AN INTEGRATED FIREWALL POLICY VALIDATION TOOL

A thesis submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the degree of Master of Science
in the Faculty of Engineering, Computing & Creative Industries

September 2009

By
Richard. J. Macfarlane
School of Computing

Authorship Declaration

I, Rich Macfarlane, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date: 07/09/2009

Matriculation no: 07017081

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Acknowledgements

My thanks goes to Professor Bill Buchanan, for acting as my supervisor for this work, and for all the help and enthusiasm he provided.

Additional thanks goes to Robert Ludwiniak for acting as my internal supervisor for this thesis, and to Lionel Saliou for his invaluable help in the earlier stages of the work.

Finally, thanks goes to my family who have been a constant support throughout this course of work.

Copyright

This research has been conducted under the direction of:
Prof. William Buchanan (PhD)
as of Director of Studies,
and,
Robert Ludwiniak (MSc)
as Second Supervisor.

Contents

List of Tables

List of Figures

Abstract

Security policies are increasingly being implemented by organisations. Policies are mapped to device configurations to enforce the policies. This is typically performed manually by network administrators. The development and management of these enforcement policies is a difficult and error prone task.

This thesis describes the development and evaluation of an off-line firewall policy parser and validation tool. This provides the system administrator with a textual interface and the vendor specific low level languages they trust and are familiar with, but the support of an off-line compiler tool. The tool was created using the Microsoft C#.NET language, and the Microsoft Visual Studio **I**ntegrated **D**evelopment **E**nvironment (IDE). This provided an object environment to create a flexible and extensible system, as well as simple Web and Windows prototyping facilities to create GUI front-end applications for testing and evaluation. A CLI was provided with the tool, for more experienced users, but it was also designed to be easily integrated into GUI based applications for non-expert users. The evaluation of the system was performed from a custom built GUI application, which can create test firewall rule sets containing synthetic rules, to supply a variety of experimental conditions, as well as record various performance metrics.

The validation tool was created, based around a pragmatic outlook, with regard to the needs of the network administrator. The modularity of the design was important, due to the fast changing nature of the network device languages being processed. An object oriented approach was taken, for maximum changeability and extensibility, and a flexible tool was developed, due to the possible needs of different types users. System administrators desire, low level, CLI-based tools that they can trust, and use easily from scripting languages. Inexperienced users may prefer a more abstract, high level, GUI or Wizard that has an easier to learn process.

Built around these ideas, the tool was implemented, and proved to be a usable, and complimentary addition to the many network policy-based systems currently available. The tool has a flexible design and contains comprehensive functionality. As opposed to some of the other tools which perform across multiple vendor languages, but do not implement a deep range of options for any of the languages. It compliments existing systems, such as policy compliance tools, and abstract policy analysis systems. Its validation algorithms were evaluated for both completeness, and performance. The tool was found to correctly process large firewall policies in just a few seconds.

A framework for a policy-based management system, with which the tool would integrate, is also proposed. This is based around a vendor independent XML-based repository of device configurations, which could be used to bring together existing policy management and analysis systems.

Chapter 1

Introduction

1.1 Context

Network security is becoming increasingly complex, and to cope with this security systems are becoming more policy-based. More organisations than every are spending more money on security, and in particular creating security policies to base organisational security around. The UK's Department for **B**usiness **E**nterprise & **R**egulatory **R**eform (BERR) Information Security Breaches Survey for 2008 [6], reports that in the last 6 years, triple is being spent on IT Security, and double the number of companies now having written security policies, as compared to 2002.

Network security policies are complex and have proven to be difficult and costly to implement, test and audit [7, 8]. Where policies don't exist, the task of reverse engineering policies, from existing configurations, is also non trivial. Assistance can be given to administrators when dealing with enforcement of security policies and in validating, and understanding existing polices, in the form of high level modeling and analysis languages and tools, as well as tool support for low level device configuration. However, there are many problems with the current approaches including, untrusted abstract GUI based tools, vendor specific solutions, and incomplete and non pragmatic solutions [9].

The field of firewall traffic filtering policies has been a particular focus for research, due to the widespread use of firewalls and their particularly problematic configuration. Firewall policy rule sets are ever expanding, due to the increasing number and types of network services being used in modern networks, and are therefor becoming increasingly complex to understand and maintain. The creation, management and auditing of these policies is extremely important as they enforce the overall security policy [10, 11]. As stated by Rubin et al. [12] in the following quote:

> "Configuration is a crucial task, probably the most important factor in the security a firewall provides" – Rubin et al. [12]

The task of configuring the device policies is crucial, but it is seldom carried out correctly. Firewall configurations, are often found not to enforce the goals of the

network security policy they enforce [10]. Wool [10] found, in an extensive survey of firewall configurations, that administrators found it extremely difficult to correctly implement complex policies:

> "Complex rule sets are apparently too difficult for administrators to manage effectively" – Wool [10]

Creating, testing, optimisation, and management of these firewall systems is a non-trivial task, but is still mainly carried out manually by network administrators. There is very little support, such as automation, for the administrator in this job [13]. Some vendor specific tools exist but they tend to be GUI driven, and mainly aimed at non-expert users who are not familiar with the CLI. They attempt to simplify deployment of policies, with techniques such as wizards, and automatically generated template-based configurations [14].

The thesis proposes a framework, to support administrators when manageing these complex policies. The framework draws from research into the fields of network firewall policy management [15], and usability in systems for system administrators [9].

1.2 Aims and Objectives

The aim of this thesis is to produce a tool which can parse and validate firewall rule sets. To implement and evaluate the tool with realistic test data. Objectives to support the overall aim of the thesis are as follows:

1. To investigate and review the extensive literature in the field of policy-based security and in particular firewall policy management.
2. Design a solution to the non-trivial task of off-line firewall policy validation, based around firewall device configurations.
3. Implement and test the system, using appropriate tools to realise the design sepecifications.
4. Evaluate the performance of the prototype system, validating its performance using experiments with realistic data sets.
5. Investigate and propose a framework, which the policy validation tool could integrate with, to support system administrators in the management of firewall policies.

1.3 Background

1.3.1 Security Policies

A Security Policy is a document, or group of documents, which define an organisation's security stance. The security goals of the organisation, the assets to be secured, the methods of securing those assets, and a plan for responses to any assets being compromised [16, 11]. A high level information security policy should set the goals of what the security implementation should accomplish. These goals should always be based around the protection of the three main aspects of security: confidentiality, integrity, and availability [17]. This relates to the protection of assets and data against unauthorised disclosure, unauthorised modification, and to provide access to authorised individuals when needed. These high level policies are enforced by security procedures and devices, such as authentication systems and firewalls.

1.3.2 Firewalls

Firewalls are software or hardware which controls the data flowing between two different networks [18, 19, 20, 21, 22]. They are typically deployed on the perimeter of an organisation's network, where it connects to external networks such as the Internet. Firewalls can also be placed between internal networks to enforce different security postures of different network segments, and give a layered approach to network security. For example, internal firewalls might be used to protect critical network segments like server farms. Firewalls can also be deployed on mobile users systems, outside the organisation's network, such as staff working from home.

In buildings a firewall is a concrete wall which prevents fires from spreading from one part of the building to another. In vehicles, a firewall is barrier between the engine and the passenger compartment. In network security, a firewall is a barrier between two networks, used to separate them at a trust boundary. The firewall should only let specified 'good traffic' between the networks.

A good analogy is a medieval castle and the guard on its gate. The walls keep certain people in, and certain people out. The gate is the only place to move between the outside and the inside of the castle, and the guard vets the people passing through the gate and decides if they can enter or leave the castle.

A firewall should be the only transit point between the two networks. It's a choke point which all traffic must pass through. A Network firewall allows good traffic to pass between one network and the other, and blocks bad traffic. The good and bad traffic, in each direction, should be defined in the Network Security Policy.

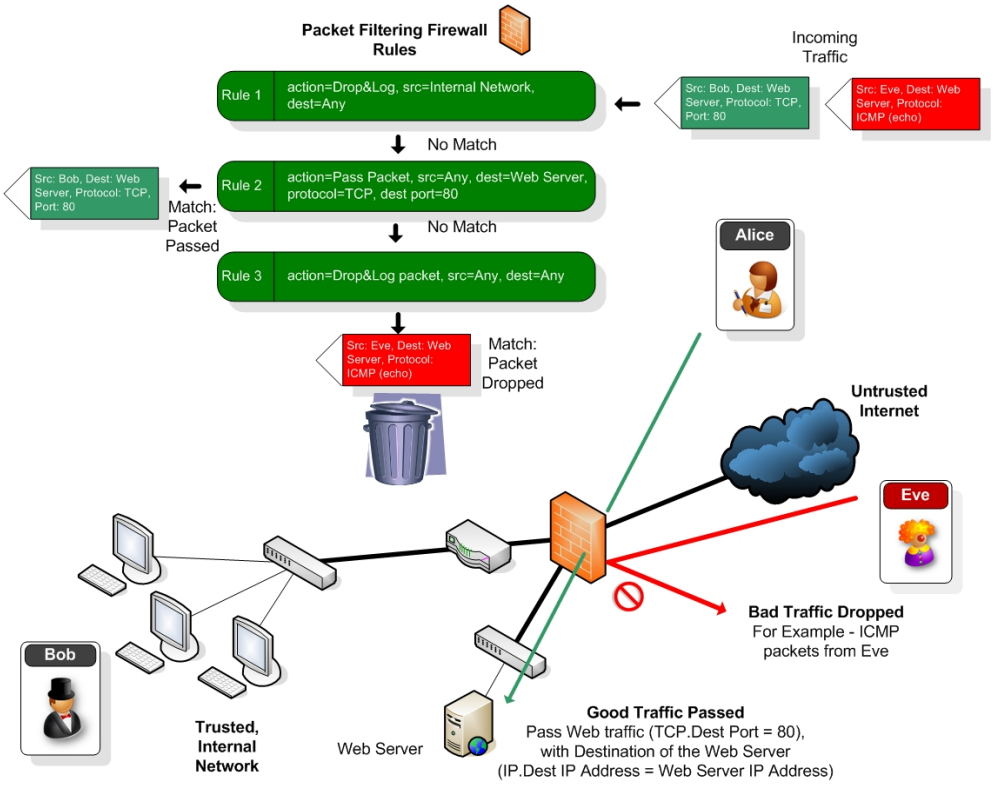

Figure 1.1 – Packet Filtering Process

1.3.3 Packet Filtering Firewalls

A basic type of firewall is a Packet Filtering firewall, which could be described as a stateless filtering and routing device. Firewalls are routers, which also perform filtering of the traffic which they route. Stateless packet filters perform their filtering based on a set of filtering rules, known as a rule set. They compare the rules with the information in the packet to make a decision on whether to route the packet on to its destination or filter out the packet (drop the packet).

The firewall inspects packets as they enter an interface (inbound or outbound). Each packet's header information at OSI model layers 3 and 4 (**I**nternet **P**rotocol (IP) & **T**ransport **C**ontrol **P**rotocol (TCP) headers) is decapsulated, and compared with the filtering rule set to decide if the packet can be routed or is to be dropped.

Static Packet Filtering

Packet filtering firewalls, use OSI layers 3 and 4 header information to filter the packets, and ignore the higher layers. They have one main strength, and that is they are very fast. Because of this they are typically used as a first line of defence, such as deployed on a boundary router or perimeter firewall. They are often used to perform bulk filtering, the front line defence of a network which is connected to the Internet.

This includes filtering such as IP Spoofing, RFC2827 and RFC1918 filtering, or mitigation of Reconnaissance attacks, for example ICMP traffic filtering. The higher level OSI layer filtering would be done by other devices behind the perimeter defences. Packet filtering firewalls are also very scalable as they work below, and are independent of, high level protocols such as user application protocols. They are flexible, because they work on the lower levels of the layered models, and so can provide filtering for almost any network based protocol.

Most operating systems and routing devices have packet filtering built in. Unix and Linux both have advanced packet filtering firewalls. Windows has a more basic packet filtering firewall, although the new Vista version has much improved features. Network devices which route traffic also usually have packet filtering built in, such as ACLs on Cisco routers and firewalls.

Listing 1.1 – Cisco Extended ACL Example

```
access-list 110 deny ip 127.0.0.0 0.255.255.255 any log
access-list 110 permit ip any 146.1.1.0 0.0.0.255
access-list 110 deny any any log
```

Packet Filtering Process

The firewall searches through a list of firewall rules (a rule set) to determine what to do with each packet. The typical actions are Pass the packet onto its destination (i.e. route it) or Drop the packet at the firewall. Rejecting the packet, by sending an ICMP packet back to the client, informing them that the packet was dropped, is another common option. Also, with most firewalls, it is usually possible to Log the action taken along with the packets information. This can then be analysed to identify attacks or problematic traffic.

If a packet does not match any specific rule in the rule set, a default action is applied. The packet is typically dropped on most firewalls. So we Pass what we know and Drop what we don't. This is what's known as a Closed Firewall, and is an example of a closed security stance, which is generally regarded as best practice, and is implemented by default on most firewalls. An example abstract filtering rule set, for incoming traffic, is shown in Figure 1.1. An example of a Cisco ACL packet filtering rule set is shown in 1.1.

1.4 Thesis Structure

Chapter 2 is an exploration of recent literature in the area of network security policies in general, and more specifically firewall, and other network traffic filtering device,

policies. A taxonomy of research into systems which can aid the system administrator in the management of these policies is also provided. The systems are categorised by which stage in the development life cycle they operate in, the functionality of the systems, as well as how usable and useful the proposed systems are, and includes reviews of seminal work by important authors in the field, including Mayer, Wool, and Ziskind, Al-Shaer and Hamed, Cuppens, Cuppens-Boulahia, and Alfaro, and Guttman.

Chapter 3 introduces a framework which could assist the system administrator with network security policy management. That is followed by a detailed design of one of the framework components, namely a firewall rule set parser and validation tool. The design of the tool has been created based on ideas, and conclusions, drawn from the literature review in Chapter 2. Also an outline design for an evaluation application, created to validate and assess the tool.

Chapter 4 examines the implementation of the software tool and its evaluation GUI application, and details how they were developed.

Chapter 5 describes the evaluation of the implemented tool, in terms of performance and completeness.

