The result of the entire transaction is that the client keeps its existing policy file.

```

read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>

```



```
ParsePolicy::isAuthentic::hashReceived :  
<57125e33b634d0725baa326907a0a121>  
Successfully authenticated.  
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64  
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >  
Waiting for an event..
```

Figure 53 RPT Received, Transaction Completed

4.3.6 The Server Has a Newer Policy

In Chapter 3, we noted that it might be common for the network administrator to modify, edit, or delete some elements of an existing policy, or even create a new one, via the policy management interface. In such a scenario, we expect the policy decision control server to propagate the new policy to all affected policy clients by initiating an unsolicited Decision (DEC) message. This process is critical because it allows the network administrator to update one or many policies and, with one stroke, propagate the new policies to many network elements.

In this test scenario, we will show the functionality of server-side propagation by modifying the existing office-client device policy on the server and saving the policy file with a higher version, version 3.7.

Figure 54 illustrates the use of the unsolicited DEC message by the policy decision control server. The message is immediately generated after the policy file has been saved, then the server pushes a 1,029-byte file to the policy client and awaits a Report (RPT) message in the reply.

```

--- Send unsolicited DEC ---
hashBufferKey::buffer: <11200000905020>
hashBufferKey::buffer_temp: <11200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 8383a3135b62146e768239f9cef77432
PolicyServer::DownloadPolicy::Sending file...
Checking file: <office.txt>
SendFile::Size of file: 1029 bytes
SendFile::snd_Hashcode: ebf50ac4874dfa216dd8254fd98f8e82
Waiting for query. Expecting RPT.

```

Figure 54 Unsolicited DEC Sent

The policy client in Figure 55 receives the unsolicited DEC message, accepts the file download, and compares the sent hash code with its calculated hash for the file.

```

--- Unsolicited DEC ---
read_policyPacket::Message-length: 9
hashBufferKey::buffer: <11200000905020>
hashBufferKey::buffer_temp: <11200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 8383a3135b62146e768239f9cef77432
ParsePolicy::isAuthentic::hashGenerated:
<8383a3135b62146e768239f9cef77432>
ParsePolicy::isAuthentic::hashReceived :
<8383a3135b62146e768239f9cef77432>
Successfully authenticated.
DEC received: <11 2 0 0 0 0 0 9 0 5 0 2 0 38 33 38 33 61 33 31 33 35 62
36 32 31 34 36 65 37 36 38 32 33 39 66 39 63 65 66 37 37 34 33 32 >
hashBufferKey::buffer: <11200000905020>
hashBufferKey::buffer_temp: <11200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 8383a3135b62146e768239f9cef77432
ParsePolicy::isAuthentic::hashGenerated:
<8383a3135b62146e768239f9cef77432>
ParsePolicy::isAuthentic::hashReceived :
<8383a3135b62146e768239f9cef77432>
SUCCESSFULLY AUTHENTICATED!!!!
ReceiveFile::Starting to download file...
ReceiveFile::Size of file: 1029
ReceiveFile::rcv_Hashcode: ebf50ac4874dfa216dd8254fd98f8e82
ReceiveFile::FILE SUCCESSFULLY AUTHENTICATED
ReceiveFile::Download completed

```

Figure 55 Unsolicited DEC Received

Once the file download is completed, the policy client sends a Report (RPT) message to the policy decision control server confirming the status. See Figure 56.

```
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.
```

Figure 56 RPT Sent to Policy Server

Figure 57 shows the server receiving the RPT message.

```
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...
```

Figure 57 RPT Received From Client

The unsolicited DEC message functionality can also be used by the network administrator to quickly revert any misbehaving policy. Imagine a scenario in which a policy change is made, and soon after, an undesirable effect is detected on the network. The administrator has the options to either roll back the

unwanted changes on the master policy file or simply load a new policy file.

Either way, the unstable state will quickly be reverted.

4.3.7 Periodic Checks

In this test case, we address what happens when the policy file is deleted intentionally or unintentionally from the policy client.

In the earlier sections, we demonstrated different cases in which the policy client receives new policy file from the policy decision control server, based either on a REQ message sent by the client or on an unsolicited DEC message generated by the policy decision control server in reaction to a policy file update event. However, without periodic checks, the policy client is susceptible to being without a valid policy file. This could happen due to intentional or unintentional deletion of the policy file, or even due to faulty software and/or hardware. This is because the policy client awaits an event to trigger a policy check or a policy download, as explained above. If there is no triggering event, it goes into 'sleep' mode after sending its RPT message. Meanwhile, the policy decision control server also awaits an event after receiving the RPT message. Both states are shown in Figure 58.

Policy Client side
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 > Sleeping.
policy decision control server side

```
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64  
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >  
Waiting for an event...
```

Figure 58 Waiting for an Event

To overcome this limitation, our system design allows the system administrator to configure an optional periodic check interval parameter on the policy client, measured in seconds, as detailed in section 4.2.2.

When this is enabled, the policy client sends the policy decision control server a REQ message every configured interval, and the receiving server examines the REQ message for an encoded version number. If none is found, or if the encoded version number is lower than the server's version number, then the server replies with a DEC message, followed by a file download. Otherwise, only a DEC message is sent from the server, and an RPT message is always sent from the client to confirm. Figure 59 details this concept.

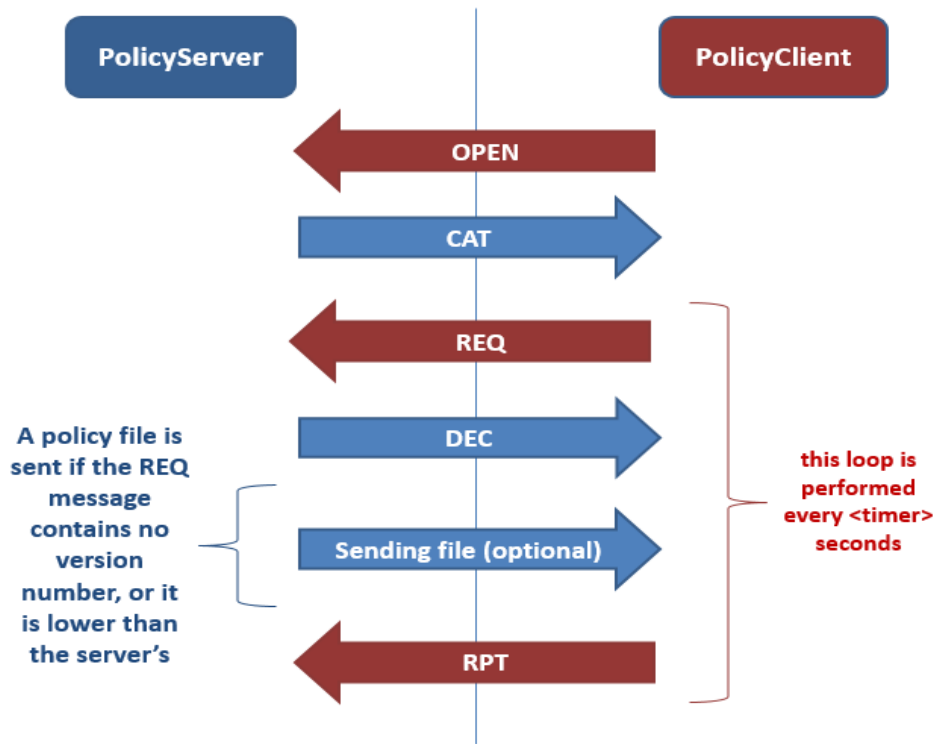


Figure 59 Client’s Periodic Check

To show how this works on our system, we configured our client with an interval of 60 seconds. After receiving the CAT message from the policy decision control server, the policy client sends a REQ message encoded with its policy file version number.

Figure 60 shows the policy client sending version 3.7 and receiving a DEC message from the server.

```

Client# ./PolicyClient 10.168.255.144 12345 office-client1 secret123 60
Connecting to the server.
hashBufferKey::buffer: <1060100017013016f66666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f66666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
  
```

```

hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
Sending OPEN request: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63
6c 69 65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31
38 37 33 63 36 30 36 32 32 37 35 39 63 33 >
Waiting for reply. Expecting CAT or CC.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
Successfully authenticated.
Packet received: <10 7 0 1 0 0 0 32 66 38 37 36 65 39 33 33 36 36 61 38
37 31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c
ParsePolicy::isAuthentic::hashGenerated:
<2f876e93366a8715e6871edc9e0bea0c>
ParsePolicy::isAuthentic::hashReceived :
<2f876e93366a8715e6871edc9e0bea0c>
SUCCESSFULLY AUTHENTICATED!!!!
CAT received.

--- Timer ---
Policy file exists. Version: %version 3.7
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e370>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e370736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e90862bae963b159948980491dc5899f
Sending REQ: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33 2e
37 0 65 39 30 38 36 32 62 61 65 39 36 33 62 31 35 39 39 34 38 39 38 30 34
39 31 64 63 35 38 39 39 66 >
Waiting for reply. Expecting DEC.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>
Successfully authenticated.
DEC received: <10 2 0 1 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
ParsePolicy::isAuthentic::hashGenerated:
<5792984e194562b83d0b9ea2c7069004>
ParsePolicy::isAuthentic::hashReceived :
<5792984e194562b83d0b9ea2c7069004>

```

```

SUCCESSFULLY AUTHENTICATED!!!
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.

```

Figure 60 Client Configured with Periodic Interval

The corresponding messages on the server side, shown in Figure 61, confirm the server's receipt of the REQ message with the most current policy file version number.

```

root@PolicyServer:~/Server# ./PolicyServer 12345 secret123
Waiting for clients...

New Client!
Waiting for query. Expecting OPEN.
read_policyPacket::Message-length: 17
hashBufferKey::buffer: <1060100017013016f666666963652d636c69656e74310>
hashBufferKey::buffer_temp:
<1060100017013016f666666963652d636c69656e74310736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e39b1526ed57cd5051873c60622759c3
ParsePolicy::isAuthentic::hashGenerated:
<e39b1526ed57cd5051873c60622759c3>
ParsePolicy::isAuthentic::hashReceived :
<e39b1526ed57cd5051873c60622759c3>
Successfully authenticated.
OPEN expected: <10 6 0 1 0 0 0 17 0 13 0 1 6f 66 66 69 63 65 2d 63 6c 69
65 6e 74 31 0 65 33 39 62 31 35 32 36 65 64 35 37 63 64 35 30 35 31 38 37
33 63 36 30 36 32 32 37 35 39 63 33 >
Client ID: office-client1
Looking in file: modem.txt
Finding Device type in: modem.txt
Looking in file: client.txt
Finding Device type in: client.txt
Looking in file: router.txt
Finding Device type in: router.txt
Looking in file: office.txt
Finding Device type in: office.txt
Policy file for this client: office.txt
hashBufferKey::buffer: <107010000>
hashBufferKey::buffer_temp: <107010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 2f876e93366a8715e6871edc9e0bea0c

```



```

Sending CAT: <10 7 0 1 0 0 0 0 32 66 38 37 36 65 39 33 33 36 61 38 37
31 35 65 36 38 37 31 65 64 63 39 65 30 62 65 61 30 63 >
Waiting for an event...

--- Client Request ---
read_policyPacket::Message-length: 15
hashBufferKey::buffer: <1010100015011012576657273696f6e20332e370>
hashBufferKey::buffer_temp:
<1010100015011012576657273696f6e20332e370736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: e90862bae963b159948980491dc5899f
ParsePolicy::isAuthentic::hashGenerated:
<e90862bae963b159948980491dc5899f>
ParsePolicy::isAuthentic::hashReceived :
<e90862bae963b159948980491dc5899f>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 15 0 11 0 1 25 76 65 72 73 69 6f 6e 20 33
2e 37 0 65 39 30 38 36 32 62 61 65 39 36 33 62 31 35 39 39 34 38 39 38 30
34 39 31 64 63 35 38 39 39 66 >
The client has an updated version.
hashBufferKey::buffer: <102010000>
hashBufferKey::buffer_temp: <102010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 5792984e194562b83d0b9ea2c7069004
Sending DEC: <10 2 0 1 0 0 0 0 35 37 39 32 39 38 34 65 31 39 34 35 36 32
62 38 33 64 30 62 39 65 61 32 63 37 30 36 39 30 30 34 >
Waiting for query. Expecting RPT.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...

```

Figure 61 Server Replying with DEC, Waiting for an Event

After completing the first exchange above, the policy client waits for 60 seconds to begin repeating the process by sending a REQ message. However, during the wait period we intentionally deleted the policy file on the client. Next, we show what happens during the second periodic check.

```

--- Timer ---
hashBufferKey::buffer: <101010000>
hashBufferKey::buffer_temp: <101010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 3905545f31ac0da0add88f3967db92d7
Sending REQ: <10 1 0 1 0 0 0 0 33 39 30 35 35 34 35 66 33 31 61 63 30 64
61 30 61 64 64 38 38 66 33 39 36 37 64 62 39 32 64 37 >
Waiting for reply. Expecting DEC.
read_policyPacket::Message-length: 9
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
Successfully authenticated.
DEC received: <10 2 0 0 0 0 0 9 0 5 0 2 0 61 66 36 39 66 64 38 63 32 64
61 63 66 35 65 33 33 63 36 37 61 65 30 64 37 32 35 38 63 62 62 30 >
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
ParsePolicy::isAuthentic::hashGenerated:
<af69fd8c2dacf5e33c67ae0d7258cbb0>
ParsePolicy::isAuthentic::hashReceived :
<af69fd8c2dacf5e33c67ae0d7258cbb0>
SUCCESSFULLY AUTHENTICATED!!!!
ReceiveFile::Starting to download file...
ReceiveFile::Size of file: 1029
ReceiveFile::rcv_Hashcode: ebf50ac4874dfa216dd8254fd98f8e82
ReceiveFile::FILE SUCCESSFULLY AUTHENTICATED
ReceiveFile::Download completed
Writing log
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
Sending RPT: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64 30
37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Sleeping.

```

Figure 62 Second Periodic Check, Client-Side

Figure 62 shows the policy client sending a REQ message, this time without a policy file version number, and in return, receiving a DEC message, followed by a file download.

The same is witnessed on the server side in Figure 63. The server is shown receiving a REQ message, but without a policy version number. Therefore, it

sends the policy file. Then, after receiving the RPT message from the policy client, it enters a waiting state.

```
--- Client Request ---
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <101010000>
hashBufferKey::buffer_temp: <101010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 3905545f31ac0da0add88f3967db92d7
ParsePolicy::isAuthentic::hashGenerated:
<3905545f31ac0da0add88f3967db92d7>
ParsePolicy::isAuthentic::hashReceived :
<3905545f31ac0da0add88f3967db92d7>
Successfully authenticated.
REQ expected: <10 1 0 1 0 0 0 0 33 39 30 35 35 34 35 66 33 31 61 63 30 64
61 30 61 64 64 38 38 66 33 39 36 37 64 62 39 32 64 37 >
No policy received. Sending policy file.
hashBufferKey::buffer: <10200000905020>
hashBufferKey::buffer_temp: <10200000905020736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: af69fd8c2dacf5e33c67ae0d7258cbb0
PolicyServer::DownloadPolicy::Sending file...
Checking file: <office.txt>
SendFile::Size of file: 1029 bytes
SendFile::snd_Hashcode: ebf50ac4874dfa216dd8254fd98f8e82 bytes
Waiting for query. Expecting RPT.
read_policyPacket::Message-length: 0
hashBufferKey::buffer: <103010000>
hashBufferKey::buffer_temp: <103010000736563726574313233>
hashBufferKey::password: secret123
hashBufferKey::md5_hash: 57125e33b634d0725baa326907a0a121
ParsePolicy::isAuthentic::hashGenerated:
<57125e33b634d0725baa326907a0a121>
ParsePolicy::isAuthentic::hashReceived :
<57125e33b634d0725baa326907a0a121>
Successfully authenticated.
Packet received: <10 3 0 1 0 0 0 0 35 37 31 32 35 65 33 33 62 36 33 34 64
30 37 32 35 62 61 61 33 32 36 39 30 37 61 30 61 31 32 31 >
Waiting for an event...
```

Figure 63 Second Periodic Check, Server-Side

During the next message exchange the client sends a REQ message, and since it has a policy file, the version number of that file will be encoded within the message. Then the whole process is repeated, guaranteeing that the policy client will always have the most recent copy of the policy file.

4.4 Summary

This chapter detailed the requirements for implementing the server/client components of the policy exchange system. It also detailed the policy exchange process, and showed, in detail the messages types and formats used for communications between the client and the server.

For illustration the chapter looked at various example scenarios and illustrated the message type exchange between the systems, emphasizing the outcome of each scenario.

In the first example we looked at an unknown device role to the server, and show the admission control process by the server. The illustration showed, when the server receives an OPEN message from an unknown client, the server blocks admission by sending Client Close (CC) message from to the client.

Example 2 captured the interaction between a client whose role is known to the server, but has no policy stored. This scenario for example occurs, when a device is being connected to the network for the first time or starts without a policy. The messages exchanged show that after admission the server pushed the appropriate policy file to the client.

In example 3 we looked at a known client with an existing policy that matches the version number of the policy stored on the server. After verification of the device role and the policy version number, the policy control server made decision to not update the client since both policies match. This matches expected behavior.

In example 4 the client has a policy that is older than the server's. After admission control, the client sent its policy version number to the server. The server detected the client's outdated version and pushed the newer policy to client.

Scenario 5 illustrated the case, when the network administrator modifies, edits, or deletes some elements of an existing policy, or even creates a new one, via the policy management interface. We showed successfully that the modified is immediately propagated from the server to all relevant devices.

In the final example, we illustrated the use of periodic checks to prevent a client being without a policy state, e.g., due to accidental or malicious deletion. When enabled, the periodic check triggered the policy client to send a REQ message every 60 seconds and the receiving server examined the REQ message for the encoded version number. When the same version number was reported the server only acknowledges the message. However, when no policy was reported, the server replied with a DEC message, followed by a policy file download.

In all, the chapter confirmed the concept of a reliable policy exchange system, what follows is a detailed explanation of the policy enforcement system.

CHAPTER 5: IMPLEMENTAING AND TESTING THE RUNTIME COMPLIANCE MANAGER

Key to our proposed Policy Compliance and Change Detection System is the ability to identify policy violations when they occur, and more importantly to make the system operator aware of these violations in real-time.

The Runtime Compliance Manager (RCM) is a local control module running on the policy client. It is responsible to continuously monitoring the network element configuration state, analyzing the current state and reporting discrepancies when applicable. The RCM is invoked whenever the network administrator enters the configuration state of the network element. The RCM then monitors and analyzes the entered configuration commands and parses the specified local policy to determine if there are any deviations from the desired state. Furthermore, if the RCM module determines that the entered command violates its operational or functional policy, the reporting module is called upon to generate the appropriate alarming or reporting action. This ensures that the network device is kept in compliance and that operators are notified in real-time.

This chapter explores the operation of the Runtime Compliance Manager (RCM), and how the real-time feedback plays a crucial role in maintaining the network elements in compliance. It also illustrates the different example scenarios.

5.1 RCM Design, Requirements and Installation

The RCM works in conjunction with the Policy Exchange and Management module, and relies on the downloaded policy file, a file called `policy.txt`, to enforce the rules. However, it can also operate in stand-alone mode as long as the system has a policy file to work with. This is because the policy file contains all the rules needed to be enforced on the policy client running the RCM. The RCM reads the rules and checks them against the configuration of the system. If it finds any mismatch, it generates a warning to inform the operator of the violation.

We implemented the RCM using standard C++ and assumed few of the Linux configuration stanzas for monitoring. However; the RCM was designed with the intention of adding new characteristics, in order to monitor in an easy way.

Figure 64Figure 64 conveys the basic system design: the RCM reads the policy file `policy.txt` to parse out the rules and checks the different system components for compliance. The system is examined against the policy by either a system call requesting the data from the system's kernel, or by comparing the data stored in the specific configurations file.

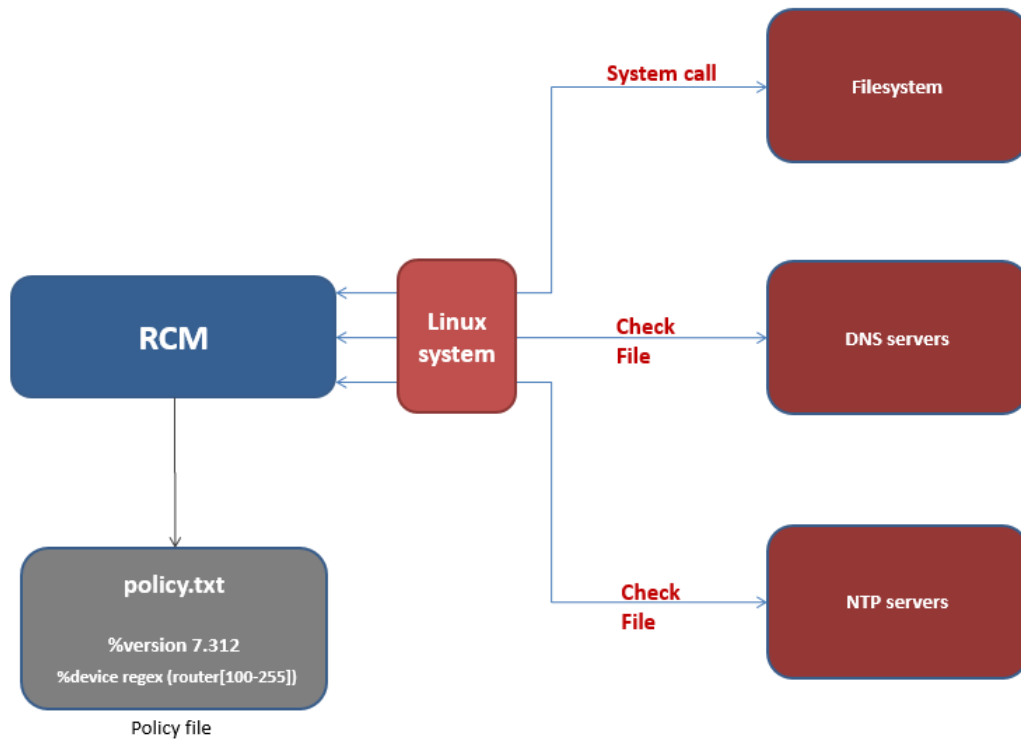


Figure 64 RCM Design

5.1.1 System Requirements

The RCM is written in C/C++ for UNIX-like environments, high level of the source code is provided in Appendix B of this research.

5.1.2 Installing a Running RCM

Once the source code (provided in Appendix B) has been compiled using the make command, the next step is to execute the binary by writing the following line:

```
./PolicyEnforcement
```


The first thing the RCM will do is the check for the 'policy.txt' file in the same folder with the binary. If there is not a policy file, the code will wait until it the policy file is detected. This file can be either manually copied to the policy client or downloaded using the policy exchange and management system described in Chapter 4.

Next, the RCM checks for compliance and prints an informational page using stdout and syslog (more about stdout and syslog in section 5.1.2 Installing a Running RCM). This information is independent, which means that by looking at the last report, the system administrator or user can have a good understanding of the current system's compliance state and which rules are in conflict. To avoid printing the same logs repeatedly in the stdout and in the syslog, the program has the intelligence to recognize repeated warnings and avoid repeated messages.

In Figure 65 we illustrate the basic behavior of the program. In essence, it reads the policies from the policy file and checks the monitored stanzas on the policy client. If the client stanzas are in agreement with the policy, nothing happens; but for each violated policy, a warning is generated in the stdout and in the syslog.

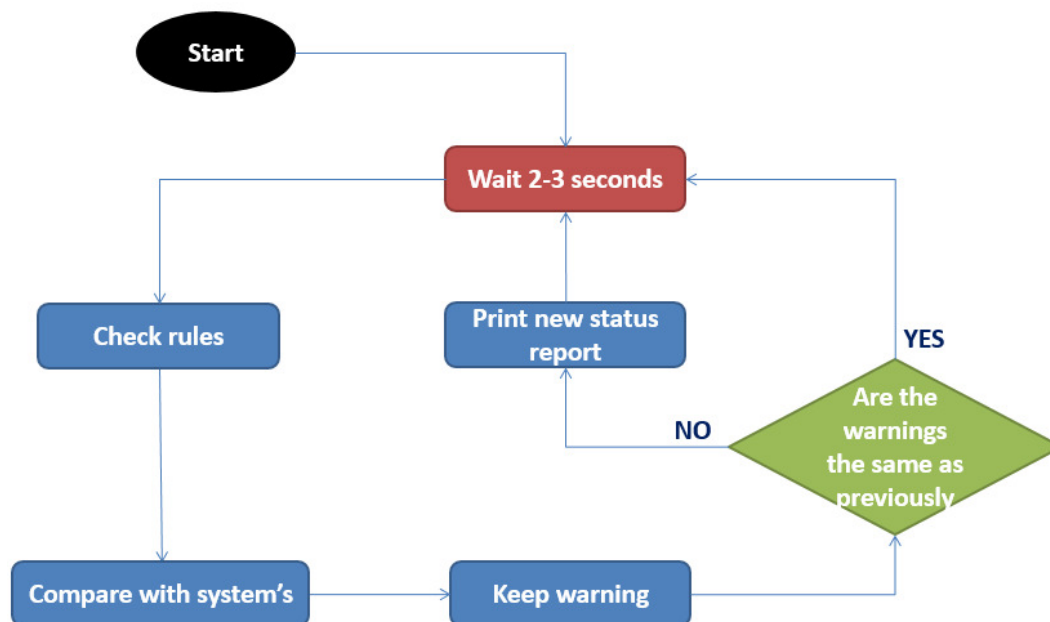


Figure 65 RCM Flow Chart

The source files which make the RCM program run appropriately are provided in Appendix B.

5.1.3 Logs

The logs play a central role in our automated compliance and change detection system. They are the mechanisms to inform the system administrator or the change owner of the policy violations when they occur.

The logs are displayed through the syslog and stdout. Syslog is a way for network devices to send event messages to a logging server, or to the local file [68]. On the development system, the logs are located in the /var/log/syslog directory. Stdout is standardized stream of data, which consist of plain text that

can be sent to devices (e.g., display monitors or printers) or be further processed by other programs [69]. Our implementation directs the output to the screen.

The first thing to notice in the log is the first line. The first line specifies whether the logs were generated because of a new policy file (“New Policy File detected”) or a system change (“Machine Report”).

Let us look at the example in Figure 66 of the stdout log of the policy enforcement:

```
<----- New Policy File detected ----->
#19-NetInt WARNING(address): 192.168.1.107 does not match (exclude).
#57-NTP    WARNING(server): 0.rhel.pool.ntp.org Not found.
#67-NTP    WARNING(server): 1.rhe2.pool.ntp.org Found (exclude).
#183-DNS   WARNING(nameserver): 127.0.0.1 Found (exclude).
#184-DNS   WARNING(nameserver): 127.0.0.1 Found (exclude).
<----->
```

Figure 66 New Policy File Detected

From the first line, we can tell that the report was generated because the policy client has detected a new policy file. Then we have the specific details for the warnings. First, the number at the beginning of each line corresponds to the line number in the policy file. Following the line number, more details about the source of the warning is shown.

The next example in Figure 67 shows a machine report generated by the RCM due to a user-made system change that does not comply with the policy.

```
<----- Machine Report ----->
#19-NetInt WARNING(address): 192.168.1.107 does not match (exclude).
#20-NetInt WARNING(netmask): 255.255.255.0 does not match (exclude).
#56-NTP     WARNING(server): 0.0.0.1 Not found. [Caused by system change]
<----->
```

Figure 67 Machine Report

The words “Machine Report” in the first line informs the administrator that the log was generated due to a system change, not by a new policy file as seen in the previous example. What follows is the same information as in the example before. However; the line 56 message, “[Caused by system change],” reveals that the warning was generated because the NTP server's file was modified.