Chapter 6 includes the main conclusions of the thesis, and ideas for future work in this area of research.

Chapter 2

Literature review

2.1 Introduction

Network security should be based around security policies. From high-level natural language, non-technical, policies created by management, down to device and vendor specific policies, or configurations, written by network system administrators. There exists a multitude of research into policy-based network systems which has been undertaken. This chapter will give an overview of the different type of policies relating to security in networks, and a taxonomy of the research into systems which have been proposed to support the network administrators in difficult tasks of creating, managing and deploying these policies.

2.2 Security Policies

High level security policy documents should be written by upper management and should be the 'what' of security in the organisation. Without this definition of these security goals, it is difficult to use security mechanisms effectively [11]. The implementation, or technical policies, are created from the overall high level policy. This is the 'how' and it is used to enforce the security policy. The term policy can be used in the literature to describe both the high level policies as well as the low level implemented rules. The processes of security policy creation and implementation are shown in figure 2.1. A good definition of an overall security policy is taken from The Site Security handbook - RFC2196 [11]:

> "A security policy is a formal statement of the rules by which people who are given access to an organization's technology and information assets must abide." Fraser et al. [11].

The Site Security Handbook - RFC2196 Fraser et al. [11], is an excellent reference for network system administrators and management level decision makers, when creating network security polices. It contains a five step, iterative, process detailing

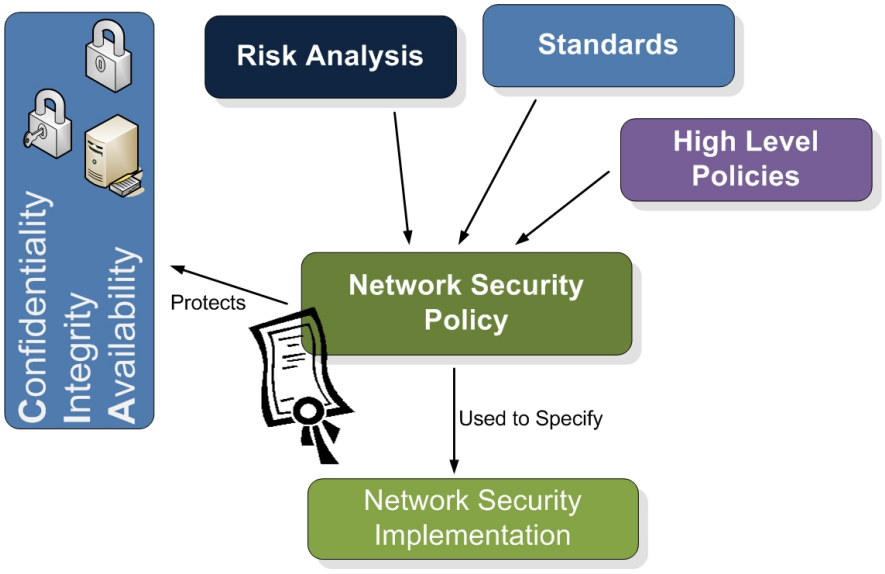

Figure 2.1 – Network Security Policy

the Security Policy creation and maintenance process. Note that this is an ongoing process, with regular reviews and auditing of security policies and mechanisms, providing feedback to improve the security policy. This matches two fundamental concepts: security being an integral part of the design and systems being an ongoing process - rather than simply the implementation of security products - which are both common throughout security literature [27, 20]. The Site Security Handbook is a good guide to creating policies, but it does not put forward a formal method of specifying the policies.

1. Identify what you are trying to protect.
2. Determine what you are trying to protect it from.
3. Determine how likely the threats are.
4. Implement measures which will protect your assets in a cost effective manner.
5. Review the process continuously and make improvements each time a weakness is found.

Industry recognised standards frameworks can be used to help create the security policy, based on industry best practices (as shown in Figure 2.1). Currently, the two best known frameworks are **C**ontrol **OB**jectives for **I**nformation and Related **T**echnology (COBIT) and IOS 27002 Code of practice for Information Security Management (previously IOS 17799). These provide detailed industry standards in IT security management, and audit compliance [14].

It is extremely important to involve the users in the implementation of a security policy. Understanding of security problems by users, and giving them clear and easy

to follow rules, can be a key factor in the successful implementation of the policy [28]. Danchev calls this the "Security Awareness program" and emphasises that the latest technical security measures, such as firewalls and **I**ntrusion **D**etection and **P**revention **S**ystem (IDPS)s, can be rendered useless by careless, or badly informed, end-users. The User's Security Handbook [29], the companion guide to the Site Security Handbook [11], can be used as a guide on how to educate users about the dangers of networked systems and how to keep data and communications safe.

Although management should have a great deal of input into the high level security policies, they may also need technical input due to the nature of the services they describe. Ideally they should be created by an individual with the power of the CEO of an organisation, and the technical ability of a system administrator. Typically administrators will help create the policies and the management will make the final decisions. Sometimes buisiness plans will trump security policy decisions [27].

2.3 Enforcing Policies

Security policies protect the confidentiality, integrity, and availability of the assets of an organisation. To enforce this, security services should to be deployed, such as authentication, encryption, antivirus software and firewalls. To do this the security policy documents are used to create technical security procedures, and guidelines, which can then be implemented in the network. Types of procedures include identification or authentication, access control or authorization, and accountability or auditing procedures. Procedures and guidelines would be created for each of the different type of security mechanisms to be used to enforce the policy. These technical mechanisms include authentication systems, firewalls, proxy servers, IDPSs, **V**irtual **P**rivate **N**etwork (VPN)s, and access control systems [30]. The guidelines are best practice suggestions for each, and the procedures are specifications for implementation of the specific security measures. This process is shown in Figure 2.2.

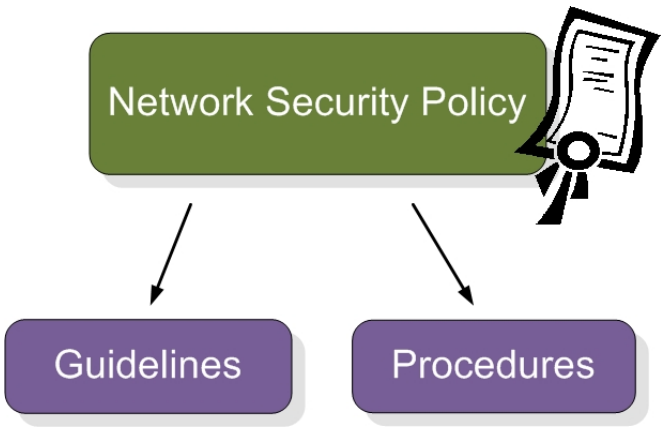

Figure 2.2 – Enforcing Security Policy

Security Policies can be split into several components, which combine to achieve the security goals of the organisation. These components are enforced by various security mechanisms or procedures. The authentication policy could be enforced using user names and passwords, software tokens, and VPNs. The accountability policy may use IDPS and firewalls to enforce auditing and incident response capabilities. The Access Control Policy, which this work focuses on, would typically use firewalls, VPNs and authorization systems to enforce access to resources [11].

The access control part of the security policy deals with making sure that authorized individuals can perform the tasks they are authorized to, and that others cannot. It is typically referred to as the 'access control policy' [30]. Access control makes sure that requests to access a specific resource are only granted if the request agrees with the security policy definition. In terms of networks, the most commonly used access control mechanisms are firewalls and filtering routers [16]. Firewalls control access to resources by filtering network traffic, only allowing access that is specified by the security policy.

The network access control policies, defining which traffic can cross network boundaries, are implemented as policies on network devices which have access control functionality, such as traffic filtering capabilities. The system administrator is typically tasked with manually creating these low level policies, or configurations. In order to determine whether to grant an access request, access control mechanisms uses a number of criteria. The primary criterion being the network address of the machine from which the traffic originates. Other criteria, which can decide whether access is granted or not, would include network service and destination of the traffic Corbitt [16]. The most common technique used to by firewalls to filter traffic is known as Packet Filtering.

Implementation of security mechanisms can be based on best practices, and several sets of guidelines exist. The **N**ational **S**ecurity **A**gency (NSA) and **N**ational **I**nstitute of **S**tandards and **T**echnology (NIST) publish configuration guidelines for implementing security controls. The Cisco **S**ecure **A**rchitecture **F**or **E**nterprise (SAFE) framework provides detailed guidelines for the implementation of network security mechanisms for various different sizes of networks and different site specific setups [31]. A compilation of the most common policy errors are detailed by Wool in his review of firewall configuration problems [10], and provide guidelines on what to avoid when implementing firewall policies.

2.3.1 Policy Enforcement Problems

If the high level policies are not defined correctly, the implementation cannot provide the security protection need by the organisation. As described by Wool in his review of firewall configuration problems [10], and as a mojor motivation for the firewall

auditing and testing system described in the latest research by Mayer, Wool, and Ziskind [3]:

> "the protection that these firewalls provide, is only as good as the policy they are configured to implement" – Mayer et al. [3]

The policy should be clear, concise, and easy for the administrator to follow. If a policy is not well designed, then it will not be enforced properly and the security goals will not be met Madigan et al. [32].

Conversely, polices are only as good as the configurations which enforce them [18, 33]. The enforcement of policies is not always an easy task. Policy management can be difficult as policies grow and become increasingly complex [7, 5, 10]. Blakley makes the following statement in [7].

> "Policies do not scale well and their complexity quickly increases as systems grow and diverge, which makes them unmanageable" – Blakley [7]

Madigan et al. categorises violations of security policies, and shows that violations from network issues were by far the largest type reported [32]. The author also states that the network violations were among the most time consuming to correct. This work was based on real security policy violations, on two university campuses, over a two year period. This could be a direct result of policy enforcement problems, such as policy configuration errors and anomalies. This is not surprising, as it is generally accepted by security experts that firewalls and other traffic filtering devices are poorly configured [10].

The configuration of a firewall is probably the most important factor in terms of the security a firewall provides [12], but are often configured incorrectly [10]. Firewall policies are made up of rule sets, and these rule sets are ever expanding due to new rules continually being added and very few removed, so device access policies tend to be large and always increasing in size [10, 34]. It follows that the management of these policies at the network device level can be extremely complex, error-prone and expensive as the policies expand [9]. The configurations are typically hand crafted and bespoke for each individual system by network administrators, which can be error prone work [15]. This is a serious problem as errors in the firewall policies mean that the intended security policy will not be enforced.

Mapping the high level security policies to the lower level implementation can be extremely difficult [30]. High level policies are written in a natural language, and describe security aims in terms of entities of an organisation, such as networks, users and resources. Enforcement policies are in terms of the points of enforcement, or devices. Firewalls and other devices have their own vendor specific languages and tools, which tend to be very low level. Many researchers have used the similie,

that these configuration languages are 'like programming in assembly languages' [9]. A conceptual gap exists between the two, and administrators can find it very difficult to map from one to the other correctly (to bridge the gap) [35, 15, 8]. GUIs are provided by some vendors, but most require the administrator to click through several windows, simple to understand a single rule fully [3].

Administrators are typically tasked with the creation of the low level device policies, which implement the security policy of the organization. In terms of firewalls, these are the firewall rule sets. The administrators will add, delete, and change the rules to match changes to the high level security requirements. For example, when new web servers are added to the organisation's network, new rules would be added to the perimeter firewalls to allow appropriate access to them from outside and inside the organisation's network. The complexity of the rule sets increase as they increase in size, but also the complexity can change depending on the rules used within them. For example, OSI layer 3 packet filtering is not as challenging to understand as OSI layer 4 or layer 7 filtering due to less filtering fields in each filtering rule. The filtering rules can be based on source address only or source and destination, as well as various other traffic attributes within a rule. Wool created a classification system for complexity of rule sets, based on the number of rules, the objects (traffic filtering parameters) and the interfaces rules could be applied to. In [10] the following rule set complexity measure is defined:

> Rule Complexity = Number of Rules + Network Objects + Number of Interfaces(Number of Interfaces -1) / 2 – Wool [10]

For most firewalls the ordering of the rules in a rule set are important, as in the common 'first match' filtering mechanism, the position of the rules in the rule set dictate if they are matched against traffic or not. The earlier in the rule set the higher the priority the rule has when matching against traffic [21]. Thus filtering rule sets, that use first match semantics, are complicated to create and amend due to this dependency on the rule ordering. As the size and complexity of the rule set increases it becomes more difficult for the administrator to predict the impact of a rule on the overall rule set. This makes rule sets extremely difficult to manage [10].

Effectiveness of security policies can be compromised due to poor policy management, especially when enforcing a security policy across a range of devices around a network [36]. Security policies can be spread over a range of different security devices [37]. Packets can take multiple paths through a network, with multiple filtering devices on each different path. An administrator needs to understand the interaction of combinations of these devices for each traffic path [3]. The low level configurations, which implement the security policy, can span heterogeneous networks, and may be spread over many network devices. Different vendors implement different algorithms and low level languages for configuring their devices. Thus, the scope of

the deployment in terms of devices, can also increase the difficulty of the task of policy enforcement for the system administrators [15]. If hundreds of network devices have to be coordinated to enforce a network security policy, manual translation of the policy into device configurations, can become extremely complex and error prone. If multiple devices such as firewalls, from different vendors, have to be used to create overlapping enforcement policies, in order to enforce a single global security policy, this adds even more complexity.

2.3.2 Policy Enforcement Solutions

Overview

Abrams and Bailey [38] describes a layered approach which can be taken with security policies. This can help the management, administrators, and users understand the policy and its implementation, as the mapping between the high level policy and the low level deployment is more clearly defined. Different individuals in an organisation will have a different view of the security policy. At a management level, an enterprise wide view concerning overall security objectives, such as protect the company's assets, will be taken. This is the highest level of abstraction of the policy, and it should be easy to understand, but does not contain guidance in how to carry out the security requirements. To do this policies at a lower level of abstraction need to be introduced. At the users level of abstraction, the policy is seen as rules relating to access to resources and data, such as systems and applications [38]. The high level policy is translated into this level of policy abstraction, providing guidance for users [29]. This level might be defined in terms of a finite-state machines, or as access matrixes [39, 30]. The lowest level of abstraction is the network implementation level, were the security mechanisms are deployed to satisfy the higher level specifications.

The policy can be shown in three different ways: first as a high level policy described in a natural language, secondly as formal statements to describe a model of the policy, or thirdly as a technical implementation [30, 38]. Natural language is prone to ambiguities, thus the high level policy may also have ambiguities in terms of the security requirements. If the policy is modeled in a formal language, the ambiguities can be reduced, or even removed completely. The formal model can also be used to audit the security procedures for compliance with the requirements, and it provides a specification free of any specific implementation methods, such as vendor specific tools and languages. The downside is that specialists are needed to work with such models, especially if they are mathematical models. Otherwise administrators, who implement the security mechanisms, would have to learn such modeling techniques. A formal model which is non mathematical, such as a more formal natural language, can be a good compromise. The ambiguities from the high level language can be reduced, and the modeling language would be understood by a wider range of individuals.

Tasking specialised individuals with such work, such as firewall administrators, is preferable to asking general system administrators to carry out such tasks [20].

Samarati and de Vimercati [30] outline how the different levels of abstraction provide the benefits of separating the requirements and design, from the implementation. The security requirements can be dealt with, regardless of how are they are to be implemented, and different methods of implementation can be compared against each other for the same requirements. The formal model could also be used to prove the security proposed is sound, and possibly even to automate the implementation of the security mechanisms, by creating device configurations from the model definition [40].

Various policy management systems have been proposed to help the network administrators with the creation, management, analysis, and auditing of these complex policies. The next section presents a taxonomy of research into these systems and tools, along with comparisons between the different approaches. Packet filtering firewalls are one of the most important mechanisms used by organisations to implement their security policies, and in the past 10 years, there has been a great deal of research into the areas of firewall policy management. The taxonomy and review of the literature is therefor focused on research into firewall policy-based systems.

2.4 Firewall Policy Management Systems

2.4.1 Introduction

Systems which can assist administrators in the creation and management of network access policies, can be used in the various stages of security development and operations. Network security should be a continual process, built around the network security policy, and it should be integrated into all stages of the SDLC [27]. Many definitions exist for the SDLC, with possibly the best known being the Waterfall Model, in which the development process is described by several phases in a downward flow. The phases for general development are typically Requirements Analysis, Design, Implementation, and Maintenance. More specifically, for security system development, the NIST process definition is defined in special publications 800-14 [41] and 800-64 [42]. This includes security system Initiation, Development, Implementation, Operation and Maintenance, and Disposal.

In the NIST definition the initiation phase is concerned with security planning, including documenting the need and high level requirements for the system. The Development phase includes the design, purchasing and creation of the system. Implementation involves the testing, and subsequent installation of the system. The

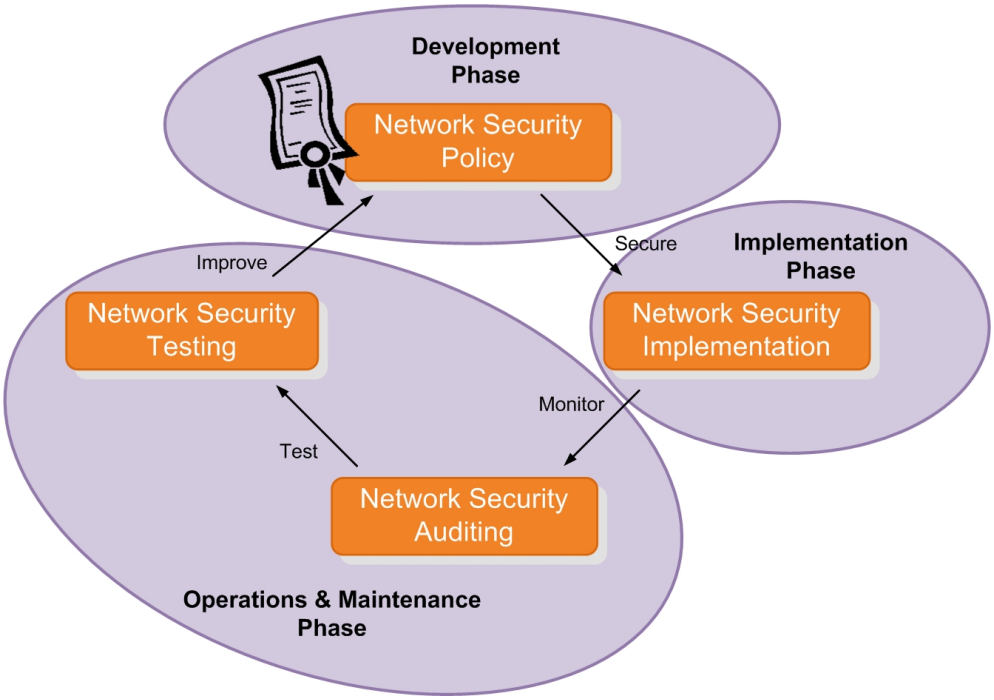

Figure 2.3 – System Life Cycle

Operation phase takes place once the system is in production, and includes the monitoring and auditing of the security mechanisms, and any configuration management.

The policy management systems, discussed in this section, can be categorised by which phase of the development life cycle they operate in. A simplified security development life cycle, showing the stages which contain the most relevant literature, is shown in 2.3.

2.4.2 Factors used to Compare Systems

- **Modeling Technique** High level, abstract, languages can be used to describe the policy model. Some systems introduce new high-level languages to describe firewall policies. These can be graphical languages, or abstract textual languages similar to high-level programming languages. These high level languages can then be translated into the vendor specific configuration language and be implemented on firewalls or routers. Some systems are based around the low level vendor specific languages, which the system administrators are already familiar with. Another method is to keep the language technically similar to the low level vendor languages, but abstract just enough to be used across a heterogeneous network, such as a XML based language. Systems can also be based around formal models, such as BDDs, bipartite graphs, or relational databases.

- **Top-Down or Bottom-Up** - Systems can be regarded as having a bottom-up

approach if the starting point, or input to the system, is from device policy. These systems may create an abstract policy model, analyse, or aggregate the policy into a higher level policy, but always start with a concrete policy from a network device. Top-down systems start from high level descriptions, spossibly a security policy. They typically create an abstract model of a policy, which then can be compiled into the low level technical policies and then deployed on devices.

- **Scope** Some of the systems describe a single network device policy, and some allow the description of entire network security policy which maps to policies for multiple firewalls.

- **User Interface** - The systems, which have been implemented as tools, are typically split between having either a CLI, or a GUI, which is an important consideration, depending on which tasks the system performs and which type of user the system is aimed at [43].

- **Administration tool design issues** Issues raised in recent literature on the subject of administration and security tool design, such as flexibility, customisability, usability, and error reporting techniques [44].

2.4.3 SDLC Development Phase

In this phase of the development life cycle, a risk assessment is carried out, which specifies the security requirements necessary to protect an organisation's network-based assets. Security requirements analysis are then carried out to define what is required to secure the network. This typically includes the creation of the high level security policy documents. Which security mechanisms are needed can then be planned for, and specific security controls - such as firewalls and filtering routers - can be developed and implemented [42].

Tsoumas and Tryfonas suggests a system to automate some of the development phase of the development life cycle [45]. Their system is top-down, and takes a a natural language description of a policy, such as the recommendations from a risk analysis, and creates a formal model of a security policy. This is an attempt to fill the gap between high level policy statements, risk analysis output and standards - shown in Figure 2.1 - and the network security policy definition. The typical approach is to task experienced administrators, along with management input, to translate the high level security requirements into a network security policy. Their research did not produce a prototype system and is left as theory, but it is an interesting idea, as potentially the system could relieve the administrators of the time consuming and error prone policy creation task [3, 9]. The authors suggest that their system, in conjunction with the output from a Risk Analysis tool, could produce a security

policy and recommendations for administrators concerning deployment issues. They also suggest that a formal model could be created using a high level policy language such as Ponder [46] or FLIP [47]. This would assist the administrators, as they would not have to learn the high level policy language used to describe the formal policy model. The output from this system could be generated in the syntax of an existing formal policy language. The system, would in this way, interface with high level policy language based systems described next.

High-level Policy Languages

Currently, there is no generally accepted high level model, which is commonly used for policy configuration [48]. The following quotes back this up:

> "The thing to note here is that there is no fixed terminology for the description of firewalls." – Fraser et al. [11]

> "generic data models and high-level languages for router configuration do not exist and are likely to remain elusive for some time" – Caldwell et al. [34]

Research by Guttman in 1997, which was funded by the NSA, led to a system called **N**etwork **P**olicy **T**ool (NPT). The system can define an overall access policy in a high level policy language, and verify that packet filtering specifications, which it generates, enforce the policy [49]. They suggest a high level access policy language to describe which packets can get where in an organisation's network. This is an abstract language, above and independent of, device configuration languages. The system deals with a global policy, covering the entire network access policy, not just the filtering at a single network boundary. Logical 'filtering postures', which define packet filtering at each device, are generated by the system. The motivation for the system was this creation of multiple filters to enforce the global access policy. The filters do not define the configuration of firewalls, but only the logical access policy. This assists the network administrators in delegating the filtering to various devices around the network, to enforce the overall policy, but the administrators would have to then manually create the device configurations.

As the system describes the relationship between devices, it needs a description of the networks topology. This is modelled using a bipartite graph, and specified using a policy specification language. The areas of the network are represented by nodes, as are the filtering devices which route traffic between the areas. The (undirected) edges between the nodes represent the interfaces of the devices connected to the different areas. Each interface can have an associated 'filtering posture' created, in both an

inbound and outbound directions. These postures are abstract representation of the bi-directional packet filtering that is enforced by routing and firewall devices.

A Lisp type language is used to describe the access policy and the network topology, and the filtering posture NPT generates. An example of the language is shown in Listing 2.1. This consists of source and destination hosts and areas, and the traffic which is allowed to flow between them. This system can be categorised as top-down, as the network administrator would have to manually create the abstract access policies in the modeling language, as well as the network topology information. The interface to the system is text based, and the administrator would have to learn the lisp type policy specification language and become familiar with this way of modeling the access policy. This is not the most complex of the high level policy languages encountered in the review of this literature, but it would still be a challenge for administrators to learn and use.

Listing 2.1 – NPT Language Code Snippet [49]

```
(defined-host-sets          ; define some host sets
  (internal                 ; new name
   ((areas engineering      ; two areas
           financial)))
```

Guttmans NPT system also can also audit the filters it generates, by comparing them to the global policy specification. This verifies that the filters created, correctly implement the overall policy. This conformance checking could be useful if the administrator makes changes to the filters, and wants to validate them against the overall access control policy. However, this auditing facility could only be used in the design stage of the SDLC, as once the filters have been implemented on devices, the validation would no longer provide any assurance of the implemented security measures. To use this in the operations phase, a mechanism would have to be added to reverse engineer the high level language policy from the configured devices, before running the auditing facility.

Later research by Guttman documents an evaluation of the NPT system, by generating filtering postures from a realistic sized network. They use a dozen filtering devices connecting sixteen network areas and their evaluation metric consisted of timings of the system runs. Their performance evaluation produced results which seem usable, with runs only taking seconds. Some problems were encountered with their system. These were mainly concerning the usability of the system. Specifically, administrators had difficulty when creating the network and policy definitions in the abstract language, and also when trying to translate the filtering postures into concrete device configurations, such as Cisco router ACLs [26]. The administrators found it hard to work with the abstract policy representations, particularly when trying to create a representation of an existing network policy [26]. Another tool, the

Atomizer, was created to assist the administrators in this task [50]. It can take Cisco ACL configurations (Cisco filtering language used on routers and firewalls) as input and generates abstract NPT policy specifications, combining common traffic filtering behaviors. The tool uses BDDs to model the traffic filtering policy generated from the device configurations, and 'atomizes' the sets of traffic within the filters together. The output of their NPT system, is still in the difficult to use policy language, and does not provide automatic generation of firewall rule sets.

The work of Bartal, Mayer, Nissim, and Wool started with the Firmato Firewall Management Toolkit [2] in 1999, which is another, top-down, configuration generation system. A high level policy language is used to create a, vendor independent, global policy, which can be compiled into individual vendor specific device configurations. The high level policy definition language is used to manually specify the network topology and the high level security policy. This is then translated into an entity-relationship model which is a role based model of the access policy and its relationship to the network topology.

One of the aims of the Firmato system was to separate the network topology and the high level policy definitions, which is an improvement on the work of Guttman, as changes to the topology does not mean that the policy has to be reworked. Other motivations behind the system were to abstract the policy away from low level languages, enabling vendor independent management of firewall configurations, and to automatically generate configurations, across multiple filtering devices, from the abstract global policy.

The high level policy language used to describe the abstract policy is called **M**odel **D**efinition **L**anguage (MDL) and is used to specify both the policy, and the network topology. An example of the MDL language is shown in Listing 2.2. Again, like the first generation system from Guttman [26], the administrator has to manually create these definitions.

Listing 2.2 – MDL Code Snippet [2]

```
corp_gw =
 {
   I_dmz_corp : { addr=ethero, INVIS,
                  file="RULES_I_dmz_corp" }
   I_corp_in : { addr=ether1, INVIS,
                 file="RULES_I_corp_in" }
   I_admin : { addr=ether2,
               ip = 111.222.3.1, file="RULES_I_admin"}
 } : LMF
```

The language is very different from the configuration languages of firewalls, and administrators generally find this type of language problematic to use Guttman [26].

In the testing of the Firmato system, existing firewall rules were converted into MDL manually, but rule sets with under 50 rules were used. The authors recognised that any more rules would have necessitated an automated mechanism for translating the rules into MDL. The system does improve on the work by Guttman [26], solving one of their systems main problems, in that the administrator does not now have to translate abstract filter definitions into the low level device configurations, as this can performed automatically. A limitation of both this system and Guttman's is that it models a closed firewall with only pass rules, and a single drop 'all other traffic' rule at the end of the rule set. This means the rule set is conflict free, but may have more rules than necessary, and may be more difficult to understand for the administrator. The interface provided to the administrator is textual, using text files, and no GUI was provided. This was regarded as a surprise success of the system, and an important feature for the users [2]. As recent research by Haber and Bailey [43] shows, this is not so surprising. A survey carried out by the authors, found that the majority of system administrators preferred a text based CLI, due to CLIs being more reliable, faster, and more robust.