5.2 Policy File Syntax

Much of section 3.4 Common Policy Language Format of this dissertation was devoted to the Common Policy Language used to represent the device policies. In order for the RCM program to understand the written policies and monitor the system appropriately, the file should follow several syntax rules. In this section we will explain some of these rules, and how they apply to our implementation.

In Figure 68, “NetworkInterface,” “NItag” and “DNS” are examples of section names that can be used to distinguish different sections of the policy file. It is worth mentioning that the first two lines are not used in the policy enforcement process, as they are just the version and the device name, and those parameters are irrelevant in this layer of the system enforcement. However, they play a major role in the policy file exchange process described in Chapter 4.

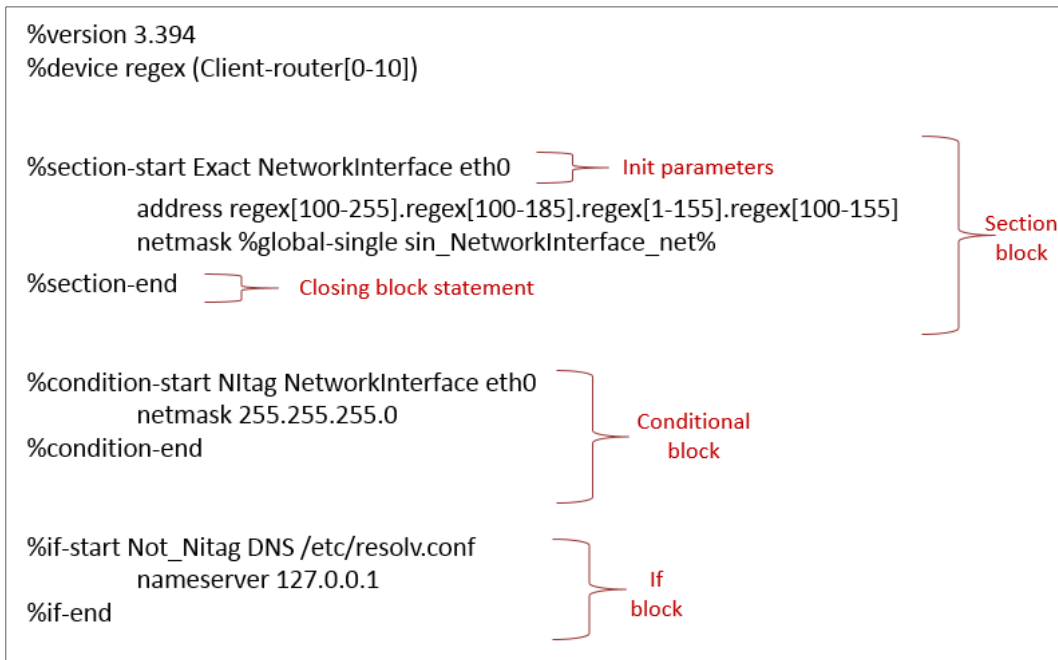


Figure 68 Policy File Example

We have three kinds of blocks: Section, Conditional and If. Those are all the kind of blocks that the proposed common policy language supports. Each block starts with initialization parameters (init parameters) and ends with (closing block statement) with its individual section name. This is important to keep in mind for referencing different section blocks throughout the policy file.

In the Figure 68 example, the section block “NetworkInterface” uses regular expression to express the permissible network interface IP addresses. The example expects the network interface on the system, namely, eth0, to have an address range of 100-255 in the first octet, 100-185 in the second, 1-155 in the

third, and 100-155 in the fourth octet. The subnet mask of the interface global signal can hold any value. Thus, if the eth0 of the policy client is configured within the specified range, then the policy client is in compliance with our policy. Otherwise, it is not.

The conditional block NItag is a Boolean condition that evaluates to true or false. In our example, the expression is true if the value of the eth0 subnet mask is 255.255.255.0. The expression is used in conjunction with the if block.

The if block in our example expects the system DNS nameserver to be set to 127.0.0.1 if, and only if, the conditional block is evaluated as false. In other words, if the system's eth0 subnet mask is not 255.255.255.0, and the DNS server is not set to 127.0.0.1, then the RCM will throw a warning.

Now that we have seen the general structure of the policy file with its blocks, let us get into the details of specifying each of them.

We start by explaining our implementation of the section block. To give ourselves a wider range of test variables, we decided to monitor the following functions: Interface variables, NTP, hostname, file system size limit (FSSL), domain name system (DNS), secure shell (SSH), HOSTS file and shell environment (ENV) variables. Table 6 shows all the monitored features. We will use the table as a reference for all the possible ways to create the initialization parameters (init-parameter) sentences of the section block:

Table 6 Section Block Parameters

Section Keyword	Predefined Keywords	Feature	Extra parameter
%section-start %section-end	Exact Exclude Ignore	NetworkInterface	<Name of the Interface>
		NTP	<Path-file of NTP servers>
		Hostname	-
		FSSL	<FileSystem>
		DNS	<Path-file of DNS servers>
		SSH	<Path-file of SSH config file>
		ENV	-
		HOSTS	<Path-file of hosts>

Unlike any other language block defined in the Common Policy Language, the section block has a special parameter called 'predefined keywords,' and each of the currently defined keywords has a special meaning. They are handled accordingly by the RCM. Table 7 describes the predefined keywords and their meanings.

Table 7 Predefined Keywords

Predefined Keyword	Meaning
Exact	When set, the entire content of this block is compared with the policy client's settings. If they are different, a warning is displayed; otherwise, nothing happens.
Exclude	The RCM uses the 'exclude' keyword to check the client's configuration for the specified code block. If it exists on the system, a warning is displayed. In other words, the client should not have the section configured.
Ignore	The RCM ignores and does not check the client for this code section. Helpful for troubleshooting.

Table 8 summarizes all of the possible ways to create the init-parameter sentences of a conditional block.

Table 8 Conditional Block Parameters

Conditional Keyword	ID to identify the conditional	Feature	Extra parameter
%condition-start %condition-end	<Conditional ID>	NetworkInterface	<Name of the Interface>
		NTP	<Path-file of NTP servers>
		Hostname	-
		FSSL	<FileSystem>
		DNS	<Path-file of DNS servers>
		SSH	<Path-file of SSH config file>
		ENV	-
		HOSTS	<Path-file of hosts>

The conditional block requires a conditional identifier to be referenced by the Common Policy Language, especially when writing the if block.

The if block in Table 9 it expects a Boolean condition. If the evaluated condition returns true, then policies defined within the if block are compared to the device's configuration. The negation operator (Not) returns the opposite of the given Boolean expression.

Table 9 If Block Parameters

If keyword	ID related to the condition	Feature	Extra parameter
%if-start %if_end	<Conditional ID>	NetworkInterface	<Name of the Interface>
		NTP	<Path-file of NTP servers>
		Hostname	-
		FSSL	<FileSystem>
	Not_<Conditional ID>	DNS	<Path-file of DNS servers>
		SSH	<Path-file of SSH config file>
		ENV	-
		HOSTS	<Path-file of hosts>

5.2.1 Variables

Variables in the policy files are treated as global variables, which means that they can be accessed in any part of the policy file. To access them, we insert the variable name between percent signs (%), and when declared, the variable is replaced by its value.

There are two kinds of variables: global-single and global-list. The difference between them is:

- 1) A global-single variable holds only one value, while a global-list can hold multiple variables.

- 2) When a global-list variable is declared and there is nothing assigned to it, a warning will be displayed. On the other hand, with global-single, nothing will happen and the variable will have a null value.

5.2.2 Printing Variables

The printing functionality was not specified in the Common Policy Language specification described in section 3.4 of this dissertation. Rather, it was added for the Linux environment as a troubleshooting tool.

By printing the values of the variables on the screen during the testing or the troubleshooting phase, the operator is quickly able to verify if the policy is operating as intended.

To use the printing functionality, the variables must be placed in the print block, between the `%section-start` and `%section-end`. The following example in Figure 69 illustrates the concept:

```
%section-start Exact PRINT
    %sin_NetworkInterface_add%
    %sin_NetworkInterface_net%
    %lst_NTP_serv%
    %sin_NTP_serv%
%section-end
```

Figure 69 Printing Block Example

As shown in the above example, the variables are placed between the % signs. An undeclared variable returns white space.

5.3 Client-Monitored Features

The CRM can be tailored to monitor any networking device. However, as explained in section 1.4 of this dissertation we are limited with our test environment due to the proprietary nature of the systems available on the market.

We chose Linux for our proof of concept system because its open nature provides ready access to system configurations. The Linux system servers and only as an example to demonstrate the capabilities of the CRM. Principally, this framework can be ported to any system or device.

In the following examples the CRM will monitor eight different features on a Linux system. The features are explained next:

5.3.1 Network Interface

With the network interface feature, we are concerned with monitoring the layer three protocol interface IP and the interface subnet mask, whether the information was obtained by means of the dynamic host configuration protocol (DHCP), or by static configuration.

The interface parameter in the Common Policy Language may be expressed as a value, as a regular expression, or as a global-single variable. Table 10 shows the different configuration options.

Table 10 Network Interface Block

Parameter 1	Parameter 2	Parameter 3
address	X.X.X.X	-
netmask	%global-single	<Unique var name>%

X.X.X.X = Dotted decimal notation

The following examples show the various ways to represent the interface IP and subnet mask value:

Examples:

- address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
- address 192.168.1.125
- netmask 255.255.255.0
- address %global-single IP_NetworkInterface_var%
- netmask %global-single Net_NetworkInterface_var%

The first example uses regular expression to represent the four octets of the IP address; the second and the third use the absolute value of the IP and mask, and the fourth and the fifth examples use global-single variables to represent the IP and the mask. Recall that the global-single variable can have one and only one value.

5.3.2 NTP

The network time protocol (NTP) provides a means for clients to synchronize their clocks with a trusted time source using the TCP/IP protocol.

Once the protocol is installed on a Linux machine, it can be configured to both synchronize with upstream servers and provide time services to other machines on the local network.

Correct time reporting is crucial to some system functions, such as syslog, which needs the correct time stamp for its system event reporting; and the DNS server, which will not accept a zone transfer when the idea of time between the master and the server is significantly different [70]. We thought it would be important to monitor the NTP configuration to ensure that the time source servers configured on our policy clients are authorized and trustworthy. Table 11 demonstrates the different NTP configuration options.

Table 11 NTP Block

	Parameter 2	Parameter 3
	X.X.X.X	-
	DNS Name (e.g. 0.rhel.pool.ntp.org)	-
	%global-single	<Unique var name>%
	%global-list	<Unique var name>%

X.X.X.X = Dotted decimal notation

The following examples show the various ways to configure the NTP server variable.

Examples:

- server 192.168.1.3
- server 2.rhel.pool.ntp.org
- server %global-single sin_NTP_serv%
- server %global-list lst_NTP_serv%

The first example above assigns a dotted decimal value to the NTP server, the second uses a hostname, the third uses a global-single value and the fourth uses a global-list value. The difference between the global-single and the global-list is that the global-single holds only one value for the parameter server, while the global-list holds one or more values for the server.

5.3.3 Hostname

In a Linux operating system, a system's hostname usually has a corresponding entry in the domain name system (DNS), and some services use the hostname to identify the system that they are running on. If the administrator has not set up a hostname during installation, one will be assigned to the machine [71]. This is not an issue for most system users. However, in controlled or production environments, a managed hostname assignment plays a major role in stability. This is particularly true for software that requires a valid fully

qualified domain name, or FQDN, for the hostname to be used by their licensing verification systems.

Hostname is another system variable that we selected to monitor with our compliance system. The hostname parameter is set either to a string or to a global-single variable. Table 12 lists all the possible ways to write a monitor statement for the hostname feature.

Table 12 Hostname Block

Parameter 1	Parameter 2	Parameter 3
name	string (e.g. machine-ubuntu)	-
	%global-single	<Unique var name>%

The following two examples illustrate the use of the parameters:

Examples:

- name Lab-Client
- name %global-single sin_Hostname_nam%

The first example uses a string that specifies the exact match of the system's hostname, and the second is a global-single variable that stores the configured hostname.

5.3.4 File System Size Limit (FSSL)

The File System Size Limit (FSSL) feature monitors the amount of disk space available on the file system containing each file name argument. The feature relies on the Linux command `df -h` to evaluate the percentage of the disk space used. For example, in Figure 70 the output the shows 41% of disk `sda1`'s space is in use.

Filesystem	Size	Used	Avail	Use%	Mounted on
<code>/dev/sda1</code>	20G	7.7G	12G	41%	<code>/</code>
<code>udev</code>	241M	4.0K	241M	1%	<code>/dev</code>
<code>tmpfs</code>	100M	756K	99M	1%	<code>/run</code>
<code>none</code>	5.0M	0	5.0M	0%	<code>/run/lock</code>
<code>none</code>	248M	76K	248M	1%	<code>/run/shm</code>

Figure 70 FSSL Disk Usage Example

Our implementation of the FSSL block looks for the configured limit value and compares the parameter to the actual disk usage. If it exceeds it, then the RCM will generate a report. Table 13 shows how to write a monitor statement for the FSSL feature.

Table 13 FSSL Block

Parameter 1	Parameter 2	Parameter 3
limit	Number (e.g. 15)	-
	%global-single	<Unique var name>%

In the following examples, the limit refers to disk usage, and it can be expressed as an integer or as a global-single variable.

Examples:

- limit 42
- limit %global-single sin_FSSL_lim%

5.3.5 DNS

The domain name system maps between hostnames and IP addresses and has two components, a client side and a server side. The client side is our interest for this implementation. It provides the ability to resolve names and addresses by making requests to one or more DNS servers to obtain the IP address of a host.

Typically, several servers are configured on the client so that, should a failure occur with a particular DNS server, other backup systems will respond to the client request ([72], pp. 297). In Table 14, we show the nameserver configuration which specifies that the DNS server can be expressed as an IP

address in dotted decimal notation or as either a global-single or a global-list variable.

Table 14 DNS Block

Parameter 1	Parameter 2	Parameter 3
nameserver	X.X.X.X	-
	%global-single	<Unique var name>%
	%global-list	<Unique var name>%

X.X.X.X = Dotted decimal notation

The first example below displays the use of an IP address to represent the DNS server. This is an exact match in our policy. The second example assumes only one configured value, while the global-list example allows the system to check for multiple entries for nameserver.

Examples:

- nameserver 127.0.0.1
- nameserver %global-single sin_dns_nams%
- nameserver %global-list lst_dns_nams%

5.3.6 SSH

The secure shell service (SSH) provides an encrypted communications channel between two hosts for remote system access. This includes file copy and terminal access for executing arbitrary commands on the remote system ([72], pp.

241). SSH has two components: a server program and a client program. For the implementation and testing of our system, we will focus on the server side.

The server program listens to incoming SSH connection requests, authenticates those requests, and provides access to the command line interface of the system.

We installed OpenSSH on our test machines because OpenSSH is a free implementation of the SSH protocol and is widely used and re-useable by everyone under a BSD license [73].

Our SSH code block allows for the monitoring of any of the variables defined in the configuration file of the protocol. Table 15 illustrates the possible ways to write a monitor statement for the SSH feature.

Table 15 SSH Block

Parameter 1	Parameter 2	Parameter 3
<SSH variable> (e.g. HashKnownHosts)	Value string (e.g. yes)	-
	%global-single	<Unique var name>%

The examples below illustrate the use of string and variable parameters. ServerKeyBits has a string value, while ListenAddress is a variable.

Examples:

- ServerKeyBits 768
- ListenAddress %global-single sin_SSH_Listen%

5.3.7 Environment Variables (ENV)

An environment variable is a named object that contains data used by one or more applications. In simple terms, it is a variable with a name and a value [74]. These variables are known as shell environment variables, which can be used by various commands to get information about the user environment, such as the type of system that is running, the user's home directory and the shell in use.

Environment variables are used by Linux operating systems to help tailor the computing environments of the systems, and include helpful specifications and setup, such as the default location of all executable files in the file system, the default editor that should be used, and the system locale settings and software libraries [75].

Although there are several standard variables in Linux environments, such as PATH, HOME, SHELL and TERM to name few, the CRM is customized to monitor all of the variables. Table 16 shows the various options for representing the environment variable in the Common Policy Language.

Table 16 ENV Block

Parameter 1	Parameter 2	Parameter 3
<ENV variable> (e.g. SHELL)	Value string (e.g. /bin/bash)	-
	%global-single	<Unique var name>%

Next, we show a few examples of the syntax usage:

Examples:

- TERM vt100
- LANG en_US.UTF-8
- SSH_TTY /dev/pts/2
- SSH_TTY %global-single sin_ENV_sshTTY%
- LOGNAME guest

5.3.8 Hosts

The hosts file is a static mapping of IP to hostnames, and is consulted by the system for hostname-to-IP resolution before the DNS lookup, which could speed up the lookup process.

In the HOSTS feature, we can monitor all the values in the configuration file for Hosts in Linux machines. In Table 17 we list all the possible ways to write a monitor statement for the hosts.

Table 17 Hosts Block

Parameter 1	Parameter 2	Parameter 3
<IP address>	string	-
	FQDN (e.g. client.acme.com)	-
	%global-single	<Unique var name>%
	%global-list	<Unique var name>%

X.X.X.X = Dotted decimal notation

In the examples below we see the use of a string in the first example, the fully qualified domain name in example two, a global-single variable in example three, and finally, a global-list in example four. This is because it is possible for one IP to have multiple hosts names.

Examples:

- 127.0.0.1 localhost
- 10.168.255.1 mail.acme.com
- 127.0.0.1 %global-single sin_HOSTS_127%
- 127.0.0.1 %global-list lst_HOSTS_127%

5.4 Example Scenarios

The Runtime Compliance Manager (RCM) is responsible for monitoring and enforcing the device policy on the policy client. Central to its operation is the policy file. In Chapter Four we detailed its exchange and maintainability process. We will build on the policy exchange process to implement the next stage of the

Automated Policy Compliance and Change Detection System: policy enforcement and discrepancy reporting.

In this section we will use our Linux system to run the RCM program, and customize policies to illustrate the client-monitored features we discussed earlier in section 5.3. The practical test scenarios are staged in different ways. This is partly so we can examine system change events that may or may not trigger policy compliance warnings, and partly to illustrate the Common Policy Language use of various system-monitored features in the different Common Policy Language sections.

5.4.1 Section with 'Exact' Keyword

Our first test illustrates the usage of the section delimiter with the predefined keyword 'Exact.' The section delimiter can match any part of the configuration stanza, such as interface, NTP, DNS...etc. The section starts with the keyword 'section-start,' and ends with 'section-end.' The predefined keyword 'Exact' in the policy looks for an exact match in the system. Let us look at the example provided in Figure 71 below:


```

1 %version 3.394
2 %device regex (Client-router[0-10])
3 /*##### NetworkInterface #####
4 %section-start Exact NetworkInterface eth0
5     address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
6     netmask 255.255.255.0
7 %section-end
8 /*##### NTP #####
9 %section-start Exact NTP /etc/ntp.conf
10     server 0.rhel.pool.ntp.org
11 %section-end
12 /*##### Hostname #####
13 %section-start Exact Hostname
14     name client
15 %section-end
16 /*##### FSSL (File System Size Limit) #####
17 %section-start Exact FSSL /dev/sda1
18     limit 42
19 %section-end
20 /*##### DNS #####
21 %section-start Exact DNS /etc/resolv.conf
22     nameserver 8.8.4.4
23 %section-end
24 /*##### SSH #####
25 %section-start Exact SSH /etc/ssh/sshd_config
26     KeyRegenerationInterval 3600
27     ServerKeyBits 1024
28 %section-end
29 /*##### ENV (Environment variable) #####
30 %section-start Exact ENV
31     TERM vt100
32     LANG en_US.UTF-8
33 %section-end
34 /*##### HOSTS #####
35 %section-start Exact HOSTS /etc/hosts
36     127.0.1.1 client-virtualBox
37 %section-end

```

Figure 71 Policy File: Section with 'Exact' Keyword

In Figure 71, we see an example of the section block code with the use of the 'Exact' predefined keyword for each of the system features we are interested in monitoring. The use of regular expression (regex) on line 5 is a powerful way to represent the IP address of the eth0 interface; the RCM will match the pattern if the system IP's first octet falls between 100 and 255, the second octet between

100 and 185, and so on. The remaining sections use a string to represent each of the variable values.

Now when the RCM is run, it generates the following compliance report.

```
root@Client:~/Client#
root@Client:~/Client# ./PolicyEnforcement
PolicyEnf::run():Start runing

<----- New Policy File detected ----->
#5-NetInt WARNING(address): 10.168.255.149 does not match.
#10-NTP WARNING(server): 0.rhel.pool.ntp.org Not found.
#14-Hname WARNING(name): client Does not match.
#22-DNS WARNING(nameserver): 8.8.4.4 Not found.
#27-SSH WARNING(ServerKeyBits): 1024 Not found.
#31-ENV WARNING(TERM): vt100 Does not match.
#36-HOSTS variable WARNING(127.0.1.1): client-virtualBox Not found.
<----->
```

Figure 72 Section with 'Exact' Report 1

The first thing worth noting in Figure 72 is that the report was generated due to the detection of a new policy file. That is because the policy decision control server has pushed a new policy file to the client, as was explained in Chapter 4.

Next, we notice the warnings, each with a line number corresponding to our policy in Figure 71. Below are the details for each of the warning lines:

#5 is a warning that the current system IP address does not match the NetworkInterface section of the policy. Recall that the policy expects [100-255],[100-185],[1-155],[100-155] for the IP. From Figure 73 below, we see that the system's current IP address is 10.168.255.149, which is a violation of the first

and third octets set by the policy. However; the subnet mask on line 6 of the policy is a match, which is also visible in Figure 73:

```
root@Client:~/Client# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e0:7e:a5
          inet addr:10.168.255.149  Bcast:10.168.255.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1551 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:209745 (209.7 KB)  TX bytes:1278 (1.2 KB)

root@Client:~/Client#
```

Figure 73 Client's IP Address

#10 is a warning that the specified NTP server in the policy does not exist on the system. Line 10 of the policy file specifies an NTP source of “0.rhel.pool.ntp.org,” however; the system’s NTP configuration does not specify 0.rhel.pool.ntp.org as a time source server, as we see in Figure 74 below:

```

root@Client:~/Client# more /etc/ntp.conf
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org

```

Figure 74 Client's NTP Servers

#14 is a warning about a mismatch between the hostname configured on the system and what the policy file expects it to be. In Figure 71, the policy is configured for a hostname of 'client,' while the system is actually configured with 'Client.'

Figure 75, shows the actual configuration on the policy client.

```

root@Client:~/Client# hostname
Client
root@Client:~/Client# more /etc/hostname
Client
root@Client:~/Client#

```

Figure 75 Client's Hostname

#22 is a warning about a violation of the DNS configuration on the system. Line 22 of Figure 71 requires the system to be configured with 8.8.4.4 for a nameserver. However, from the report we can conclude that the system does not have the server configured and therefore it is not in compliance. Figure 76 shows the system's actual DNS configuration.

```
root@Client:~/Client# more /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.168.255.1
```

Figure 76 Client's DNS Configuration

#27 is a warning about the SSH feature. In Figure 71 we configured two parameters for this section, `KeyRegenerationInterval` with a value of 3600 on line 26, and `ServerKeyBits` with value of 1024 on line 27 of the policy file. However, as seen in Figure 72, a report was generated for `ServerKeyBits` only.

After examining the client's actual SSH configuration in Figure 77, we see that the system's `KeyRegenerationInterval` is configured with a matching value of 3600, while the `ServerKeyBits` is configured with 768.

```

root@Client:~/Client# more /etc/ssh/sshd_config
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768
..
<truncated>

```

Figure 77 Client SSH Configuration

#31 is a warning about the environment variables. In the policy file in Figure 71, we defined two parameters for this section: TERM and LANG, however the RCM is reporting that the TERM (terminal) type, vt100, is a mismatch between the policy file and what is actually configured on the system. By examining Figure 78 below, we see that the mismatch has occurred because the system's TERM type is set to xterm.

```
root@Client:~/Client# env
TERM=xterm
SHELL=/bin/bash
SSH_CLIENT=10.168.255.70 51250 22
SSH_TTY=/dev/pts/0
USER=root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33
MAIL=/var/mail/daniel
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/root/Client
LANG=en_US.UTF-8
SHLVL=1
<truncated>
```

Figure 78 Client's ENV Variables

#36 is the last warning in the report, and it is in reference to the entry in the system's hosts file. Line 36 of the policy file supposes that the system should have an IP to host mapping of client-virtualBox, but according to the report, there is a violation to the policy. Our verification of the system's configuration in **Figure 79** reveals that the system is configured differently, specifically, the hosts file entry is set to 'client' instead of 'client-virtualBox.'

```
root@Client:~/Client# more /etc/hosts
127.0.0.1    localhost
127.0.1.1   client
```

Figure 79 Client's Hosts File

Now let us make some changes to the system and see how they will affect the policy compliance reporting. We will start by making changes to the network interface IP. The current eth0 interface IP is 10.168.255.149, as observed in Figure 73, and we will change it to match the regular expression in the policy file in Figure 71. Specifically, we will change the IP address to 100.100.1.100, which matches the regular expression `regex[100-255].regex[100-185].regex[1-155].regex[100-155]`.

Figure 80, shows the commands we entered to change the IP, and shows the confirmation of the change we made using the `ifconfig` command.

```
root@Client:~/Client# ifconfig eth0 100.100.1.100 netmask 255.255.255.0

root@Client:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e0:7e:a5
          inet addr:100.100.1.100  Bcast:100.100.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13003 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1728683 (1.7 MB)  TX bytes:1278 (1.2 KB)

root@Client:~#
```

Figure 80 System Interface IP Changed

Immediately after the change, the RCM generated the report in Figure 81. Two things are different in this report when compared to Figure 72. First, the report type is machine, whereas in Figure 72 it was new policy file. Second, we notice that #5 is excluded from the report. That is because the interface IP is now in compliance with the policy.

```
<----- Machine Report ----->
#10-NTP    WARNING(server): 0.rhel.pool.ntp.org Not found.
#14-Hname  WARNING(name): client Does not match.
#22-DNS    WARNING(nameserver): 8.8.4.4 Not found.
#27-SSH    WARNING(ServerKeyBits): 1024 Not found.
#31-ENV    WARNING(TERM): vt100 Does not match.
#36-HOSTS  variable WARNING(127.0.1.1): client-virtualBox Not found.
<----->
```

Figure 81 Network Interface in Compliance

To correct the NTP compliance issue on #10 of the report, the system NTP server configuration must include the server 0.rhel.pool.ntp.org listed on line 10 of the policy file (Figure 71). The output in Figure 82 shows the change after we added the new server to the NTP server list.

```

root@Client:~# more /etc/ntp.conf
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server 0.rhel.pool.ntp.org

```

Figure 82 System NTP Servers Updated

The RCM generates the Machine report in Figure 83, this time excluding the NTP compliance violation.

```

<----- Machine Report ----->
#14-Hname  WARNING(name): client Does not match.
#22-DNS    WARNING(nameserver): 8.8.4.4 Not found.
#27-SSH    WARNING(ServerKeyBits): 1024 Not found.
#31-ENV    WARNING(TERM): vt100 Does not match.
#36-HOSTS  variable WARNING(127.0.1.1): client-virtualBox Not found.
<----->

```

Figure 83 NTP Is in Compliance

Correcting the remaining compliance violations seen in Figure 72 can be accomplished by bringing into compliance each of the features listed by

modifying the feature's relative configuration as demonstrated earlier. For brevity, we will not show the remaining examples.

The power of our automated compliance and change detection system's feature of alerting and notifying the system operator in real-time is clearly demonstrated in the examples above.