Around the same time, research at Cisco Systems by Hinrichs [35], produce a policy-based management system. This can model abstract filtering device polices, and automatically create the low level device configurations, in the form of Cisco ACLs. The system can create filtering device policies, for multiple devices, from global policy rules, as well as performing some basic rule set analysis. The motivation for the system is summarised in the title of their research paper 'Policy-Based Management: Bridging the Gap' [35]. The gap between device configurations and the high level, natural language, security policy is difficult to bridge for system administrators, and the research introduces a functional language to express the high level policy. This is done in a precise way, using nested sets of conditional statements. An example of the language, taken from [35], is shown in Listing 2.4.3. This language has to be manually created by the system administrator, which is a drawback of the system, as the administrator would have to learn this abstract high level language as well as the low level device languages.

Listing 2.3 – Code Snippet of an HTTP Policy [35]

```
corp_gw =
 {
   If Service is HTTP
     If Destination is S
        If Source is H
           Service level is premium
           Permit
        Else If Source is N1 or N4
           If Source is N4
```

```
        Use encrypting tunnel
    Permit
```

The system was implementation as part of the **C**isco **S**ecure **P**olicy **M**anager (CSPM) tool [51], and its functionality has since been incorporated into the Cisco's latest security management tool, Cisco Security Manager [1]. The CSPM interface is a GUI which provides the administrator with a tree view to implement both the policy and the network topology information. The topology tree represents the enforcement devices (firewalls and filtering routers) and the network areas between the devices. The policy tree contains a policy structure with the individual policies, defined in the abstract policy language. These can then be applied to the enforcement devices in the topology tree to enforce policies between the network zones. Figure 2.4, taken from [1],shows the interface to the CSPM tool. Note the two trees, the top represents the policy, and the bottom the network topology. Note also that the inner security policy has been applied to the PIX2 firewall device, which seems to be segmenting a specific internal network from the general internal network network.

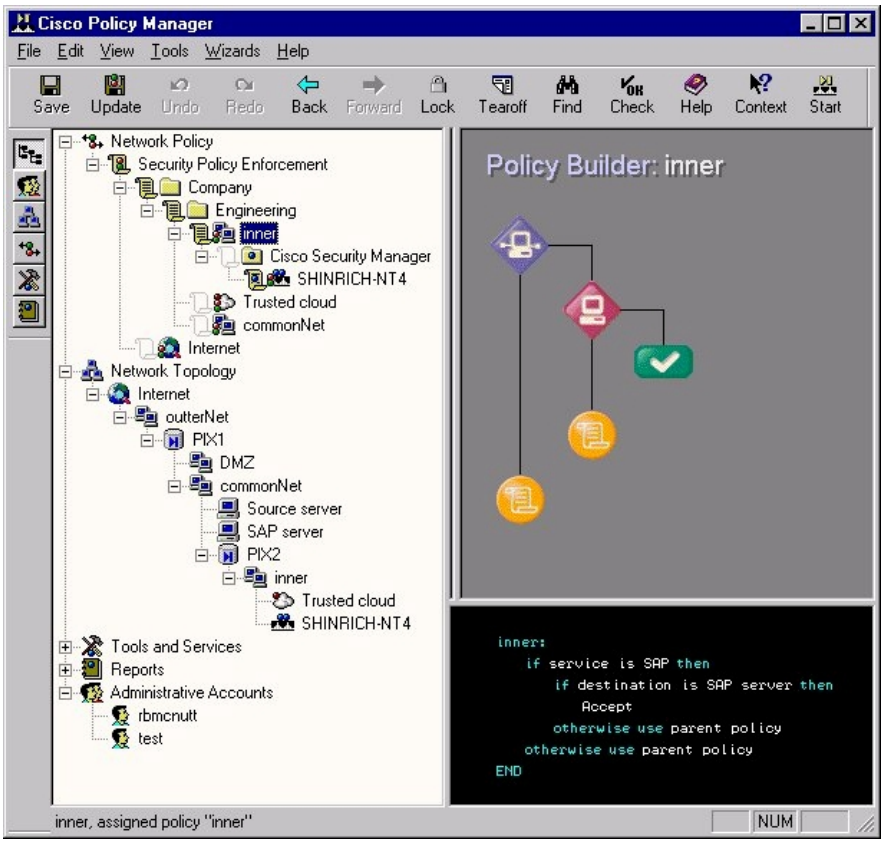

Figure 2.4 – Cisco Security Policy Manager GUI [1]

The generation of the filtering device configurations consists of four steps. Firstly, the high level policy is applied to device and the policy is distilled into rules which are applicable to the individual devices. The authors call this process pruning, and involves a mechanism to detect if policy rules specify traffic which could pass through

each device. In this way, devices only receive rules which are needed on the device. This is an improvement on, and in contrast with, the systems discussed previously which either apply all rules to all devices, or perform very minimal pruning. The second step in the process, is some simple rule set analysis. This checks that the device has resources to carry out the policy, such as the memory available for filtering rules. It also performs some rule set anomaly analysis, checking for conflicting, or overlapping, filtering rules. An example of conflicting rules would be if the filter has the same source and destination address and the same service, but different actions. The administrator is warned about any analysis problems and the administrator would have to decide on the solution, such as which of the two conflicting rules would be used in the rule set. The next step is to generate the device filters. The tool creates an intermediate, abstract, filter rule set for each device and stores them in a database. This stores the semantics of the device configurations to be created, but leaves the creation to an agent which reads the generic filter rules and creates low level rules in the device configuration language. This means filtering devices could be interchanged and the fourth stage repeated to create configurations for the new devices. Although this is a Cisco specific closed system, it could in theory be used to create other vendor device configurations, if agents were added to do this.

The system is MS Windows based, and provides only a GUI for the network administrator. There is no support for the CLI and this means the tool, even in its latest form, can not be scripted. This is something that is sought after by administrators, so the tool can be combined with other tools, and provides customisation in there work practices [44]. Small configuration changes could be time consuming to perform in this type of system, as the high level policies would have to be studied and amended, and the fourth stage process run again. This type of all or nothing configuration generation my not be very useable for the system administrator [13].

Several other similar approaches, using high level policy languages and creating low level device configurations from them, appear in the literature. Uribe and Cheung [52] describes a similar system to Guttman's, with the addition of modeling and configuring **N**etwork **I**ntrusion **D**etection **S**ystem (NIDS) as well as firewalls. PRESTO is a configuration management system for routers [53]. It is interesting, in the way it extends the low level configuration language, creating a hybrid scripting language rather than introducing an entirely new language. Templates, or configlets, are used with information embedded into the low level configuration language. These are then used at runtime to generate the low level configurations, with a database providing the content of the templates. The Ponder policy specification language can be used to specify the entire high level security policy, including access control policies, user authorisation policies, and traffic filtering policies Damianou et al. [46]. Ponder seems more complex than is needed to specify device filtering policies. A recently proposed high level firewall policy modeling language, FLIP [47] can also be compiled into low

level device configurations. The scope of the FLIP system is global and can manage firewalls across an entire network. FLIP generates conflict free rules automatically, by performing conflict analysis as it generates the low level device configutaions. This improves on most of the systems described, which would need to be analysed separately for rule conflicts, although tying the functionality together like this means less flexibility [44]. More recently, in 2008, work at the University of Seville, produced another high level policy language **A**bstract **F**irewall **P**olicy language (AFPL) [54]. This is again intends to fill the gap between high level network security policies and low level firewall languages. It has been designed to be simpler than some of the proceeding high level languages, but still provide all the functionality needed to describe filtering policies. It can be automatically compiled into the leading vendors firewall filtering languages. These high level policy languages have a common problem, in that the administrators generally do not welcome the overheads involved with learning and using them [9].

2.4.4 SDLC Implementation Phase

Policy Testing Systems

Testing of packet filtering rule sets was explored by Hazelhurst et al. in the late 1990's [55]. One of their main motivations for their system was the analysis of low level rule sets to understand the policy they implement. This ended up being the main focus of research, with a query-based system being developed to analyse rule sets. BDDs were used to represent firewall and router access policies. Each rule, in the rule set, can be converted into a boolean expression, and the boolean expressions combined in a BDD. Queries are used to pose questions about the rules, which can then be answered using the BDDs, which represent the rules. An example of these 'what if' queries might be 'which destination addresses can packets reach from a source address and a certain port'. The user can analyse, and so test, a policy by querying the rule set in various ways. This can be used to validate and explore the policy before deploying the rule set onto filtering devices. The language used to specify the queries is a functional language 'FL', and the output is a textual representation of the query answer. The FL query language is a low level and is difficult to use, for the system administrator when creating the queries. This was recognised, and the system was improved to include a GUI for easier querying of the policy [56]. The authors also recognised that the best interface was a GUI for visualising important information about the rule set, and for basic querying, but a textual interface was better suited for an advanced user to develop more powerful queries [56]. The scope of the system only extends to a single rule set, but later research expanded query-based systems to cover entire networks, some of which are covered later in this chapter. The primary analysis mechanism is the manual query and answer system, but a basic rule set

conflict analysis process was also developed. This automated the task of detecting redundant rules in the rules set prior to deployment. This seems to be one of the first systems to perform conflict analysis within rule sets and is cited by most research in the area.

The research of Mayer, Wool, and Ziskind continued with the creation of the FANG system [23]. This was built from earlier research into the Firmato system [2], and FANG is actually an analysis engine which runs on top of the same Firmato policy model. It can be used in the Implementation phase, to test a policy before it is deployed, and could also be use in the Operations phase to audit a deployed policy. It has functionality to build the model from existing filtering configurations, so it is classed as a bottom-up system. It can take Cisco Router configuration files or Lucent Firewall files as input to create the policy model. It uses a separate parser module for each filtering device it supports. This is a good system as it support extensibility. The system works on multiple filtering devices, and so a global policy can be tested. A network topology has to be entered, and this is still done manually using the MDL language, the same way as described for the Firmato system. The queries which can be performed on the FANG system are based around a triple of source host group (source network address range), a destination host group (destination network address range) and a network service. Queries can be created such as (*, web_servers, http_services) to find the answers to questions, such as 'which systems have access to the organisations web servers'. A GUI was created to perform queries, and drop down menus implement the query triples. An example, taken from [2], showing the result of a query asking 'which services can get from the internal network to the **D**e**M**ilitarised **Z**one (DMZ) network', is shown in figure 2.5.

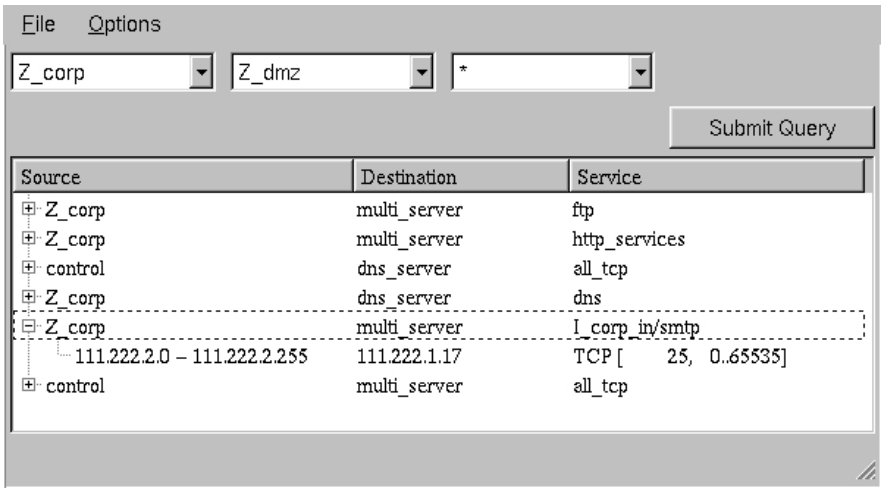

Figure 2.5 – The FANG systems GUI showing results of a query [2]

A more recent, but similar query-based system, which was created for the Linux iptables firewall, is ITVal [57]. It uses **M**ulti-way **D**ecision **D**iagram (MDD)s rather then BDDs, but operates in much the same way. Their main motivation was to provide

a simple query tool to aid in firewall configuration, so an administrator could test a firewall configuration before depolying it. The query language is designed to be simple and natural language based. A single iptables rule set can be read in by the tool and a MDD model built. The queries are created in the english based query language and return a simple textual answer, in a similar way as the FANG system does.

The main problem with analysis systems using queries, is that they have to be created by the administrator. The system administrator has to learn another query language, as well as knowing which queries to perform. The onus is on the administrator to work work out what, and when, to query.

These off-line passive testing systems have advantages over active testing systems, such as vulnerability testing or penetration testing, as they can be performed before policies are deployed. With active testing the policy has to be deployed before testing, and if problems are found, the production network is vulnerable until a solution can be deployed [23].

Policy Deployment

The **S**imple **N**etwork **M**anagement **P**rotocol (SNMP) protocol [58] was developed by the **I**nternet **E**ngineering **T**ask **F**orce (IETF) in the 1980's as a comprehensive network management system. It was intended to be a standard way of managing increasing numbers of devices, across heterogeneous networks. Its functionality includes the ability to configure managed devices remotely. It is implemented as 'agents' on network devices, and the configuration is modeled using **M**anagement **I**nformation **B**ase (MIB)s which store information on each specific device. In a workshop held by the IETF in 2002, attended by protocol developers and network administrators, it was found that SNMP was not being used to configure network devices as much as they originally intended. One of the main issues raised was device manufacturers have not implemented all the parts of the protocol needed to read and write configurations to and from devices. As the following, from the 2003 workshop [59] on the subject, states:

> "There is too little deployment of writable MIB modules. While there are some notable exceptions in areas, such as cable modems where writable MIB modules are essential, it appears that router equipment is usually not fully configurable via SNMP" – Schoenwaelder [59]

> "It is usually not possible to retrieve complete device configurations via SNMP so that they can be compared with previous configurations or checked for consistency across devices. There is usually only incomplete

> coverage of device features via the SNMP interface, and there is a lack of differentiation between configuration data and operational state data for many features." – Schoenwaelder [59]

It seems network device vendors are reluctant to fully implement the SNMP agents on their devices, thus their own proprietary languages are needed to manage the device configurations. SNMP is widely supported by vendors, but mostly the monitoring part of the protocol is implemented, rather than the configuration part [60]. This could be a good business decision for vendors, as rather than the administrator using the same IETF protocol on any device, they have to become familiar with that vendor's low level device specific language, and perhaps have to take some training courses from that vendor. For example, Cisco **O**perating **S**ystem (OS)s have subtle differences across their range of network products, and different certifications are needed to manage the different devices. Investment in training administrators in a vendor's products can make it difficult to justify using any other vendor's products [34]. Vendors typically provide GUI-based products to perform some management of the devices, but these are normally aimed at simple initial setups and inexperiences users performing basic tasks [59]. Network administrators invariably have to use the CLI to perform more complicated management work. In the workshop reported in Schoenwaelder [59] the CLI was found to be the administrators favorite way of configuring devices.