5.4.2 Section with 'Exclude' Keyword

The 'exclude' predefined keyword has the opposite effect of the 'exact' keyword discussed in the earlier section. The 'exclude' keyword specifies that the configuration lines within the section should not exist on the policy client device. These are any commands the network or the system administrator may deem undesirable, perhaps due to a security risk or the potential for triggering a software bug, or simply because it is bad practice.

To better illustrate the use of the 'exclude' command, we wrote a new policy for our system, seen in Figure 84. It includes a sample for each of the monitored features defined by the RCM. For NetworkInterface, for example (lines 5 and 6 of the policy) an alarm would be generated if the eth0 IP address fell anywhere within the regular expression. It would also trigger an alarm if the eth0 subnet mask was 255.255.255.0. Similarly, the NTP section would trigger an alarm if our policy client listed 0.rhel.pool.ntp.org as one of its configured NTP servers.

The other sections - Hostname, FSSL, DNS, SSH, ENV and Hosts - would each trigger a separate alarm if any of their configured parameter values appeared on the system.

```
1 %version 3.395
2 %device regex (Client-router[0-10])
3 /*##### NetworkInterface #####
4 %section-start Exclude NetworkInterface eth0
5     address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
6     netmask 255.255.255.0
7 %section-end
8 /*##### NTP #####
9 %section-start Exclude NTP /etc/ntp.conf
10     server 0.rhel.pool.ntp.org
11 %section-end
12 /*##### Hostname #####
13 %section-start Exclude Hostname
14     name Client
15 %section-end
16 /*##### FSSL (File System Size Limit) #####
17 %section-start Exclude FSSL /dev/sda1
18     limit 12
19 %section-end
20 /*##### DNS #####
21 %section-start Exclude DNS /etc/resolv.conf
22     nameserver 8.8.4.4
23 %section-end
24 /*##### SSH #####
25 %section-start Exclude SSH /etc/ssh/sshd_config
26     KeyRegenerationInterval 3600
27     ServerKeyBits 1024
28 %section-end
29 /*##### ENV (Environment variable) #####
30 %section-start Exclude ENV
31     TERM xterm
32     SHELL /bin/bash
33 %section-end
34 /*##### HOSTS #####
35 %section-start Exclude HOSTS /etc/hosts
36     127.0.1.1 client
37 %section-end
```

Figure 84 Policy File: Section with 'Exclude' Keyword

Next, the RCM checks the newly-downloaded policy file rules against the system's configuration and generates a report for the sections that are non-compliant. Figure 85 lists the RCM check results.

```
<----- New Policy File detected ----->
#6-NetInt WARNING(netmask): 255.255.255.0 does not match (exclude).
#10-NTP    WARNING(server): 0.rhel.pool.ntp.org Found (exclude).
#14-Hname  WARNING(name): Client Matches (exclude).
#22-DNS    WARNING(nameserver): 8.8.4.4 Found (exclude).
#26-SSH    WARNING(KeyRegenerationInterval): 3600 Found (exclude).
#31-ENV    WARNING(TERM): xterm Matches (exclude).
#32-ENV    WARNING(SHELL): /bin/bash Matches (exclude).
#36-HOSTS  variable WARNING(127.0.1.1): client Found (exclude).
<----->
```

Figure 85 Section with 'Exclude' Report

#6 is a warning about the network mask configuration on line 6 of the policy file in Figure 84. According to the policy file, the system's eth0 interface should not have a subnet mask of 255.255.255.0.

Next, in Figure 86, we examine the system's network configuration, and we clearly see the mask is indeed in violation of the policy, hence the warning. However; it is worth noting that in regards to line 5 of the policy, the IP address did not trigger an alarm. That is because the configured eth0 IP address, 10.168.255.149, does not fall within the range of the regular expression [100-255].regex[100-185].regex[1-155].regex[100-155].

```
root@Client:~/Client# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e0:7e:a5
          inet addr:10.168.255.149  Bcast:10.168.255.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:24315 errors:0 dropped:0 overruns:0 frame:0
          TX packets:743 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3206564 (3.2 MB)  TX bytes:96015 (96.0 KB)

root@Client:~/Client#
```

Figure 86 Network Interface Mask

#10 of the report is a warning about the presence of the NTP server 0.rhel.pool.ntp.org on the system, which according to line 10 of the policy file should have been excluded. Once again, we check the system for the validity of the report, and we observe in Figure 87 that the system's NTP configuration does include the forbidden server 0.rhel.pool.ntp.org on its list.

```

root@Client:~/Client# more /etc/ntp.conf
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server 0.rhel.pool.ntp.org

```

Figure 87 NTP Server Present

#14 is the Hostname, and this time the hostname 'Client' does match both the policy (line 14 in Figure 84) and the system's Hostname configuration. However, since the keyword 'exclude' was used in the policy, the RCM generated the warning. Further verification of the system's configuration in Figure 88 confirms our proposition.

```

root@Client:~/Client# hostname
Client
root@Client:~/Client# more /etc/hostname
Client
root@Client:~/Client#

```

Figure 88 Hostname Is a Match

For brevity, we will stop the 'exclude' examples here, but the pattern for the remaining warnings (DNS, SSH, ENV and HOSTS) can all be verified by examining the system's configuration and contrasting it to the policy file configuration in Figure 84.

5.4.3 Section with 'Ignore' Keyword

The 'ignore' keyword gives the administrator the flexibility to specify commands or sections that should be ignored when our policy compliance system compares the policy to the device configurations. The 'ignore' keyword is useful for writing a comprehensive device template to be mirrored during the device creation process, while allowing certain sections of the policy to be skipped by the RCM.

To demonstrate the concept of the 'ignore' keyword, let us look at the policy from section 5.4.2, Figure 84. Specifically, let us target the Environment variable section, lines 29–33, shown in Figure 89.

```
29 /*##### ENV (Environment variable) #####  
30 %section-start Exclude ENV  
31     TERM xterm  
32     SHELL /bin/bash  
33 %section-end
```

Figure 89 Environment Variable Section

We also know from the previous section that line 31 has triggered a non-compliance report, as seen in Figure 90.

```
<----- New Policy File detected ----->
#6-NetInt WARNING(netmask): 255.255.255.0 does not match (exclude).
#10-NTP    WARNING(server): 0.rhel.pool.ntp.org Found (exclude).
#14-Hname  WARNING(name): Client Matches (exclude).
#22-DNS    WARNING(nameserver): 8.8.4.4 Found (exclude).
#26-SSH    WARNING(KeyRegenerationInterval): 3600 Found (exclude).
#31-ENV    WARNING(TERM): xterm Matches (exclude).
#32-ENV    WARNING(SHELL): /bin/bash Matches (exclude).
#36-HOSTS  variable WARNING(127.0.1.1): client Found (exclude).
<----->
```

Figure 90 ENV Variable Trigger

One way to include the Terminal type (TERM) configuration in the template but not have it checked by the CRM is by the use of the 'ignore' keyword as illustrated in Figure 91 below:

```

1 %version 3.396
2 %device regex (Client-router[0-10])
3 /*##### NetworkInterface #####
4 %section-start Exclude NetworkInterface eth0
5     address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
6     netmask 255.255.255.0
7 %section-end
8 /*##### NTP #####
9 %section-start Exclude NTP /etc/ntp.conf
10     server 0.rhel.pool.ntp.org
11 %section-end
12 /*##### Hostname #####
13 %section-start Exclude Hostname
14     name Client
15 %section-end
16 /*##### FSSL (File System Size Limit) #####
17 %section-start Exclude FSSL /dev/sda1
18     limit 12
19 %section-end
20 /*##### DNS #####
21 %section-start Exclude DNS /etc/resolv.conf
22     nameserver 8.8.4.4
23 %section-end
24 /*##### SSH #####
25 %section-start Exclude SSH /etc/ssh/sshd_config
26     KeyRegenerationInterval 3600
27     ServerKeyBits 1024
28 %section-end
29 /*##### ENV (Environment variable) #####
30 %section-start Exclude ENV
31     SHELL /bin/bash
32 %section-end
33 %section-start Ignore ENV
34     TERM xterm
35 %section-end
36 /*##### HOSTS #####
37 %section-start Exclude HOSTS /etc/hosts
38     127.0.1.1 client-virtualBox
39 %section-end

```

Figure 91 Using the 'Ignore' Keyword

From the RCM report in Figure 92, we see that the TERM environment variable alarm did not trigger an alarm by the RCM.

```
<----- New Policy File detected ----->
#6-NetInt WARNING(netmask): 255.255.255.0 does not match (exclude).
#10-NTP    WARNING(server): 0.rhel.pool.ntp.org Found (exclude).
#14-Hname  WARNING(name): Client Matches (exclude).
#22-DNS    WARNING(nameserver): 8.8.4.4 Found (exclude).
#26-SSH    WARNING(KeyRegenerationInterval): 3600 Found (exclude).
#31-ENV    WARNING(SHELL): /bin/bash Matches (exclude).
<----->
```

Figure 92 TERM Alarm Cleared

We also see from the system verification output in Figure 93 that the system is configured with the xterm terminal type, which we asked the policy to ignore in our policy file.

```
root@Client:~/Client# env
TERM=xterm
SHELL=/bin/bash
SSH_CLIENT=10.168.255.70 61468 22
SSH_TTY=/dev/pts/0
USER=root
...
<truncated>
```

Figure 93 ENV Configuration

5.4.4 Conditional Block

The conditional block allows the CRM to perform different actions depending on whether the specified parameters within the conditional block, (Boolean condition) evaluate to true or false. If the block has more than one parameter, then all of the parameters are treated as a logical AND, meaning that

all values must match for the condition to evaluate to true. We use the following example in Figure 94 to explain how to construct the conditional block:

```
%version 3.397
%device regex (Client-router[0-10])
%condition-start NItag NetworkInterface eth0
    address 10.168.255.149
    netmask 255.255.255.regex[0-252]
%condition-end
```

Figure 94 Conditional Block

The conditional block starts with the delimiter `%condition-start` followed by a conditional block identifier, and ends with `%condition-end`. Between are the parameters and their values that will set the Boolean condition to true or false, depending on whether the system matches the specified values.

In our Figure 94 example, the condition is set to true when the system's eth0 interface IP is set to 10.168.255.149 AND its netmask is set to anything between 255.255.255.0 and 255.255.255.252. Otherwise, the Boolean condition evaluates to false.

We will demonstrate the use of the conditional block in the next section.

5.4.5 If-Then Block

The conditional block from the previous section is used in the if-then block. When the RCM finds an 'if' that matches the condition from the previous section, whose Boolean value is true, then the policies defined within the if-then block are compared to the device's configuration. Otherwise, they are skipped.

The conditional section is identified by if-start and the terminator is identified by if-end. The negation operator (not) returns the opposite of the given Boolean expression. Figure 95, shows an example of a policy using the if-then block.

```

1 %version 3.398
2 %device regex (Client-router[0-10])
3 /*****Conditional Block*****/
4 %condition-start NItag NetworkInterface eth0
5     address 10.168.255.149
6     netmask 255.255.255.regex[0-252]
7 %condition-end
8 %condition-start Hostn Hostname
9     name Client
10 %condition-end
11 /*****NTP if then*****/
12 %if-start NItag NTP /etc/ntp.conf
13     server time-a.nist.gov
14 %if-end
15 /*****NTP negation operator *****/
16 %if-start Not_NItag NTP /etc/ntp.conf
17     server 10.168.255.1
18 %if-end
19 /*****DNS if then*****/
20 %if-start Hostn DNS /etc/resolv.conf
21     nameserver 8.8.4.4
22 %if-end
23 /*****DNS negation operator *****/
24 %if-start Not_Hostn DNS /etc/resolv.conf
25     nameserver 208.67.222.222
26 %if-end
27

```

Figure 95 If-Then Policy

Line 4 of the policy in Figure 95 sets the first conditional block with a tag name of NItag. The block is true if the IP and the netmask of eth0 that are configured in the policy both matches the system.

Line 5 of the policy in Figure 95 sets the second conditional block, whose tag name is Hostn. The block is true if the system hostname is 'Client.'

Line 13 checks the Boolean value of the NItag conditional block. If true, the RCM would expect the system to have one of its NTP servers pointing to time-a.nist.gov. If false, according to line 17 it would expect one of the NTP servers to point to 10.168.255.1.

Similarly, the RCM expects the nameserver to point to 8.8.4.4 if the Hostn conditional block is true (line 21), and to 208.67.222.222 if false (line 25).

Once the new policy was downloaded to our policy client, the RCM generated the Figure 96 report.

```
<----- New Policy File detected ----->
#13-NTP   WARNING(server): time-a.nist.gov Not found.
<----->
```

Figure 96 If-Then Report

From the report output we can conclude that the NItag was evaluated to true, because the system's IP address and subnet mask both matched the policy (Figure 97). However, since a true evaluation of the expression on line 13 expects

the NTP sever to be set to time-a.nist.gov and the RCM found no matching configuration on the system, it generated the warning.

```
root@Client:~/Client# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e0:7e:a5
          inet addr:10.168.255.149  Bcast:10.168.255.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39047 errors:0 dropped:0 overruns:0 frame:0
          TX packets:743 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5125695 (5.1 MB)  TX bytes:96015 (96.0 KB)
```

Figure 97 Eth0 IP Configuration

Figure 98 below confirms that the system's configuration does not include the time-a.nist.gov NTP server.

```

root@Client:~/Client# more /etc/ntp.conf
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server 0.rhel.pool.ntp.org

```

Figure 98 NTP Configuration

To better understand the DNS if-then blocks on line 21 and line 25 of Figure 95, we need to check the conditional block Hostn. If the block evaluates to true, then the nameserver should be set to 8.8.4.4, and if false to 208.67.222.222.

Figure 99, from the system confirms that the system hostname is set to 'Client.' Therefore the Boolean expression is true, and we would expect the server to have the correct nameserver configuration. We know this because the RCM did not generate a DNS alarm, as seen in Figure 96.


```
root@Client:~/Client# hostname
Client
root@Client:~/Client# more /etc/hostname
Client
root@Client:~/Client#
```

Figure 99 Hostname Configuration

Next, we will change the system hostname to something other than 'Client,' so that the Boolean expression evaluates false, resulting in a DNS warning being generated by the RCM.

In Figure 100, we issue the Linux hostname command to change the system's hostname from 'Client' to 'desktop.'

```
root@Client:~/Client# hostname desktop
root@Client:~/Client# hostname
desktop
```

Figure 100 Hostname Changed to 'desktop'

Immediately after the change, the RCM generated the report in Figure 101, informing us in real-time that the system is no longer in compliance with line 25 of the policy.

```
<----- Machine Report ----->
#17-NTP WARNING(server): 10.168.255.1 Not found. [Caused by system change]
#25-DNS WARNING(nameserver): 208.67.222.222 Not found. [Caused by system change]
<----->
```

Figure 101 DNS Warning

When we check the system's DNS configuration, we clearly see in Figure 102 that server 208.67.222.222 is not part of the configuration, which was expected when the condition `Hostn` became false.

```
root@Client:~/Client# more /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.168.255.1
nameserver 8.8.4.4
root@Client:~/Client#
```

Figure 102 DNS Configuration

5.4.6 Variables

We discussed variables in details in Section 3.4.5 of this dissertation. To summarize, variables are a powerful way to represent device-specific parameters appearing in multiple sections of the policy, with one reference. In other words, instead of repeating the parameter and its value, we have the option to set a unique variable name and reference it instead.

For testing, two different types of variables were implemented: global-single and global-list. Both types of variables are contained between percent signs (%) and start with the keyword 'global-single' or 'global-list,' followed by a unique name.

The policy in Figure 103 illustrates the use of the global-single variable. For example, the `eth0` IP address on line 5 is now a variable with a unique name

called InterfaceIP, and we see on line 13 that the same variable is being referenced for the SSH ListenAddress parameter. Another example in

Figure 103 is the variable MyHostName on line 9, which is being referenced on line 18 to set the system FQDN.

```
1 %version 3.399
2 %device regex (Client-router[0-10])
3 /***** NetworkInterface *****/
4 %section-start Exact NetworkInterface eth0
5     address %global-single InterfaceIP%
6 %section-end
7 /***** Hostname *****/
8 %section-start Exact Hostname
9     name %global-single MyHostName%
10 %section-end
11 /***** SSH *****/
12 %section-start Exact SSH /etc/ssh/sshd_config
13     ListenAddress %InterfaceIP%
14     ServerKeyBits 1024
15 %section-end
16 /***** HOSTS *****/
17 %section-start Exact HOSTS /etc/hosts
18     127.0.0.1 %MyHostName%.acme.com
19 %section-end
```

Figure 103 Global-Single Policy

Once our client received the new policy, the RCM generated the report in Figure 104.

```
<----- New Policy File detected ----->
#13-SSH  WARNING(ListenAddress): 10.168.255.149 Not found.
#14-SSH  WARNING(ServerKeyBits): 1024 Not found.
#18-HOSTS WARNING(127.0.0.1): Client.acme.com Not found.
<----->
```

Figure 104 Global-Single Report

Line 13 is a warning because the ListenAddress address does not match the policy. According to the policy, it should be set to 10.168.255.149. However, upon examining the system, we found that the ListenAddress is set to 0.0.0.0, as seen in Figure 105.

The line 14 warning is not related to the variable configuration, but as we can see in Figure 105, the system's ServerKeyBits is set to 768, while the policy in Figure 103 expects 1024.

```
root@Client:~/Client# more /etc/ssh/sshd_config
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768
..
<truncated>
```

Figure 105 SSH Configuration

Line 18 is a warning because the FQDN Client.acme.com does not exist in the hosts file, as confirmed by the verification command in Figure 106.

```
root@Client:~/Client# more /etc/hosts
127.0.0.1    localhost
127.0.1.1    client
```

Figure 106 Hosts File

The values of both our policy variables, `InterfaceIP` and `MyHostName`, can be changed by the administrator. Our RCM would adjust dynamically and refresh the reporting based on the newly-entered information. Let us demonstrate the concept by changing the interface IP address. We will issue the command shown in Figure 107.

```
root@Client:~/Client# ifconfig eth0 192.168.1.1 netmask 255.255.255.0
```

Figure 107 Eth0 IP Address Change

Notice the report in Figure 108 is now a Machine type, and the RCM immediately adjusted and issued a new warning for Line 13.

```
<----- Machine Report ----->
#13-SSH WARNING(ListenAddress): 192.168.1.1 Not found.
#14-SSH WARNING(ServerKeyBits): 1024 Not found.
#18-HOSTS WARNING(127.0.0.1): Client.acme.com Not found.
<----->
```

Figure 108 New Line 13 Warning

The same results are observed when the hostname is changed from 'Client' to 'Lab.' Figure 109 shows the new RCM report.

```
<----- Machine Report ----->
#13-SSH    WARNING(ListenAddress): 192.168.1.1 Not found.
#14-SSH    WARNING(ServerKeyBits): 1024 Not found.
#18-HOSTS  WARNING(127.0.0.1): lab.acme.com Not found.
<----->
```

Figure 109 New Line 18 Warning

The difference between global-single and global-list variables is that a global-single variable holds only one value, while a global-list can hold multiple variables. For example, the parameter TERM in the system's environment configuration can be assigned one and only one terminal type. This would be an example of a global-single use. An NTP configuration or a DNS configuration, for example, could hold one or more server values. These would be good uses for global-list.

The policy in Figure 110 illustrates the use of global-list variables.

```

1 %version 3.4
2 %device regex (Client-router[0-10])
3 /***** NTP *****/
4 %section-start Exact NTP /etc/ntp.conf
5     server %global-list lst_NTP_serv%
6 %section-end
7 /***** DNS *****/
8 %section-start Exact DNS /etc/resolv.conf
9     nameserver %global-list lst_dns_nams%
10 %section-end
11 /***** HOSTS *****/
12 %section-start Exact HOSTS /etc/hosts
13     127.0.0.1 %global-list lst_HOSTS_127%
14 %section-end
15 /***** PRINT *****/
16 %section-start Exact PRINT
17     %lst_NTP_serv%
18     %lst_dns_nams%
19     %lst_HOSTS_127%
20 %section-end

```

Figure 110 Global-List Policy

The policy in Figure 110 sets NTP, DNS and HOSTS to global-list variables, and the PRINT section is added so that the values of the variables will be printed to the screen for verification. Figure 111 displays the report generated by the RCM.

```
<----- New Policy File detected ----->
#17-%lst_NTP_serv%: 0.ubuntu.pool.ntp.org;
                   1.ubuntu.pool.ntp.org;
                   2.ubuntu.pool.ntp.org;
                   3.ubuntu.pool.ntp.org;
                   0.rhel.pool.ntp.org;ntp.ubuntu.com
#18-%lst_dns_nams%: 10.168.255.1;
                   208.67.220.220;
                   8.8.8.8
#19-%lst_HOSTS_127%: localhost;
                   client
<----->
```

Figure 111 Global-List Report

From the report, we can see that NTP, DNS and HOSTS each have multiple values, and this is easily confirmed by checking the actual configurations on our system. Figure 112, lists all of the configured values for NTP, DNS and HOSTS, and they match the RCM's generated report.


```

root@Client:~/Client# more /etc/ntp.conf
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server 0.rhel.pool.ntp.org

root@Client:~/Client# more /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.168.255.1
nameserver 208.67.220.220
nameserver 8.8.8.8

root@Client:~/Client# more /etc/hosts
127.0.0.1    localhost
127.0.0.1    client

```

Figure 112 System Verification

5.5 Summary

This chapter detailed one of the major contributions of our research, the Runtime Compliance Manager (RCM). It showed how the RCM runs in the policy client, how it continuously monitors the network element configuration state, analyzes its current state and reports discrepancies when applicable. We

also presented the reporting module of the RCM, and how it is called upon to generate the appropriate alarming in real-time.

For a proof of concept we used a Linux system to implement the RCM, the open nature of the Linux operation system made it an attractive choice. However, the RCM module is not limited to a particular system and can be ported to on other systems straight forwardly.

The example scenarios in this chapter targeted network interfaces, NTP, DNS, SSH, and other system variables configuration. Throughout the example scenarios we changed the client configuration and observed the RCM report, in real-time, the discrepancies caused by any noncompliant configuration. For this type of change the RCM generated a “Machine Report” informing the operator that the discrepancies were caused by a machine change.

The example scenarios also targeted the policy file for the client, by making policy changes. In these cases, the RCM generated a “New Policy File” report whenever a mismatch was found.

In conclusion, the chapter demonstrated how the proposed automated policy and compliance system leverages the Policy Exchange and Policy Language frame work from chapter 3 to perform automatic and intelligent network configuration audits in real-time. This provides the system administrators and change initiators with the real-time feedback to prevent outages, disruptions, or vulnerabilities to their networks caused by configuration changes that violate any of the policies set for any given device.

CHAPTER 6: SUMMARY AND CONCLUSIONS

6.1 Conclusion

Today's network operators are confronted with managing multivendor networking environments supporting business-critical data, voice and video in the same network infrastructure. Maintaining the performance and effectiveness of these networks requires administrators to continuously conduct complex configuration changes without committing a single mistake. Unfortunately, change is not always successful, and changing one configuration parameter sometimes brings a complex chain of events that the network administrator cannot anticipate. These events often lead to performance degradation and vulnerability, and can even lead to services outages and downtime that are very costly to businesses. Managing change effectively and reducing the negative effects of day-to-day operations has become one of the most important tasks in IT.

In this dissertation, we tried to address the aforementioned issues. First we provided the network infrastructure devices with a device policy that conforms to organizational and industry best practices and regulatory standards, and is consistent throughout the network, by use of an automated, secured and guaranteed process. Second, we enforced the policy rules to the device

configuration and provided network administrators and change owners with feedback on any inconsistencies in real-time.

The first contribution offered in this dissertation stems from the fact that the use of policy template techniques allows for uniform configuration rules across similar devices with the same networking role. This led to the introduction of the concept of a Common Policy Language (CPL).

The Common Policy Language (CPL) introduced in this dissertation is written in ASCII text, is portable and powerful, and is remarkably easy to manage and manipulate. It is essentially similar to the device configuration structure, but with more powerful programming language functions, statements and operations. The uniformity of the Common Policy Language code makes it easy to understand and implement, and invites further customization, development and improvements. One of the focal points of the research was a systematic methodology for developing and using such a language, including an automated, secured, and reliable system for centralized policy delivery to all clients in the domain, as well as guaranteed maintainability. The system has the intelligence to report a device's role in the network and its policy version number, and to detect whether or not the device has the most current policy.

The second main contribution of the dissertation is the Runtime Compliance Manager (RCM), which is responsible for detecting any configuration violations or inconsistencies between what is defined as acceptable in the policy file, and what is actually configured on the device itself. A crucial

component of this system is the reporting. When a change is made, both the change owner and the network surveillance system are informed in real-time of any policy violations caused by the change could result in network problems.

The case studies presented in this dissertation provided detailed examples on the application of the Automated Policy Compliance and Change Detection System. In the first stage of the case studies we showed different plausible states that a network device could be in, and showed how our the policy delivery system module was able to reliably and securely guarantee the maintainability of its policy file. In the second stage, we changed our policy client configurations to test the RCM module's ability to detect inconsistencies between a device's policy file and its configuration. In each of the test cases, we successfully proved our system's ability to detect and report compliance violations in real-time.

The main conclusion of this research effort is that it is a significant challenge for any network to ensure that system configurations remain compliant with internal and regulatory security and compliance policies. Even the best administrators can make mistakes, and the cost of missing a key configuration or accidentally skipping one could be catastrophic. Our Automated Policy Compliance and Change Detection System can play a major role in helping IT professionals improve configuration management and network stability and reduce errors. This is done by auditing planned and unplanned changes and exposing any potential risks in real-time.

6.2 Limitations and Future Work

In this dissertation, the focus was on developing an Automated Policy Compliance and Change Detection System and using a Common Policy Language to represent organizational, industry, and regulatory policies. Each step of the proposed system represents a research area by itself. While we tried to address some of these areas in some level of detail, limitations still exist and there is more work still to be done.

The Common Policy Language (CPL) in which the device policies are expressed requires more attention. The design of the language concept was centered on two of the largest companies in the networking field, Cisco Systems and Juniper Networks. Although we show in our test case scenarios that the language could easily be expanded to include systems like Linux, the development of more keywords, parameters, functions and logic is still required to build a full set of standards that would be portable across many more vendors.

Another area of future work is expanding the Common Policy Language for formatting as Extensible Markup Language (XML). XML provides a flexible but fully-specified encoding mechanism for hierarchical data presentation. The flexibility of XML has led to its widespread use for exchanging data in a multitude of forms. Many networking vendors offer formatting configuration in XML format, making XML, like our proposed language structure, vendor and platform independent. Furthermore, conversion of XML documents into readable formats can be accomplished using the XML Stylesheet Language

Transformation (XSLT), thus facilitating the ability to recast data in differing lights and convert it to and from common formats [76]. Our system could benefit from having these different formatting options to express the device policies.

The Runtime Compliance Manager (RCM) also requires future enhancements. The module should be made to activate only when the network administrator enters the configuration state of the networking device. This was hard to implement on the Linux system, because there is no real delineation for system configuration, whereas in most networking devices the configuration stage is clearly delineated by entering a specific configuration level. To overcome the limitation, we configured the RCM module to monitor the configuration every 3 seconds. This enhancement lead to better CPU utilization and better overall system performance.

Another area for future work on the RCM is the reporting system. In conjunction with developing more parameters and keywords in the CPL, the RCM reporting could be enhanced to display different categories of alarm depending on the violation type. For example, the categories may include severe, high, low, optional, etc. This is important to the change owner because it portrays the severity impact of the change, and equally important to the surveillance systems because it gives them the option to filter incoming alarms and act in accordance with the alarm's severity.

Finally, although our proposed system has a great reporting mechanism, it could be improved by research on the possibility of integrating the reporting into

operation support systems (OSS), and specifically into the event correlation module. Event correlation can further reduce the risk of misconfiguration and outage minutes by simplifying and speeding up the monitoring of network events. This can be accomplished by consolidating the event messages, alarms, alerts and error logs into a root cause determination capable of detecting the origins of problems and generating real-time recommendations for finding their locations. This would be very helpful from a surveillance point of view, because instead of requiring network operators to process hundreds of alarms related to one event, the system would make a fast correlation and allow a quick recovery from any network anomaly.