Apart from SNMP, several other systems were also discussed at the workshop including the **C**ommon **I**nformation **M**odel (CIM) [61], the **C**ommon **O**pen **P**olicy **S**ervice (COPS) [62], and using the device CLI and **S**ecure **SH**ell (SSH), for configuration management. It was decided that neither CIM or COPS addressed the needs of the users, and that a new vendor independent configuration management protocol was needed. The XML based Netconf protocol [63] was born out of the recommendations from this workshop. Netconf was designed to be a simple mechanism through which device configurations can be managed, using an XML-based system [64]. The entire, or partial, XML encoded configurations, of a Netconf enabled device can be retrieved, updated, and deployed back to the device by remote management applications. The protocols control messages are encoded in XML, as well as the data being sent. Major network device vendors, such as Cisco and Juniper Networks, now have XML-based agents in their latest products and are participating in Netconf standardisation [65]. Cisco Netconf configuration is detailed in [66], and Juniper in [67]. XML has many advantages over SNMP, such as XML is human readable, there are many standard tools and libraries for parsing and processing XML, and XML-based languages are extensible by the user [68].

2.4.5 SDLC Operation and Maintenance Phase

The operations phase of the SDLC includes configuration management of deployed security systems. This could range from small changes to a single device, to implementation of an new security policy over the entire network. This phase also includes monitoring and auditing of deployed systems, checking for compliance with policies and procedures. This can be done by comparing the implemented mechanisms with the higher level policy definitions. System Administrators can also use these auditing systems to help with their understanding of deployed policy implementations. For example, to create high level policies for management where none currently exist, or during the process of making changes to device configurations. Understanding deployed policies can be extremely challenging [69], especially if the policy is being enforced across many devices over an entire network. Periodic testing of security mechanisms is also recommended in this phase of the life cycle, to check the systems for, and alert administrators to, the latest security vulnerabilities. The auditing and testing processes can be carried out in parallel, and both aim to highlight any weaknesses in the current security solution, and possibly to suggest improvements. This would then feed back into the development phase of the SDLC, as part of a continual process, to improve the security policies and implementation [70, 14]. This continual security process is shown in Figure 2.3, and summarised nicely by the following quote from Fraser et al. [11].

"Implement measures which will protect your assets in a cost effective manner. Review the process continuously and make improvements each time a weakness is found." Fraser et al. [11]

Policy Auditing Systems

Some of the systems discussed previously can also be used for testing or auditing of deployed access control policies. Such as the FANG system Mayer et al. [23]. It can reverse engineer a model of a policy from firewall configurations. The administrator can then query the policy, to become familiar with it before changes are made, or to check it matches the overall security requirements.

This auditing system has been improved on, by the creation of the Lumeta Firewall Analyser [69, 3]. This improved on the Fang system by automatically creating the queries needed to analyse the the firewall policy model. Lumeta generates what the authors describe as the most interesting queries, and then displays the answers to these queries. This tries to highlight possible risks in the firewall policy, and to limit the need for user input to the system. The authors recognised the fact that one of the problems with their earlier system was that the administrator had to decide which queries to ask the system, and then create the queries manually. This is described as

a major usability problem with FANG [69].

In the FANG system, the administrator has to enter the network topology description manually using the Firmato MDL language. Using this language was raised as a problem by beta testers and in the new Lumeta system, the routing table is used to create this automatically. The GUI to FANG, shown in Figure 2.5, was also replaced as it was deemed difficult to use by testers. This has been replaced by a batch process which performs a comprehensive simulation of traffic through the firewall policy and reports on this. This is interesting as the administrator users found the original Firmato CLI interface easy to use, and yet it was replaced with a GUI. This shows the design was perhaps not tailored to the type of user correctly [44]. The output from the system is now a report in the form of web pages, with the ability to drill down into more detail if the users needs to. Using HTML to provide this type of flexible reporting is described as an ideal mechanism for security analysis tools Jaferian et al. [44]. The FANG system can only translate Lucent Managed Firewall, which does not have a large market share. The Lumeta system added parser modules for CheckPoint firewall and Cisco ACL configurations , so heterogeneous networks could be modeled, and therefor the product would be useful to a wider audience. The low level configurations are abstracted to the Lucent Managed Firewall based language used by Firmato and FANG, and the analysis query engine uses this as input. The Lucent Managed Firewall language was used as it is contains high level constructs and is easy to parse. The Lumeta architecture, taken from [3], is shown in 2.6.

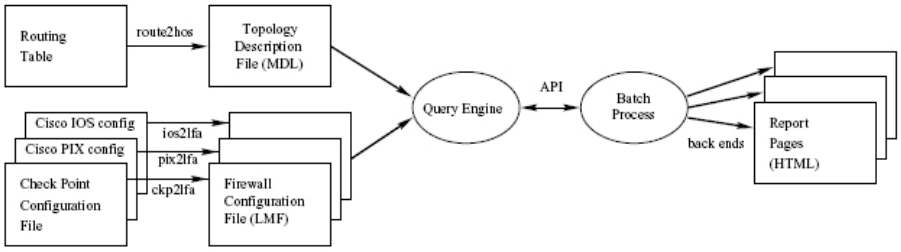

Figure 2.6 – The Lumeta system architecture [3]

The Lumeta system has since been developed into a commercial product, the Algosec Firewall Analyser from Algorithmic Security Inc. [71], which provides multi-vendor firewall analysis, monitoring and auditing. The Firewall Analyser system supports the major enterprise firewall platforms: CheckPoint Firewall-1 software firewall (which runs on various hardwares platforms), Cisco PIX, ASA and Router firewalls, and Juniper Netscreen.

DePaul University in Chicago has contributed greatly to research in the areas of firewall policy modeling and analysis. Al-Shaer and Hamed have been the main contributors, with over a dozen publications between them. Their first research into firewall policy analysis was concerned with auditing legacy firewall policies to automatically discover conflicts and anomalies in firewall policies. This anomaly checking can also

assist administrators when editing the deployed policies [5, 72]. The rule set conflict detection aims to highlight possible problems in a rule set based on the order of the rules, and the dependencies between rules due to the ordering. For example a more general rule before a more specific rule in a rule set, would mean the more specific rule would never be reached. The more specific rule is classified as 'shadowed' by the first rule if the filtering actions taken are different (pass and drop), or 'redundant' if the actions are the same (for example, both rules pass the packet). These are classed, by the authors, as rule set 'anomalies' [72].

The FPA tool was created, based around a formal model of the firewall rules and the relationships between them. Modeling of the filtering policies is done using BDDs and algorithms to detect anomalies in the rule set model, have been created. To prove the concept they demonstrate a five tuple filtering syntax, which is used to describe the filtering rules used as input to the system. This maps directly from current low level filtering languages, such as Cisco ACLs. The format of the five tuple filtering rule is shown in Listing 2.4. The literature only shows examples of these commonly used filtering fields, but the authors state this could easily be extended to include any other filtering fields from low level languages [5]. This could be extended to include the filtering options available in modern low level filtering languages, such as Cisco ACLs or Linux IP Tables. Note that the filters used in the examples only use classful ranges of IP addresses, and classless ranges would need a more sophisticated wildcard specification. Al-Shaer and Hamed define all the possible relationships between rules, which are then proved mathematically to be the union of all possible relations [5].

Listing 2.4 – Format of filtering rules, used as input to the FPA tool [5]

```
<order>  <protocol><src_ip><src_port><dst_ip><dst_port>  <action>
```

The firewall rule set, and the relations within, are then modeled as a BDD. This is then represented as a policy tree, with nodes on the tree representing filtering fields, and branches being the values. Each path through the tree represents a rule in the input rule set. This model was chosen, as the rule set and the anomalies can be visualised by the users. An example of this type of tree, taken from [4], is shown in Figure 2.7.

Algorithms to detect any anomalies within the rule set are then run, and can be displayed to the user in the FPA tool interface. The interface provided by the system is a GUI, which can show the policy tree, and any rule set anomalies. The interface is shown in Figure 2.8, which is taken from [5]. The policy tree is shown in the left hand pane, and the discovered anomalies in the right hand pane. The rule set is shown in the bottom pane, with the conflicts highlighted. The FPA tool also allows the user to maintain the rule set, inserting, modifying and deleting rules. As the user edits the rules, the tool provides feedback on any conflicts that may be introduced by these

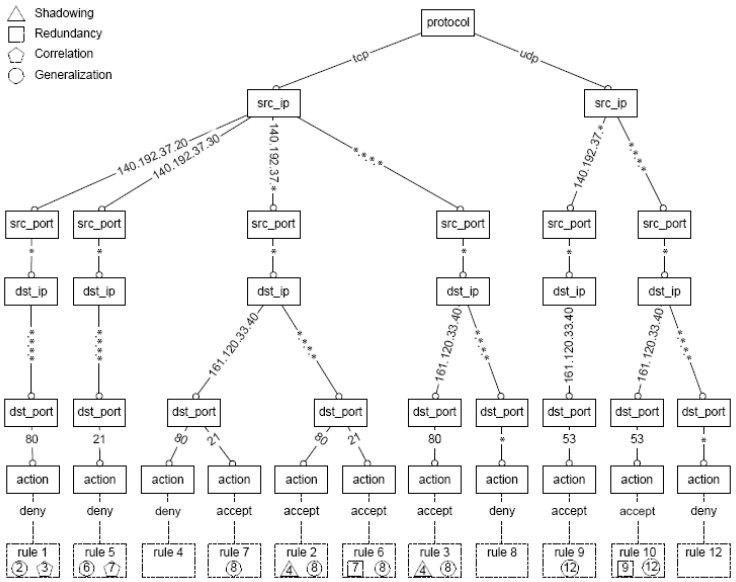

Figure 2.7 – FPA tool BDD Tree model of a filtering rule set [4]

actions.

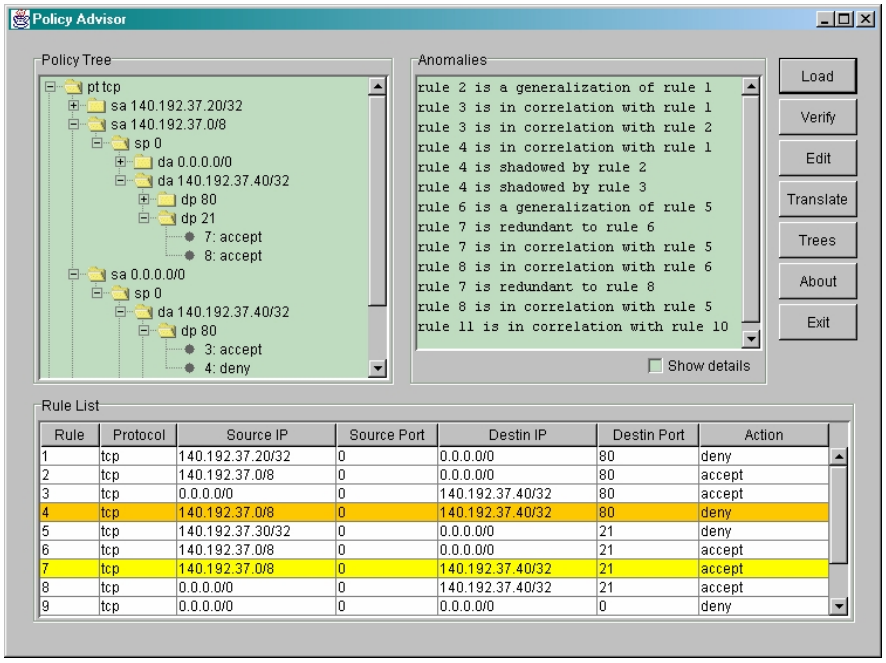

Figure 2.8 – FPA tool GUI [5]

Although the use of system is described as analysis of legacy firewall policies, by the authors, no automatic importing of these policies is described. The system seems to need an administrator to manually translate the low level rule set into the five tuple syntax. This would classify the systems top-down, however, the translation of deployed policies is not difficult as it is a direct mapping from most firewall configuration languages. Fixed format configurations from devices can be easily parsed

using a scripting language, such as Perl [69].

The authors regarded this system as a step forward from the query-based analysis systems such as FANG [23] and Lumeta [69]. The main reason for this is that it takes much more effort to redefine deployed rules sets into the high level policy specification languages used by these systems, rather than analysing the rule sets directly with the FPA system [4].

The scope of the FPA system was a single device and rule set, but this was extended to anomaly detection across multiple firewalls [36, 24]. Further research extended the system to include other filtering devices, including **I**ntrusion **D**etection **S**ystem (IDS), and VPN gateways, and the inter-device anomaly analysis, across the global policy [73].

The FIREMAN Firewall Analysis system Yuan et al. [74] can perform analysis of firewall filtering, detecting redundant and conflicting rules, which may point to mis-configurations. Again, BDDs are used to model one or more firewalls. The analysis can discover mis-configurations, similar to anomalies defined in the FPA system, as well as detecting policy violations in the policy. The policy violations are based on a blacklist or whitelist input by the user, as well as a general policy for all rule sets based on the twelve common firewall configuration errors identified by Wool [10]. The model is created with a bottom-up approach, by parsing device configuration files. An evaluation of the system was carried out, by creating artificial rule sets of up to 800 rules. Performance evaluation was carried out based on timing metrics. The experimental conditions used, were rule sets increasing size. The system can analyse up to 800 rules in less than 3 seconds.