APPENDIX A – CLIENT/SERVER POLICY EXCHANGE AND MANAGEMENT SOURCE CODE

The source files referenced include the following:

Buffer.cpp and **Buffer.h**: The class and subclasses are related to the operations of the buffer used to send information between server and client. Within these classes are specified all the details of the methods and data members for the creation and modification of the message to send/receive.

We have the base class buffer, which contains the basic tools for a network buffer. We use this base class afterwards in order to inherit a new class called CommonHeader, which contains the fundamentals of the protocol in the present document, especially for sending purposes. In addition, we have a couple of stand-alone classes: Object and ParsePolicy. Object class is used to add objects just as it is done in the protocol COPS-PR, whereas ParsePolicy is mainly used to parse the buffer received.

ClientMain.cpp: This is the main of the client code. This file is where the program begins for the client. It directly imports code from PolicyClient.cpp. So the logic and complexity of this file are very low.

debugTools.cpp and **debugTools.h**: These contain functions which do not contribute to the behavior of the program. Instead they exist just for debugging purposes.

For now they have only one function, `PrintRawBuffer`, to print all the bytes contained in the buffer. This is important since the standard C++ library for printing on the screen does not do a great job when there are null characters along the string.

FileOperation.cpp and **FileOperation.h**: These perform file operation functions. All the file operations such as sending, receiving, opening, closing, and getting the version are written in these files. There is only one class, and it handles all the necessary behaviors related to file operation for this project. There are also some independent functions for sending, receiving, writing and regular expression, which are used specifically for files.

hashFunction.cpp and **hashFunction.h**: These contain function interfaces and implementation for the hash operations. With these stand-alone functions is possible to generate the hash code for a file or for a network buffer.

md5.cpp and **md5.h**: These files are for the MD5 operations. These files are used to compute the md5 algorithm from a network buffer only. The hash function is available in its entirety at [77].

PolicyClient.cpp and **PolicyClient.h**: These contain the class for the client's behavior. This class makes it easier for any user to implement a client. It can be instantiated and then called by the run function, which completes the job. This class is where all the behavior of the client is coded.

PolicyServer.cpp and **PolicyServer.h**: These contain the class for the server's behavior. They are analogous to the PolicyClient files, but in this case

they are for the server. PolicyServer has a similar structure to PolicyClient. The main difference is that with PolicyServer, it is possible to create multiple threads - as many threads as there are clients, subject to the limitations of the machine.

Socket.cpp and **Socket.h**: These contain the basic socket operations. The focus of these files is to deal with the complexities of the creation of sockets on the standard Unix environment. Although there are a lot of free and open source solutions that perform the same function, we opted to create new ones in order to avoid the extra installation of libraries. In this way the system is easier to install, since we avoided one requisite.

PolicySocketOp.cpp and **PolicySocketOp.h**: Functions for the socket communication for this particular application. The key difference between these classes and the sockets ones is that these are created specifically for the details of this network protocol between server and client. They serve two functions, by which they read the messages in an organized way and there by guarantee that the correct message is being read.

ServerMain.cpp: This is the main program for the server and is analogous to the ClientMain. This file is where the server starts its execution. After booting, this is the first file to be called, followed by the PolicyServer files.

APPENDIX B – RUNTIME COMPLIANCE MANAGER CODE

BlockFactory.h and BlockFactory.cpp: These files handle all the details related to each block in the policy file. They also provide the base class BlockFactory, which is the main class used as an interface, as well to improve code-enhancement in the future.

For future improvements, it is recommended to divide these files if the code becomes too large. This is only for the purpose of readability.

CmdLineUtilities.h and CmdLineUtilities.cpp: With these command-line-utility files, we have those extra tools to handle all operations related to Linux commands. To put it briefly, they hold a group of independent functions that send requests to the system asking for specific information. Operations related to filesystems, hostname, environment variables and so on, are managed by these files.

NetUtilities.h and NetUtilities.cpp: The network-utility files are written for operations related to networks. These are useful for finding features on the system, checking to see if a network pattern fits a regular expression, and so on. They are all related in some way to the network environment. In these files, we can see operations related to finding the IP of the machine, finding the netmask, finding a parameter in a network configuration file, and so on.

PEmain.cpp: This is the file which contains the main function. This is where the entire algorithm begins. One of the first things this file does is call the PolicyEnf files to carry out the appropriate behavior. This file was left with few responsibilities, which makes it easier to implement this code. It only takes an initialization and a call of the run function.

PolicyEnf.h and PolicyEnf.cpp: The policy-enforcement files are in charge of carrying out the general behavior of the code. They are coded with an infinite loop, which creates a program that will continually monitor the system until stopped by an external action.

StrUtility.h and StrUtility.cpp: These files are written to be used as a tool to work with string, and are especially focused on our particular software. This is because we have to deal with several kinds of character strings in the policy file.

Operations such as analyzing a machine report, adding the source of a modification and concatenating variables are carried out by these files.

Variables.h and Variables.cpp: The variables files are created to handle all the global-single and global-list variables in an easy way. Inside this file there are a couple of classes which work together to create an efficient way to work with variables. We have the Variable class, which defines a single variable in a policy file, and we have the VariableManager class, which defines a vector of Variable classes to store as many variables as the system can assign.

APPENDIX C – POLICY FILE EXAMPLE

```
%version 3.394
%device regex (Client-router[0-10])

/*#####
#####NetworkInterface#####
#####*\

%section-start Exact NetworkInterface eth0
    address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.255.0
    address %global-single sin_NetworkInterface_add%
    netmask %global-single sin_NetworkInterface_net%
%section-end

%section-start Exclude NetworkInterface eth0
    address regex[200-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.254.0
    address %sin_NetworkInterface_add%
    netmask %sin_NetworkInterface_net%
%section-end

%section-start Ignore NetworkInterface eth0
    address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.255.0
    address %sin_NetworkInterface_add%
    netmask %sin_NetworkInterface_net%
%section-end

%condition-start NItag NetworkInterface eth0
    address regex[200-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.255.0
    address %sin_NetworkInterface_add%
    netmask %sin_NetworkInterface_net%
%condition-end

%if-start NItag NetworkInterface eth0
    address regex[200-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.255.1
    address %sin_NetworkInterface_add%
    netmask %sin_NetworkInterface_net%
%if-end

%if-start Not_NItag NetworkInterface eth0
    address regex[100-255].regex[100-185].regex[1-155].regex[100-155]
    netmask 255.255.255.0
    address %sin_NetworkInterface_add%
    netmask %sin_NetworkInterface_net%
%if-end
```

```

/*#####
##### NTP #####
#####*\

%section-start Exact NTP /etc/ntp.conf
    server 0.0.0.1
    server 0.rhel.pool.ntp.org
    server 1.rhel.pool.ntp.org
    server 2.rhel.pool.ntp.org
    server %global-list lst_NTP_serv%
    server %global-single sin_NTP_serv%
%section-end

%section-start Exclude NTP /etc/ntp.conf
    server 1.1.1.0
    server 0.rhel.pool.ntp.org
    server 1.rhel.pool.ntp.org
    server 2.rhel.pool.ntp.org
    server %lst_NTP_serv%
    server %sin_NTP_serv%
%section-end

%section-start Ignore NTP /etc/ntp.conf
    server 1.1.1.0
    server 0.rhel.pool.ntp.org
    server 1.rhel.pool.ntp.org
    server 2.rhel.pool.ntp.org
    server %lst_NTP_serv%
    server %sin_NTP_serv%
%section-end

%condition-start NTPservers NTP /etc/ntp.conf
    server 0.0.0.1
    server 0.rhel.pool.ntp.org
    server 1.rhel.pool.ntp.org
    server 2.rhel.pool.ntp.org
    server %lst_NTP_serv%
    server %sin_NTP_serv%
%condition-end

%if-start NTPservers NTP /etc/ntp.conf
    server 0.0.0.1
    server %lst_NTP_serv%
    server %sin_NTP_serv%
%if-end

%if-start Not_NTPservers NTP /etc/ntp.conf
    server 192.168.1.254
    server %lst_NTP_serv%
    server %sin_NTP_serv%
%if-end

/*#####
##### HostName #####
#####*\

%section-start Exact Hostname
    name saeed-VirtualBox
    name %global-single sin_Hostname_nam%
%section-end

```

```

%section-start Exclude Hostname
    name Machine
    name %sin_Hostname_nam%
%section-end

%section-start Ignore Hostname
    name Client-router
    name %sin_Hostname_nam%
%section-end

%condition-start Hostn Hostname
    name saeed-VirtualBox
    name %sin_Hostname_nam%
%condition-end

%if-start Hostn Hostname
    name saeed-VirtualBox
    name %sin_Hostname_nam%
%if-end

%if-start Not_Hostn Hostname
    name NotClient
    name %sin_Hostname_nam%
%if-end

/*#####
##### File Size System Limit (FSSL) #####
#####*\

%section-start Exact FSSL /dev/sda1
    limit 42
    limit %global-single sin_FSSL_lim%
%section-end

%section-start Exclude FSSL /dev/sda1
    limit 20
    limit %sin_FSSL_lim%
%section-end

%section-start Ignore FSSL /dev/sda1
    limit 15
    limit %sin_FSSL_lim%
%section-end

%condition-start filesize FSSL /dev/sda1
    limit 50
    limit %sin_FSSL_lim%
%condition-end

%if-start filesize FSSL /dev/sda1
    limit 80
    limit %sin_FSSL_lim%
%if-end

%if-start Not_filesize FSSL /dev/sda1
    limit 40
    limit %sin_FSSL_lim%
%if-end

```



```

/*#####
##### DNS #####
#####*\

%section-start Exact DNS /etc/resolv.conf
    nameserver 127.0.0.1
    nameserver %global-single sin_dns_nams%
    nameserver %global-list lst_dns_nams%
%section-end

%section-start Exclude DNS /etc/resolv.conf
    nameserver 255.255.255.0
    nameserver %sin_dns_nams%
    nameserver %lst_dns_nams%
%section-end

%section-start Ignore DNS /etc/resolv.conf
    nameserver 255.255.255.0
    nameserver %sin_dns_nams%
    nameserver %lst_dns_nams%
%section-end

%condition-start DNStag DNS /etc/resolv.conf
    nameserver 127.0.0.1
    nameserver %sin_dns_nams%
    nameserver %lst_dns_nams%
%condition-end

%if-start DNStag DNS /etc/resolv.conf
    nameserver 127.0.0.1
    nameserver %sin_dns_nams%
    nameserver %lst_dns_nams%
%if-end

%if-start Not_DNStag DNS /etc/resolv.conf
    nameserver 255.255.255.1
    nameserver %sin_dns_nams%
    nameserver %lst_dns_nams%
%if-end

/*#####
##### SSH #####
#####*\

%section-start Exact SSH /etc/ssh/ssh_config
    GSSAPIAuthentication yes
    Host %global-single sin_SSH_ServKeyB%
%section-end

%section-start Exclude SSH /etc/ssh/ssh_config
    KeyRegenerationInterval 255.255.255.0
    HashKnownHosts no
    ServerKeyBits %sin_SSH_ServKeyB%
%section-end

%section-start Ignore SSH /etc/ssh/ssh_config
    KeyRegenerationInterval 255.255.255.0
    ServerKeyBits %sin_SSH_ServKeyB%
%section-end

```

```

%condition-start SSHtag SSH /etc/ssh/ssh_config
    HashKnownHosts yes
    Host %sin_SSH_ServKeyB%
%condition-end

%if-start SSHtag SSH /etc/ssh/ssh_config
    HashKnownHosts yes
    Host %sin_SSH_ServKeyB%
%if-end

%if-start Not_SSHtag SSH /etc/ssh/ssh_config
    KeyRegenerationInterval 255.255.255.0
    ServerKeyBits %sin_SSH_ServKeyB%
%if-end

/*#####
##### Environment Variables (ENV) #####
#####*\

%section-start Exact ENV
    TERM vt100
    LANG en_US.UTF-8
    SSH_TTY /dev/pts/2
    SSH_TTY %global-single sin_ENV_sshTTY%
    LOGNAME saeed
    MAIL /var/mail/saeed
    PATH %global-single sin_ENV_path%
%section-end

%section-start Exclude ENV
    TERM 255.255.255.0
    LANG asf
%section-end

%section-start Ignore ENV
    TERM 255.255.255.0
    LANG asf
    PATH %sin_ENV_path%
%section-end

%condition-start NITag ENV
    TERM vt100
    LANG en_US.UTF-8
    PATH %sin_ENV_path%
%condition-end

%if-start ENVtag ENV
    TERM vt100
    LANG en_US.UTF-8
    PATH %sin_ENV_path%
%if-end

%if-start Not_ENVtag ENV
    TERM 255.255.255.0
    LANG asf
    PATH %sin_ENV_path%
%if-end

/*#####
##### HOSTS #####

```

```

#####*\

%section-start Exact HOSTS /etc/hosts
    127.0.0.1 localhost
    127.0.1.1 saeed-VirtualBox
    127.0.0.1 %global-single sin_HOSTS_127%
    127.0.0.1 %global-list lst_HOSTS_127%
%section-end

%section-start Exclude HOSTS /etc/hosts
    127.0.0.1 %my_Hostname%.acme.com
%section-end

%section-start Ignore HOSTS /etc/hosts
    127.0.0.1 %my_Hostname%.acme.com
    127.0.0.1 %sin_HOSTS_127%
    127.0.0.1 %lst_HOSTS_127%
%section-end

%condition-start NITag HOSTS /etc/hosts
    127.0.0.1 localhost
    127.0.0.1 %sin_HOSTS_127%
    127.0.0.1 %lst_HOSTS_127%
%condition-end

%if-start NITag HOSTS /etc/hosts
    192.168.1.1 %sin_Hostname_nam%.software.com
    127.0.0.1 %sin_HOSTS_127%
    127.0.0.1 %lst_HOSTS_127%
%if-end

%if-start Not_NITag HOSTS /etc/hosts
    192.168.1.1 %sin_Hostname_nam%.software.com
    127.0.0.1 %sin_HOSTS_127%
    127.0.0.1 %lst_HOSTS_127%
%if-end

/*#####
#### Print variables (Debug purpose) #####
#####*\

%section-start Exact PRINT
    %sin_NetworkInterface_add%
    %sin_NetworkInterface_net%
    %lst_NTP_serv%
    %sin_NTP_serv%
    %sin_Hostname_nam%
    %sin_FSSL_lim%
    %sin_dns_nams%
    %lst_dns_nams%
    %sin_SSH_ServKeyB%
    %sin_ENV_path%
    %sin_HOSTS_127%
    %lst_HOSTS_127%
    %sin_ENV_sshtty%
%section-end