Research at AT&T labs produced a configuration auditing and management system called EDGE Caldwell et al. [34]. It creates an abstract model of the networks routing information by reading in and processing entire device configuration files. The network is modeled using an entity relationship model stored in a database of network services described in the configuration files. Off-line analysis of the network configuration is then provided, and reports are generated. These can be used to identify problems such as inconsistent routing configurations on devices. The administrator can decide whether to correct these highlighted problems, before the system rewrites the configurations back out to the devices. This is an example of a pragmatic, bottom-up, approach to modeling from network configurations. The motivation for the system is similar to many of the firewall management systems. The configuration languages used to configure these complex routing systems are difficult for the administrator to get right. Automation provisioning of configurations is proposed as a solution in this case, with the database model of the current network being used to facilitate this. The interface to the system is via the EDGE web site. EDGE users

have access to their own networks data. Various network visualisations are available to help the administrator understand the policies, such as network topology and routing information. Various web-based reports are also available on request. This system has been successfully commercialised, and runs daily on thousands of enterprise networks, helping administrators manage complex networks [34].

One of the main problem with systems which convert to and from vendor specific configurations, such as the systems discussed in this section, is that the configuration languages tend to change rapidly. For example, Ciscos ACL filtering language has had many additional features added to it since the first version in the mid 1990s. This means that the parser modules have to be continually reviewed, to keep them up to date with the latest language changes [34].

"Command line interfaces often lack proper version control for the syntax and the semantics. It is therefore time consuming and error prone to maintain programs or scripts that interface with different versions of a command line interface." – Schoenwaelder [59]

More recently, as part of another configuration management system being developed by Caldwell et al., some interesting research into Cisco Router IOS parsing and modeling was carried out at AT&T Labs. In [75], a learning system and adaptive parser are described. The overall system will be able to process and extract information from existing network configurations, much as described in their previous work with the EDGE system [34], but the parser can adapt to changes in the configuration language being parsed. The parser is automatically generated from valid configurations, which are fed into the learning system. The system, if perfected, would not need manual changes to be made whenever the configuration language version had new features introduced. This would overcome one of the major problems with the bottom-up approach to any type of policy modeling, especially when dealing with rapidly changing configuration languages such as Cisco device operating systems.

Reverse Engineering Security Policies

Al-Shaer and Hamed, at DePaul University seem to have produced the first research into the reverse engineering of natural language, high level, policies from existing device configurations. This was as recently as 2002 [5, 76]. They aggregate network services together, using their FPA system model as a base, and produce a basic text based abstract policy description from a filtering rule set. They take their BDD model of the filtering rule set, and create another BDD based on network services from that. Services can then be aggregated together, changing it into a form which can be presented to the user in a natural language. The interface for the tool was a GUI showing the network services based tree and the textual representation of the policy. Figure 2.9, taken from [5], shows the network service-based tree and the high level

policy which has been inferred.

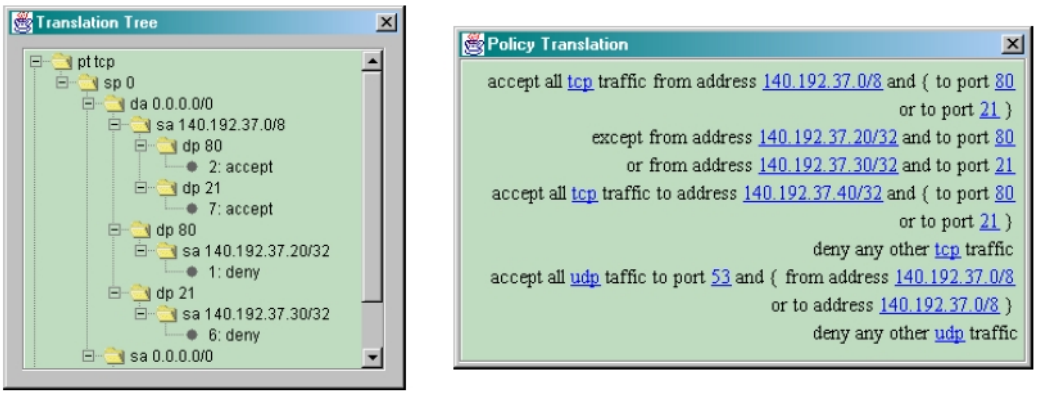

Figure 2.9 – Policy Inference Tool GUI [5]

The creation of an abstract high level policy from low level device configurations is a fairly new subject for research. The 2007 paper by Tongaonkar, Inamdar, and Sekar describes a technique to extract the high level security policy from analysis of the low level representation of a rule set [77]. Their system attempts to flatten the rule set, by aggregating overlapping rules together, thus eliminating the dependency on the order of the rules. This also reduces the size and complexity of the rule set, hopefully giving the administrator a better understanding of the policy it represents. This is a similar technique to the Al-Shaer and Hamed [5] system.

More recentlt, Bishop and Peisert [8] developed a system of reverse engineering access control policies from Linux file system access control configurations. This doesn't involve firewall policies however, only user access policies from Linux password files.

Around the same time Golnabi et al. have also worked on the problem of inferring high level policies [78], but their approach is based around an active system, and the data mining of the device logs, and not by static analysis of the device configurations.

Policy Visualisation

Another way to analyse or validate a policy is to graphically represent the rule set in some form. This technique has been used in several tools to some extent, for example the FPA system Al-Shaer and Hamed uses visualisation of BDDs tree to represent the firewall policy. This was regarded as an important design feature. The choice of a BDD as a model was used over, faster modeling techniques for simplicity and ease of visualisation to the user [72].

Tran et al. [79] describes a firewall packet filter visualisation tool, PolicyVis. A GUI provides the administrator with a visualisation of packets passed or blocked by the firewall, based on a set of specified query parameters. The administrator can enter

queries, much like the Firmato or FANG systems, and instead of textual answers, a visualisation of the filtered traffic is generated. Like the query-based systems reviewed earlier in the Chapter, this leaves the administrator to create the queries.

Wong [9] discusses a system, to compliment the CLI, visualising the most complex parts of the policy. Their system was based around improving the usability of network administrators current systems, by complimenting them with visualisations, but not replacing them. Their approach is different from the clean-slate approaches taken in some of the systems reviewed in this section, such as with new high level policy specification languages. They assume that the administrator prefers the low level CLI based tools, they are familiar with, and attempt to give additional usablility through visaulisation rather than replacing the CLI tools. This assumption is made, based on research into the network administration community and the context of the work administrators perform [43, 80].

2.5 Conclusions

This chapter explored literature in the areas of network security, security policies, policy enforcement, and firewall policy management systems. A taxonomy of systems which aid the administrator in the management of firewall policies, was also carried out.

It has been shown that security policies play a central and important role in network security. High level policies are created with input from management, and are typically written in a natural language. These policies then have to be mapped to the technical policies which network devices enforce. The process of mapping these high level policies to low level device configurations is normally performed manually by network administrators. This task is difficult and error prone as there is a large conceptual gap between the high level policies and the configurations. The low level firewall policies themselves are difficult to understand, reverse engineer, and maintain due to the complexity in large rule sets, and the fact that overall policies are sometime enforced over multiple firewalls and filtering devices. Many systems to help with the understanding and management of these policies have been proposed in the literature.

Much research into policy-based systems has been carried out, including high level abstract policy languages, query-based rule set testing systems, and firewall rule set auditing systems. High level languages, used to bridge the gap between high level policies and low level firewall configurations have been developed, but have some problems. A common problem with abstract policy languages is that they need specialist to configure them. The administrator typically does not want to, or have the time to learn these new languages. Some of the languages also have limitations with

the features they support. Query-based analysis systems have been produced specifically for firewall auditing, and analyse the firewall rule set by simulating network traffic passing through the firewall, based on queries input by the user. Some have overcome the problem of abstract language based systems, as they use models hidden from the user, which can be created from device configurations automatically. Some query-based systems have problems also, as they put the onus on the administrator to create the queries, and sometimes to learn a new query language which leads to the same problems as high level modeling languages. Other off-line auditing systems were reviewed, including rule set anomaly detection systems, visualisation systems, and high level policy reverse engineering systems - which are still in their infancy.

From the review, it was highlighted that off-line configuration analysis is the area were some of these systems have become commercialised, such as automated firewall configuration analysis, ans routing configuration analysis. These off-line passive systems have advantages over active testing systems, such as vulnerability testing, as they can be performed before policies are deployed. Another issue raised in the research more than once, was that of which interface systems had. Network and security administrators prefer a low level, textual based CLI interface over an abstract interface, such as a GUI for most tasks.

Chapter 3

Design

3.1 Introduction

From the literature review carried out in Chapter 2 it was recognised that the almost all of the systems are built around models of device configurations. Whether it be a high level language model in systems such as NPT Guttman [49], or SNMP MIBs, or using the five tuple syntax to build BDDs such as [5]. A framework is suggested in the first section of this Chapter as a way of bringing these systems together with a simple common modeling language, and providing a starting point for new systems to be created.

A significant part of the framework was designed and implemented and the second section of this Chapter focuses on this. A tool to support the network administrator with validation of low level policy configurations was chosen as it is a critical part of the framework and no such tool was found in the research literature reviewed. The tool, as part of the framework, is shown in figure 3.1. The last section in the Chapter gives a brief design overview of an application which was designed to help evaluate the validation tool.

3.2 Framework Design

3.2.1 Motivation

Other frameworks have been suggested in literature, such as Saliou et al. [81], which is further developed in [82]. This is an overall framework for security policies, as part of an integrated security process for an organisation, and based around an automatically generated network security policy. A top-down system is proposed, with the administrator creating an abstract policy model using an XML-based policy language, which can then be used to drive analysis systems, and automated low level configuration creation. Another overall security policy specification and implementation frame work is described by Damianou [83]. This again covers the entire security policy, and all enforcement devices within the network, and is based around a complex

policy specification language. These both suffer from the problem, described earlier, of the administrator tasked with creating the high level language policy specification, which has been shown to be unpopular.

The proposed framework is focused on firewall and filtering policies, which is the focus of this thesis, rather than the entire organisation's security policy, but could be possibly extended to be used for configuration management and sercurity policy management.

3.2.2 Policy Model

A framework could use a common model, which would hopefully replace, or compliment, some of the more complex models described in the literature. The XML-based model put forward by Saliou et al. [81] seems like an obvious choice to create a modeling language to describe policies. XML provides an open way of describing abstract services which are implemented in the device configurations. XML can be self validating using various built in techniques, and designed to be portable - between platforms and over the Internet - as it is text based, human readable and designed to be easy to use [84, 85]. Thus, XML could be used to create a simple language to store the abstract firewall configurations in the proposed framework.

The Lumeta system [3] added parser modules for CheckPoint firewalls and Cisco ACL configurations, so heterogeneous networks could be modeled, and so the analysis tool would be useful to a wider audience. The low level configurations are abstracted to the Lucent Managed Firewall based language used by Firmato and FANG systems [23], and their analysis query engine uses this as input. The Lucent Managed Firewall language was chosen as it is contains simple high level constructs, and because it is easy to parse. XML also has both of these attributes. One option would be to create a new modelling language, as suggested by Saliou et al., but another option which could be investigated, is to use the Netconf XML schema to store the abstract firewall configurations. It is an existing standard which is used by the major firewall vendors, and may provide simple retrieval and deployment of configurations to and from devices.

Most of the literature reviewed covers firewall modeling. If these systems create a global model of firewalling and filtering from many devices, throughout a network, the models have to have a second part, containing the network topology information [49, 69]. The proposed model, however could model more of a networks device configurations, including the topology information contained in the interface and routing configuration commands. If those could be modeled as part of the XML abstract configuration files, all the information would be available, and a separate network topology model would not be necessary.

3.2.3 Analysis Systems

Similar to the Lumeta system [69], the system to perform analysis parses the abstract XML device definitions, and builds an internal model suited to the type of analysis being performed, such as a BDD. Or current systems, like Lumeta [69], or Al-Shaer's policy conflict analysis system [5], could interface directly to the XML model to provide the analysis needed. XML parsers would need to be added, but this a fairly simple task.

Modules could be added as needed, or current systems such as reviewed in Chapter 2 could be added. New systems could be researched and added, such as a GUI visualisation system and editor, a security best practice or standards conformity checker module, a static analysis system, a security services simulator, a performance analysis engine, and a high level security policy inference engine.

3.2.4 Configuration Deployment

Netconf XML descriptions are designed to be used for configuration retrieval and deployment, so this should be simplify deployment. If a separate XML language was used, an e**X**tensible **S**tylesheet **L**anguage **T**ransformations (XSLT) system could use vendor specific e**X**tensible **S**tylesheet **L**anguage (XSL) style sheets to translate the abstract XML into low level vendor configurations [86]. This has previously been successfully implemented in systems such as Cuppens et al. [15].

Although the system is proposed for security services, this could easily be extended to include other services such as routing, and so model entire device configurations. This could then be used as the basis for an extremely powerful configuration management framework, as well as configuration testing, auditing, simulation of entire network devices and networks. An outline of the framework is shown in figure 3.1.

3.3 Firewall Policy Validation Tool Design

3.3.1 Design Motivation

Network configurations tend to be created off-line, stored centrally, and applied to devices at a later stage. Administrators do not tend to create configurations straight onto the network devices [34]. With firewall rule set development in particular, it is strongly recommended to develop solutions off-line [19, 87]. This is mainly due to firewall rule sets being complex to understand and maintain, as covered in the review in Chapter 2. A practical problem with the CLI interface, preferred by administrators, is raised by both Caldwell et al. [34] and Wool [10], in that it only displays a limited

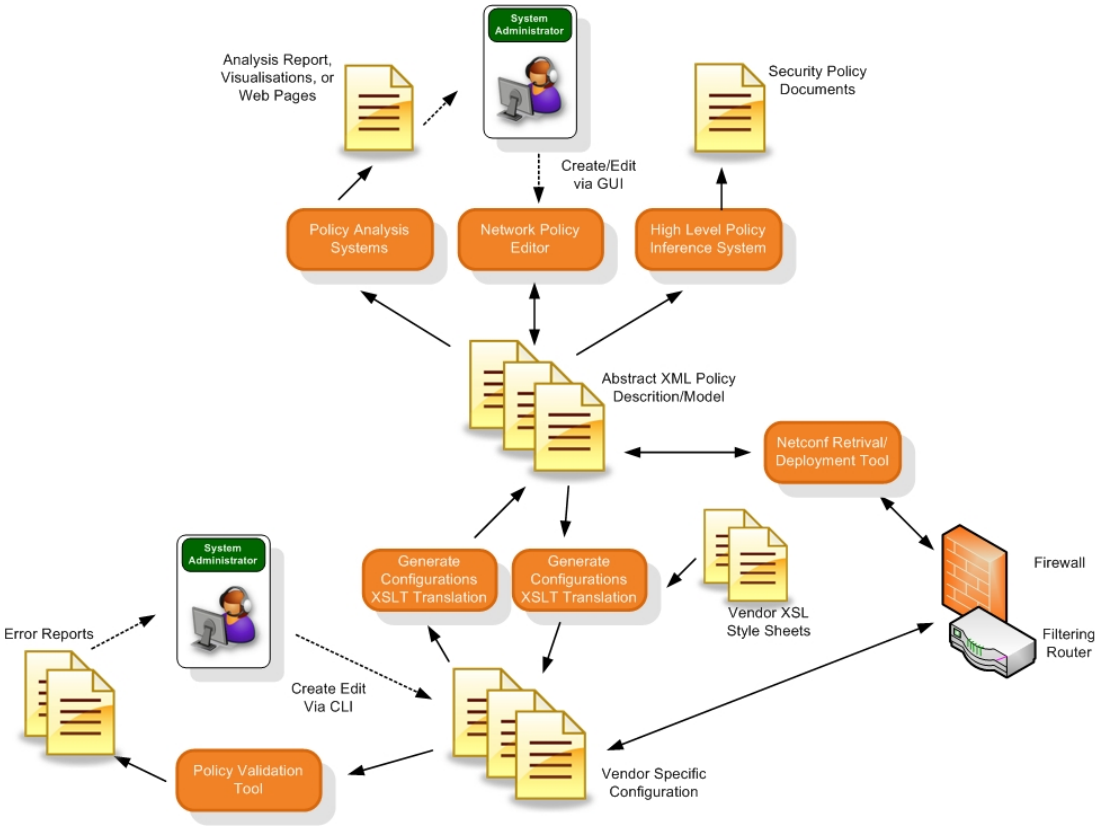

Figure 3.1 – Network Security Policy Management Framework

number of configuration statements at any one time. This is another reason that offline development and maintenance of firewall rule sets is favored.

While reviewing the research into this area, no off-line validation tools or compilers for simple text files were found to exist, which suggests the administrator has to deploy a rule set onto a network device to validate it for errors. Any tools that did parse the low level configurations, presume formatted and validated output which has come from the network devices.

The systems reviewed in section 2.4.3, which operate in the development phase of the security SDLC, typically output firewall or filtering rule sets in a vendor specific language for use on a specific device. These systems do not validate the configuration output, and this may cause the device to reject the configurations when they are deployed. A tool which could be run on the output from these systems, could provide feedback to the administrator with any problems, before the configuration was deployed to the device. The tool could be used in collaboration with systems like the one created by Hinrichs [35].

Some of the other systems reviewed imported low level configurations, such as firewall rule sets in vendor specific languages, before running offline testing or analysis on the policies. For example, systems such as the FIREMAN analysis system Yuan et al. [74]. A tool which could validate the syntax of the low level language could be

used prior to the import. If the tool could provide strict validation, such as the firewall or filtering device OS provides, new or edited configurations could be validated off-line, before being analysed by the auditing system. This would remove the need for the configuration or firewall rule set to be entered into a network device, prior to being imported into the analysis system. How the tool may integrate with these systems is shown in figure 3.1.

Some of the systems reviewed, such as FPA system Al-Shaer and Hamed [72], have no automatic importing of the policies, or configurations. These systems need an administrator to manually translate the low level rule set into a high level abstract language. When testing was done with some systems using high level languages, administrators complained that it was harder to create the policies in the abstract language, than using the low level vendor specific languages [26]. Some researchers introduced modules to parse the device configurations, but if new policies have to be created the high level language would have to be used to define them. However, this is could be automated as an extra output from the proposed tool. An output module could be added for any of the high level policy specification languages needed, including the XML based language proposed for use in the new framework. The administrators could then create the policies in the vendor low level languages that they prefer to work with.

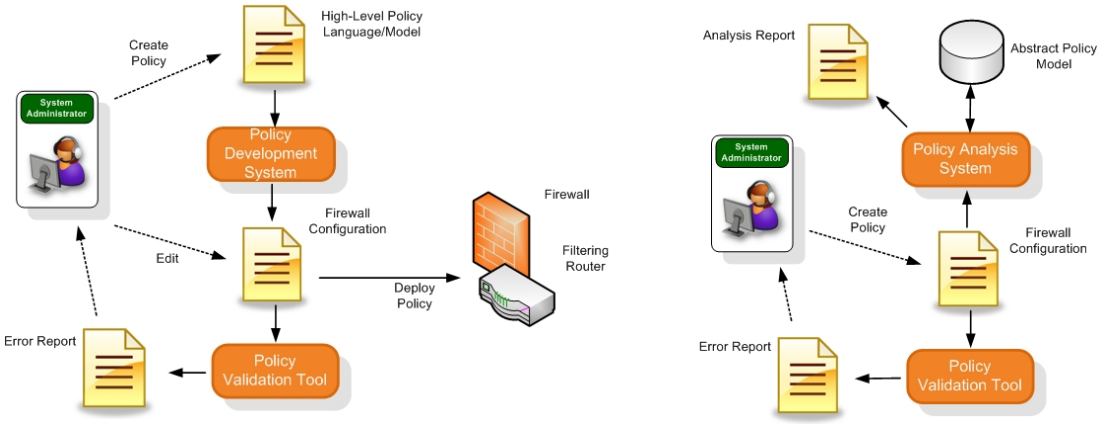

Figure 3.2 – Proposed policy validation tool integrating with existing systems.

Another motivation for the design of the tool was that it should be modular, so it could integrate with other network tools if necessary. This is something that network administrators typically require, as tasks are often automated, such as with scripting languages [44]. Multiple tools are also often used to perform one task, therefor tools should be combinable. This modularity and combination of tool use can be supported with CLI interfaces, which can be easily scripted, and textual outputs in standard formats which can be parsed and correlated [80, 88].

Another general motivation was customisation and automation of the tool. Often administrators have to deal with unpredictable situations, such as vendor software

upgrades, or configuration changes because of a new security vulnerability. Tools should be customisable, with the ability to add new modules easily. For example, if the proposed tool is typically only used to validate a single configuration, it should also be able to be automated to run over many configurations if necessary.

Another motivation relating to the reviewed research was flexibility. The tool should be designed to be flexible in its use. It should be able to be used as a stand alone tool, either from the CLI for network administrators to use, or as the back-end for another system, such as an GUI-based teaching tool for less experienced users. This could provide different user interfaces for different types of users depending on experience, which is useful as administration tools are often used by several types of end-user [80].

A tool is therefor proposed, which can validate a low level rule set for syntactic and semantic errors, and give useful feedback about any errors to the administrator. The input rule set would be in the form of a device configuration, as this is typically how administrators prefer to create firewall and filtering policies. Figure 3.2 shows how the tool might interface with some of the existing policy creation and analysis systems, which were reviewed in Chapter 2.

Cisco's ACL packet filtering and firewall policy language were chosen as the low level language to base the tool around. Cisco currently has largest market share in both filtering routers and firewalls, and ACLs are the preferred way to define filtering on these devices [3].

When networking students are learning to create and edit Cisco device OS configurations, they tend to find it difficult due to the low level language and the single line editing style. When creating even small ACLs this can cause configuration errors. Cisco recommend off-line ACL creation to students in their course literature, but they do not provide an off-line compilers for their low level languages. Students must rely on simulators or the device CLI itself, which are both using single line editors. The proposed validation tool could be used to support the teaching of these courses, and applied ACLs in general. The students can concentrate on the firewall semantics and use the tool to check the systax.

?

3.3.2 Interface

It is suggested, from the review of literature, that network administrators prefer the CLI, as an interface for working with in general. It is also suggested that in particular, for firewall configuration creation and management, the preferred method is a CLI [9, 43]. So a simple CLI with arguments is proposed as the interface for the tool. A simple argument line parser should be implemented to accept these arguments, and

provide help to the user. The arguments are listed below:

- **-c** a string which specifies the input configuration file to validate.
- **-s** a string which contains an ACL configuration.
- **-v** a boolean flag (default false) to specify verbose error description output or simple output.
- **-x** a boolean flag (default false) to specify whether to produce XML abstract configuration output.
- **-q** a boolean flag (default false) on whether to output the error report file or not.
- **-h –help** display help to the user, then exit the tool.

A CLI will also mean the tool can be scripted, which is often useful to an administrator. However, the tool should be designed to be flexible enough to be able to interface with Web or Windows GUI applications, so it could be incorporated into systems for less experienced users, such as networking students.

3.3.3 Design Overview

The validation tool functions much like a compiler in operation. A compiler reads in a source code file as input, analyses the code returning any errors to the user, then once the code is error free generates object code as output [89]. The validation tool will read in the configuration file containing the Cisco ACLs to validate, return any errors found to the user, and output the configuration with any optimisation of the code necessary. In this case, the source code and object code will be the same language, but otherwise the operation of the tool is similar to a compiler. The main task of the tool is to make sure the source configuration is error free before it is deployed onto a network device, or imported into one of the existing abstract modeling systems.

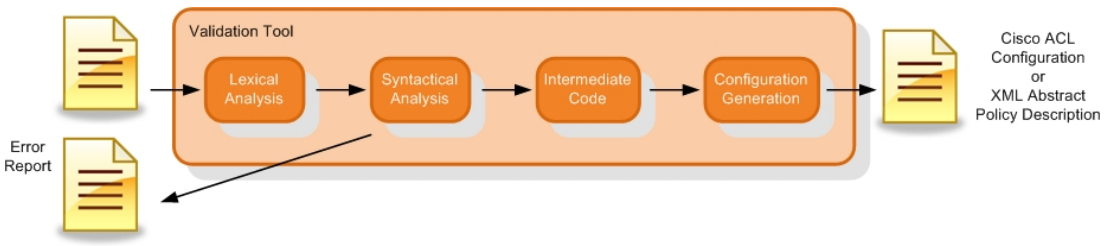

Figure 3.3 – Validation Tool Components

A compiler is defined as having five phases: Lexical analysis, Syntactic analysis, Intermediate code generation, Optimisation, and Object code generation [90]. The

validation tool is designed in a similar way, with most of the functionality we are interested in, in the first two phases. The tool should have the following components:

1. Lexical Analysis
2. Syntactic Analysis and Error Generation
3. Intermediate Cofiguration Generation and Optimisation
4. Output Configuration Generation

3.3.4 Lexical Analysis

This component of the tool breaks the source configuration into units for the syntactical analysis module to process. The units are known as tokens in compiler design teminology [90]. For maximum flexibility in the design, the module should accept a file name if input stream is to be a file, or a string if the input stream is that string, such as when passed as input directly from another application.

Listing 3.1 – Cisco Named ACL Example

```
ip access-list extended INBOUND_FROM_INTERNET
  remark Block Invalid Source Addresses From Internet
  deny ip 127.0.0.0 0.255.255.255 any log
  deny ip 0.0.0.0 0.255.255.255 any log
  deny ip 172.16.0.0 0.15.255.255 any log
  permit ip any 146.1.1.0 0.0.0.255
  permit tcp any host 146.1.2.10 eq http  ! Access to web server
  deny any any log
```

Taking the example ACL in Listing 3.1 some of the tokens are:

- **Commands:** ip access-list, remark, deny, permit
- **Keywords:** eq, host
- **Parameters:** extended, INBOUND_FROM_INTERNET, ip, tcp, any, 127.0.0.0 0.255.255.255, http

Tokenising ACLs, in Cisco's configuration language, is fairly straightforward as all of the commands and parameters are separated by white space. The lexical analysis module should also perform several other tasks. Comments, which are marked by the use of a '!' character as shown in the example, should be removed. The tokens passed to the syntactic analysis module should have case converted to lower case, as Cisco's configuration languages are not case sensitive. Any string parameters, such as ACL name in the example, should be maintained in the output configuration, and

in the error report output to the user. White space should be removed, as it has no relevance in Cisco's configuration languages.

Tokens in most compiler systems are defined using 'regular expressions'. A regular expression is a compact notation that can describe a range of characters. This regular expression, or 'pattern', can then be matched against strings, which is known as pattern matching [91]. In the context of the validation tool, regular expressions can be used to specify a range of characters which are valid values for a particular token.