```

REFERENCES

- [1] Elbadawi, K.; J. Yu. "Improving Network Services Configuration Management" *Computer Communications and Networks (ICCCN)*, pp. 1-6, 2011.
- [2] Opeenheimer, D, A. Ganapathi, A. Petterson. "Why Do Internet Services Fail and What Can Be Done about It?" in *USITS'03: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, Mar. 2003.
- [3] Kerravala, Z. "As the Value of Enterprise Networks Escalates, So Does the Need for Configuration Management," *The Yankee Group*, Jan. 2004.
- [4] Subramanian, M., T. Gonsalves T., N. Usha. (2010). *Network Management: Principles and Practice*. Pearson Education India. pp. 63-64.
- [5] Shields, G. (2010). *The Shortcut Guide to Network Management for the Mid-Market*. Realtime Publishers.
- [6] Claise, B. Wolter R. (2007). *Network Management: Accounting and Performance Strategies*. Cisco Press.
- [7] Telecommunication Management Network [Online]. Available <http://yonk1991.xtgem.com/post/Kuliah/semester5/ManJar/TMN.html>
- [8] Raman, L. "OSI Systems and Network Management" *IEEE Communication Mag.*, vol. 36, no. 3, pp. 46-53, March 1998.
- [9] Clemm, A. (2007). *Network Management Fundamentals*. Cisco Press.
- [10] Boutaba, R.; A. Polyraakis. "Projecting FCAPS to Active Networks" *IEEE*, pp 97-104, August 2002.
- [11] Strassner, J. (2004). *Policy-Based Network Management, Solutions for the Next Generation*. Morgan Kaufmann Publishers.

- [12] Ding, J. (2010). *Advances in Network Management*. Auerbach Publications.
- [13] Mauro, D., K. Schmidt. (2005). *Essential SNMP*, 2nd Edition. O'Reilly Publishing. Pp 30-34
- [14] Leon-Garcia, A.; I. Widjaja. (2003). *Communication Networks*, 2nd Edition. McGraw-Hill Higher Education. pp 839-840.
- [15] Schoenwaelder J. (2003). Overview of the 2002 IAB Network Management Workshop RFC3535. Available <http://www.ietf.org/rfc/rfc3535.txt>
- [16] Oritz, S. (2005). "SNMP Replacements: Is The Venerable Network Management Protocol On Its Last Legs?" Tech & Trends, April 8, 2005 Vol.27 Issue 14, pg 27. Available <http://www.processor.com/editorial/article.asp?article=articles%2Fp2714%2F32p14%2F32p14.asp>
- [17] Reboucas, R.; J. Sauve, A. Moura,; C. Bartolini, D. Trastour. "A Decision Support Tool to Optimize Scheduling of IT Changes" IFIP/IEEE International Symposium on Network Management, pp.343-352, May, 2007.
- [18] Change Management: Best Practices [Online], Available: http://www.cisco.com/en/US/technologies/collateral/tk869/tk769/white_paper_c11-458050.html
- [19] Risks and Dangers of Change Management [online]. Available: <http://www.buzzle.com/articles/risks-and-dangers-of-change-management.html>
- [20] Harris, K. (2000). *IT Organization; Building a Worldclass Infrastructure*, Prentice Hall, pp. 110-112.
- [21] "Change Control vs. Change Management: Moving Beyond IT" [online]. Available: http://www.itsmwatch.com/itil/article.php/11700_3367151_4/Change-Control-vs-Change-Management-Moving-Beyond-IT.htm
- [22] Kim, G., J. Kim, J. Na. "Design and Implementation of Policy Decision Point in Policy-Based Network Computer and Information Science," 2005. Fourth Annual ACIS International Conference. April 2006.

- [23] Kowtha, S.; J. Xi. "An N-State Driven Policy-Based Network Management to Control End-End Network Behaviors" Seventh IEEE International Workshop on Policies for Distributed Systems and Networks, 2006. Pp 75-80, June 2006.
- [24] Berto-Monleon, R.; E. Casini. "Specification of a Policy Based Network Management Architecture". Military Communications Conference. 2011 - MILCOM 2011 , pp. 1393 – 1398, 2011.
- [25] "What You Should Know before Investing in Policy-Based Network Management" [online]. Available: <http://sysdoc.doors.ch/AVAYA/AvayaWhitePaper.pdf>
- [26] Poylisher, A., R, Chadha. "PBNM Technology Evaluation: Practical Criteria" IEEE Workshop on Policies for Distributed Systems and Networks. pp 105-108, 2008.
- [27] Durham, D., J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol," RFC 2748 (Proposed Standard), IETF, Jan. 2000.
- [28] Sahita, R., S. Hahn, K. Chan, K. McCloghrie, "Framework Policy Information Base," RFC 3318 (Informational), IETF, Mar. 2003.
- [29] Chan, K., J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith. "COPS Usage for Policy Provisioning (COPS-PR)," RFC 3084 (Proposed Standard), IETF, Mar. 2001.
- [30] Franco, T.F.; W. Q. Lima, G. Silvestrin, R. C. Pereira, M. J. B. Almeida, L. M. R. Tarouco, L. G. Granville, A. Beller, E. Jamhour, M. Fonseca. "Substituting COPS-PR: An Evaluation of NETCONF and SOAP for Policy Provisioning" Seventh IEEE International Workshop on Policies for Distributed Systems and Networks 2006. pp 10, 204
- [31] W3C, "SOAP Version 1.2," 2007,[Online], available <http://www.w3.org/TR/soap12/>
- [32] Enns, R. "NETCONF Configuration Protocol" [Online]. Available <http://tools.ietf.org/html/rfc4741>, IETF, Dec. 2006.
- [33] Kosiur, D. (2001). *Understanding Policy-Based Networking* Wiley Computer Publishing.

- [34] Policy-Based Management. [Online], available <http://www.linktionary.com/p/policy.html>
- [35] Ok, Ki-Sang; D. W. Hong, Byung-Soo Chang. "The Design of Service Management System Based on Policy-Based Network Management" International Conference on Networking and Services, 2006. pp. 59-64, September 2006.
- [36] Felix, J. G. Clemente, et al. "An XML-Seamless Policy Based Management Framework," in Third International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2005, Sep. 2005, pp. 418–423.
- [37] NETCONF & YANG: Architecture. [Online], available <http://ellisonsoftware.com/2010/05/17/netconf-yang-architecture/>
- [38] Ha Manh, T., T. Iyad, S. Jürgen. (2009). "NETCONF Interoperability Testing." ACM Digital Library: Lecture Notes in Computer Science, 2009, Volume 5637/2009, 83-94.
- [39] Wegner, E. (2011). Protocol Flexibility: Why IT's Essential to an NMS. [Online], available <http://www.billingworld.com/blogs/network-management/2011/02/protocol-flexibility-why-it-s-essential-to-a-nms.aspx>
- [40] IT Sentinel, Network Auditing. [Online], available http://www.opnet.com/solutions/network_performance/it_sentinel.html
- [41] Configuration Audit. [Online], available <http://www.ecora.com/Ecora/Solutions/ConfigurationAudit.php>
- [42] Drogseth, D. (2009). Network Change and Configuration Management: Optimize Reliability, Minimize Risk and Reduce Costs. An Enterprise Management Associates (EMA™)
- [43] Westerinen, A., J. Schnizlein, John Strassner, et al. "Terminology for Policy-Based Management" RFC3198, Nov. 2001.
- [44] Strassner, J., et al. "Policy Core Lightweight Directory Access Protocol (LDAP) Schema" RFC3703, Feb. 2004
- [45] Edgard, J., et al. "Substituting COPS-PR: An Evaluation of NETCONF and SOAP for Policy Provisioning" *Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks* 2006.

- [46] Harrison, R. "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms" IETF RFC 4513, June 2006.
- [47] Melnikov, A., et al. "Simple Authentication and Security Layer (SASL)" IETF RFC 4422, June 2006.
- [48] Sermersheim, J. "Lightweight Directory Access Protocol (LDAP): The Protocol" IETF RFC 4511. June 2006.
- [49] Gregg, M., G. Mays, C. Ries, R. Bandes. (2006). *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*. Syngress Publishing, pp. 124-125
- [50] Kwok, F. (2005) *Network Security Technologies*. Auerbach Publications.
- [51] Eastlake, 3rd, D., et al. "Randomness Recommendations for Security", IETF RFC 4086.
- [52] Sanchez, L., K. McCloghrie, J. Saperia. "Requirements for Configuration Management of IP-based Networks," RFC 3139, June 2001.
- [53] Designing Device Configurations [Online], Available: http://www.cisco.com/c/en/us/td/docs/net_mgmt/prime/infrastructure/1-2/user/guide/prime_infra_ug/create_temps.html
- [54] Tahir, M.,M. Ghattas. (2009). *Cisco IOS XR*. Cisco Press. pp. 110-119
- [55] Infonetics (2013). "Enterprise Networking and Communication Vendor Leadership Scorecard [Online]", Available: <http://www.infonetics.com/pr/2013/Enterprise-Vendor-Scorecard-Highlights.asp>
- [56] Yuxiang, H., L. Shengli, S. Xiaoyan., "A Dynamic Analysis System for Cisco IOS Based on Virtualization" Third International Conference on Multimedia Information Networking and Security (2011)
- [57] JunOS [Online], available from <http://www.juniper.net/us/en/products-services/nos/junos/#overview>, retrieved 10 September 2014.
- [58] Hucaby, D., S. McQuerry. (2010). *Cisco Router Configuration Handbook*. Cisco Press.

- [59] "Linux", in Wikipedia: The Free Encyclopedia; [encyclopedia on-line]; available from <http://en.wikipedia.org/wiki/Linux>; Internet; retrieved 6 November 2014.
- [60] "10 Reasons Open Source Is Good for Business" Noyes, K. (2014). PCWorld Magazine. [PCWorld on-line]; available from http://www.pcworld.com/article/209891/10_reasons_open_source_is_good_for_business.html; retrieved 6 November 2014.
- [61] "UNIX" [Online]: available from <http://en.wikipedia.org/wiki/Unix>; retrieved 18 November 2014.
- [62] "G++" [Online]: available from http://www.vietspring.org/cpp_linux/gpp.html; retrieved 18 November 2014.
- [63] "Bash" [Online]: available from [http://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](http://en.wikipedia.org/wiki/Bash_(Unix_shell)); retrieved 18 November 2014.
- [64] "Make" [Online]: available from [http://en.wikipedia.org/wiki/Make_\(software\)](http://en.wikipedia.org/wiki/Make_(software)); retrieved 18 November 2014.
- [65] "Tar" [Online]: available from <http://www.tecmint.com/18-tar-command-examples-in-linux/>; retrieved 18 November 2014.
- [66] "Md5sum" [Online]: available from <http://en.wikipedia.org/wiki/Md5sum>; retrieved 18 November 2014.
- [67] Kozierok, C. (2005). *The TCP/IP Guide*. Charles M. Kozierok. pp. 64
- [68] "Understanding Syslog: Servers, Messages & Security" [Online], Available from <http://www.networkmanagementsoftware.com/what-is-syslog>; retrieved 20 November 2014.
- [69] "What is stdin? What is stdout? What is stderr?" [Online], available from <http://www.zenez.com/tmp/ou8faqz/cache/18.html>; retrieved 20 November 2014.
- [70] Frisch, E. (2002). *Essential System Administration*, 3rd Edition. O'Reilly. pp. 469.

- [71] “Setting Your System’s Hostname” [Online], available from <https://www.movealong.org/hostname.html>; retrieved 19 November 2014.
- [72] Maxwell, S. (2002). *UNIX System Administration*. The McGraw-Hill Companies. pp. 297.
- [73] “OpenSSH” [Online], available from <http://www.openssh.com/>; retrieved 19 November 2014.
- [74] “Environment Variables” [Online}, available from https://wiki.archlinux.org/index.php/environment_variables; retrieved 20 November 2014.
- [75] Helmke, M. (2014). *Ubuntu Unleashed*, 2014 Edition. Pearson Education Inc. pp. 217-218
- [76] Martin-Flatin, J. “Web-Based Management of IP Networks and Systems,” Ph.D. dissertation, Swiss Federal Institute of Technology, Lausanne (EPFL), Oct 2000.
- [77] “C++ md5 function” [Online], Available from <http://www.zedwood.com/article/cpp-md5-function>; retrieved 12 September 2014.

CURRICULUM VITAE

Saeed M. Agbariah is a senior network engineer with more than 15 years of experience in the networking field. He holds leading industry certifications including Cisco Certified Internetwork Expert (CCIE) - Routing and Switching #10791, Cisco Certified Internetwork Expert (CCIE) - Service Provider #10791, Juniper Networks Certified Internet Expert (JNCIE) - Service Provider #374, and Juniper Networks Certified Internet Expert (JNCIE) - Enterprise Routing Switching #437.

Mr. Agbariah is an Adjunct Professor at George Mason University, where he teaches in the Master of Science in Telecommunications degree program in the Department of Electrical and Computer Engineering. He created a Multiprotocol Label Switching laboratory class and teaches three different courses in the Telecommunications program.

He received his Bachelor of Science in Information Technology from the University of Phoenix and his Master of Science in Telecommunication from George Mason University.