Listing 3.2 – Cisco ACL access-list (IP extended) command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } protocol source source-wildcard destination destination-wildcard [precedence
    precedence] [tos tos] [log | log-input] [time-range time-range-name] [fragments]
```

The 'access-list' command, shown in listing 3.2, is a part of the command definition, which is taken from the Cisco **I**nternetwork **O**perating **S**ystem (IOS) Command Reference - Release 12.4 [92]. The 'protocol' parameter can have a value of IP protocol number 0-255, or it can be one of the IP protocol keywords `eigrp, gre, icmp, igmp, ip, ipinip, nos, ospf, pim, tcp, or udp`. This token can be defined as the following regular expression:

```
"^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]|ahp|eigrp|esp|gre|icmp|igmp|ip|ipinip|nos|
ospf|pcp|pim|tcp|udp)$"
```

Each token to be parsed can have its valid values defined by a regular expression in this way. Tokens can then be matched against these regular expressions to check for valid token values. In this way conditional statements in the implemented code can be reduced significantly. The tokens can be stored together, away from the workings of the code itself, meaning changes can be easily made to to the token values, without having to change any code.

3.3.5 Syntactic Analysis

The syntactic analysis module, or parser, analyses the structure of the input source code, and the commands within the structure.

Listing 3.3 – Cisco Extended ACL Example

```
access-list 101 permit icmp 192.168.1.0 0.0.0.255 host 192.168.1.1 echo-reply
```

The parser should recognise that, in the example in listing 3.3, the ACL is of type 'extended' and that the filtering fields 'host' and '192.168.1.1' mean that the destination range of network addresses is a single host address. The parser should recognise the

grammar of the source language, allowing it to then validate the tokens are in the correct order, and of the correct type and value.

Most compilers deal with programming languages which are not context sensitive. These languages can be parsed recursively and are not state, or context, dependent. Cisco's configuration languages are context sensitive, and commands and parameters can mean different things depending on the context. Due to this a manual parser was designed, rather than using a parser generation language - such as using lex & yacc [93] which was considered - to generate the parser code. No recursion is necessary, so only one pass through the code is needed. This may be advantageous as single pass parsers tend to have the fastest performance [90].

To parse of the context sensitive Cisco commands, state will need to be stored about the current context that the parser is dealing with. Different commands can be used, depending on the state, or 'command mode' as Cisco call it. The state for ACL commands to be entered, such as 'access-list' and 'ip access-list' is 'global configuration mode'. If the 'ip access-list' command is parsed, the state changes to 'access list configuration mode' and commands such as 'permit' and 'deny' can then be parsed. These 'permit' and 'deny' commands are not accepted in 'global configuration mode'. The change in mode, or state, can be seen in the example ACL in listing 3.1.

To define the grammer of ACL configurations the Cisco documentation has been used. The access-list commands, defined in the Cisco IOS Command Reference Release 12.4 [92], should be used as a base for the parser design. This is the OS which is used on Cisco routers. There are five commands used to define ACLs in this version of Cisco's device OS:

- access-list
- ip access-list
- permit
- deny
- remark

Within these commands, 14 different type of ACL exist, with two common types used more than any of the others. The two common types, which were defined as part of the design, are 'Standard' and 'Extended' ACLs. The standard type filters only on source-based fields of the, layer 3 OSI model, IP packet headers. The extended ACL type can filter on headers fields from layers 3 and 4, and can filter more specific traffic based on source and destination network addresses, and specified network services, ports and flags. The syntax for the standard ACL, as defined by the Cisco documentation is shown in listing 3.4.

Listing 3.4 – Cisco ACL access-list (IP standard) command definition [92]

```
access-list access-list-number {deny | permit} source [source-wildcard] [log]
```

The command is 'access-list' and the 'access-list-number' being in the range 100-199, or 2000-2699 specifies that the command is a standard ACL. The filtering rule action is specified by the keywords 'permit' or 'deny'. The filtering fields are 'source' network address, and 'source-wildcard' which can specify a range of addresses, and an optional 'log' keyword. Each of these tokens should have a regular expressions created to define their valid values, and the grammer of the commands, keywords and filtering fields should be built into the appropriate code modules, for each of the commands and ACL types. The algorithm for the standard ACL type should validate the OSI layer 3 filtering fields described in Listing 3.4. The extended ACL algorithm should validate the OSI layers 3 and 4 filtering fields, as specified in Listings 3.5 - 3.9.

Some discrepancies were noticed in the Cisco documentation, as the design was being carried out. Such as the 'dscp' filtering fields are not defined in the definition shown in Listing 3.5, but are shown in some other on-line documentation. After checking this on a Cisco router, it was found that these and some other missing filtering options are missing from the latest Cisco documentation. These were added to the tool, and as much of the validation algorithms as could be, were checked on actual devices, as the design and implementation took place.

An object oriented approach to the design of the tool was taken. The designed objects are responsible for the representation of their data and the validation of parsing that data. The proposed object model is shown in 3.4 (some of the filtering field objects are missed out for simplicity). Referring back to listing 3.4, the grammer processing for the Standard ACL should be provided in the 'Standard ACL' class shown in the figure. The functionality associated with the filtering fields is provided via the 'Action', 'Src IP Address Range', and 'Log' classes. This encapsulation within the objects provides reuse and extensibility. The 'action' class is reused as an attribute of the 'Extended ACL' class. Extensibility is also given, by the ability to add new services - as new objects- with the minimum of changes to the rest of the object model.

The model shows the ease with which the tool could be extended to deal with other commands within the Cisco OS, or to include other platforms. The 'ACL Commands' class deals with all commands in the OS which relate to ACLs, and their parsing and validation. It decides on which type of commands are being parsed, and uses the appropriate classes to process those ACLs. If a new service, such as routing, was to be validated, a new class could be added to deal with those commands, and new classes would be created to provide the functionality for the parameters within those commands.

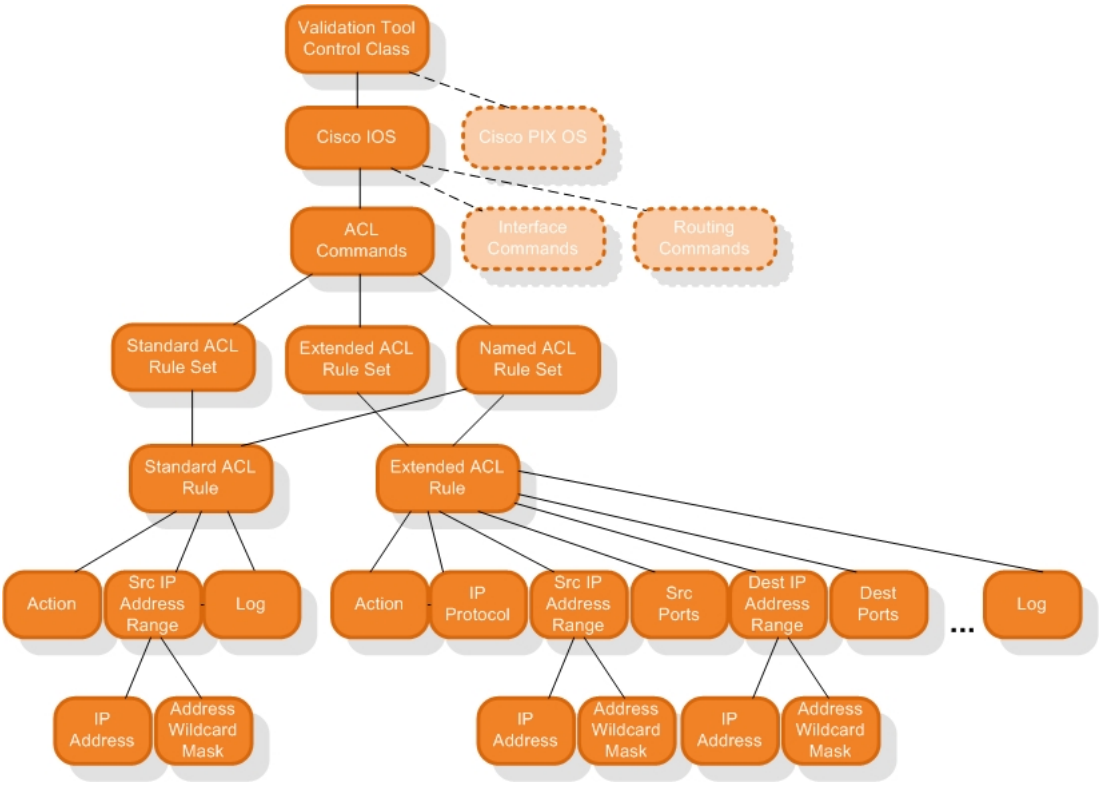

Figure 3.4 – Validation Tool Object Model

The system is designed to be flexible and extensible, as the research reviewed showed that these type of network device languages are constantly changing [75].

Taken from the Cisco documentation, the definitions of extended ACLs, based on the IP Protocol field, are shown in listings 3.5 - 3.9. These were used to design the grammer of the 'Extended ACL' class.

Listing 3.5 – Cisco extended IP ACL access-list command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } protocol source source-wildcard destination destination-wildcard [precedence
    precedence] [tos tos] [log | log-input] [time-range time-range-name] [fragments]
```

Listing 3.6 – Cisco extended ICMP ACL access-list command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } icmp source source-wildcard destination destination-wildcard [icmp-type [icmp-
    code] | icmp-message] [precedence precedence] [tos tos] [log | log-input] [time-
    range time-range-name] [fragments]
```

Listing 3.7 – Cisco extended IGMP ACL access-list command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } igmp source source-wildcard destination destination-wildcard [igmp-type] [
    precedence precedence] [tos tos] [log | log-input] [time-range time-range-name] [
    fragments]
```

Listing 3.8 – Cisco extended TCP ACL access-list command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } tcp source source-wildcard [operator [port]] destination destination-wildcard [
    operator [port]] [established] [precedence precedence] [tos tos] [log | log-input]
     [time-range time-range-name] [fragments]
```

Listing 3.9 – Cisco extended UDP ACL access-list command definition [92]

```
access-list access-list-number [dynamic dynamic-name [timeout minutes]] {deny | permit
    } udp source source-wildcard [operator [port]] destination destination-wildcard [
    operator [port]] [precedence precedence] [tos tos] [log | log-input] [time-range
    time-range-name] [fragments]
```

This component also should keep track of errors found in the configuration and write these out to an error log file, or display an error report back to the command line if the -q quiet parameter was specified. If the output is to an error log file, and the -v verbose flag was set, then display a detailed error report, including a description of the error, the configuration line, and the field the error was detected in highlighted underneath - using the '^' symbol. This is similar to the ACL error reporting that the Cisco command line produces.

3.3.6 Intermediate Code Generation and Optimization

This component creates the internal representation of the configuration to be output. In the case of ACLs this is a filtering rule set, containing a number of filter rules representing the ACL lines. This code stage is only necessary in the validation tools case, as the output configuration could be the same as the input configuration if no errors are found. if some configuration optimisation was to be done it would be in this component. The Cisco device OSs sometimes store the configurations in a slightly different manor to how they were input through the CLI, such as changing wildcard masks to shorthand keywords (192.168.1.1 0.0.0.0 to 'host' 192.168.1.1). As a future enhancement, this type of optimisation could be performed here for completeness.

3.3.7 Output Configuration Generation

The output configuration component generates the valid configuration, ready to be deployed on a device, or imported into the framework.

3.4 Evaluation Tool Design

An evaluation tool was designed create experimental test data files for the validation tool. Each file would contain ACL rule sets. A GUI was specified, allowing the user to select a range of ACL rule set sizes, number of errors in each rule set, and the type of rule set to create. The rule sets can range from 100 rules up to 2 million rules. The rule set files would contain rules which would be created automatically by the evaluation tool, with random valid values. The errors would be automatically created in random parts of the rules, and could be specified as a percentage of the total rules in a rule set. The types of ACL rules sets would be standard, extended, named, or a random mixture of different types in a test file.

The operational process would consist of firstly creating the rules, as described above, and then running experiments on the created files. The experiments, would consist of the user selecting some of the automatically generated test files, and specifying a number of runs that would take place for each file. The tool would then run the validation tool, with each specified file, a number of times, while recording performance metricsm such as time taken to process each file. These metrics would be collected and averaged, and an output file created for each experiment. The process is shown in Figure 3.5.

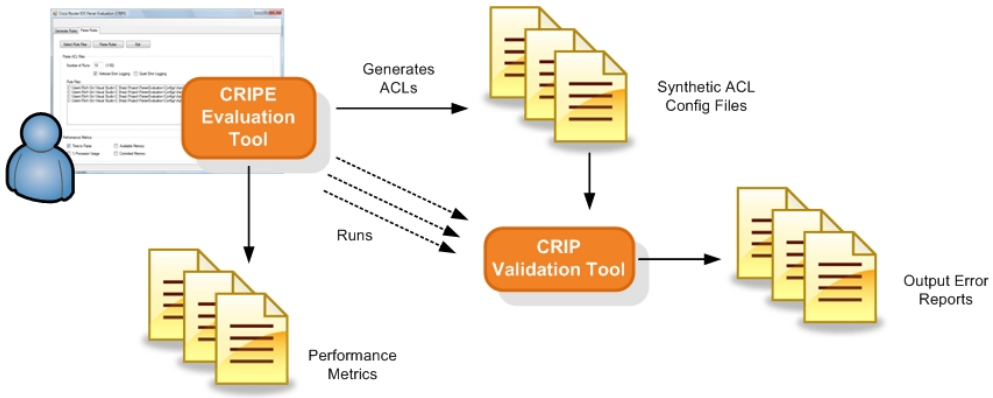

Figure 3.5 – CRIPE evaluation tool - generates synthetic ACLs, and runs CRIP tool

3.5 Design Conclusions

An overview for a simple framework to integrate existing policy-based management systems, with a central XML-based modeling language, is proposed. An XML-based model would be open rather than the closed models such as the Cisco, or Algosec systems are based on. The framework would be based on abstracting the network device configurations, into XML-based language descriptions. The XML-based Netconf protocol could possibly be used as the language, which would provide retrieval and

deployment functionality. This type of policy description language may not be powerful enough to describe entire security policies however, such as user authentication and access control policies.

The validation tool was designed to integrate into the proposed framework, and compliment other existing tools. It has been designed with a pragmatic outlook, catering for the needs of the network and security administrator. The modularity of the design is important, due to the fast changing nature of the network device languages being processed. A flexible tool was designed, due to the possible needs of different types of users. A low level scriptable text based interface is provided for administrators, but the tool could interface with GUI applications for less experienced users. The tool was designed to support the administrator, by allowing them to use the languages and tools they prefer, and not introducing new untrusted languages or levels of abstraction. The CLI was left as the main interface, allowing scripting of the tool, and combination with other tools.

Chapter 4

Implementation

4.1 Introduction

This chapter describes the implementation of the Cisco Router IOS ACL Parser and validation tool, CRIP, from the design in chapter 3. CRIP was designed as a tool to assist the network administrator when creating Cisco ACL policies. A prototype system was developed, implementing the majority of the design from Section 3.3. The resulting tool was fully tested through the CLI, as well as from custom Web and Windows GUI-based test applications.

The second part of the chapter details the implementation of a performance evaluation harness application which has been called CRIPE. The CRIPE tool can be used to create various input files to use with the CRIP tool. The evaluation application can also automate the process of running the CRIP tool with a number of different input files, over a number of runs, to collect performance metric data.

To create the applications described in this chapter, several resources were used. The IDE used was Microsoft Visual Studio 2009 - Professional Edition. The main programming language used was Microsoft Visual C# 2008, along with the .NET Runtime Framework - Version 3.5 SP1. Editing and working with test data was done mostly with a Windows version of the `vi` editor - `vim`. Also some Windows versions of unix command line utilities - `find`, `grep`, `diff` and `tail` were used. These were extremely useful in manipulating configuration and error output files.

The use of the C# programming language, with the .NET runtime environment provides an extremely powerful object oriented platform. This provided rapid implementation of both the tool and the GUI test applications. Another major benefit was access to many object libraries for access to functionality such as windows GUI handling, file handling, string handling, and regular expressions.

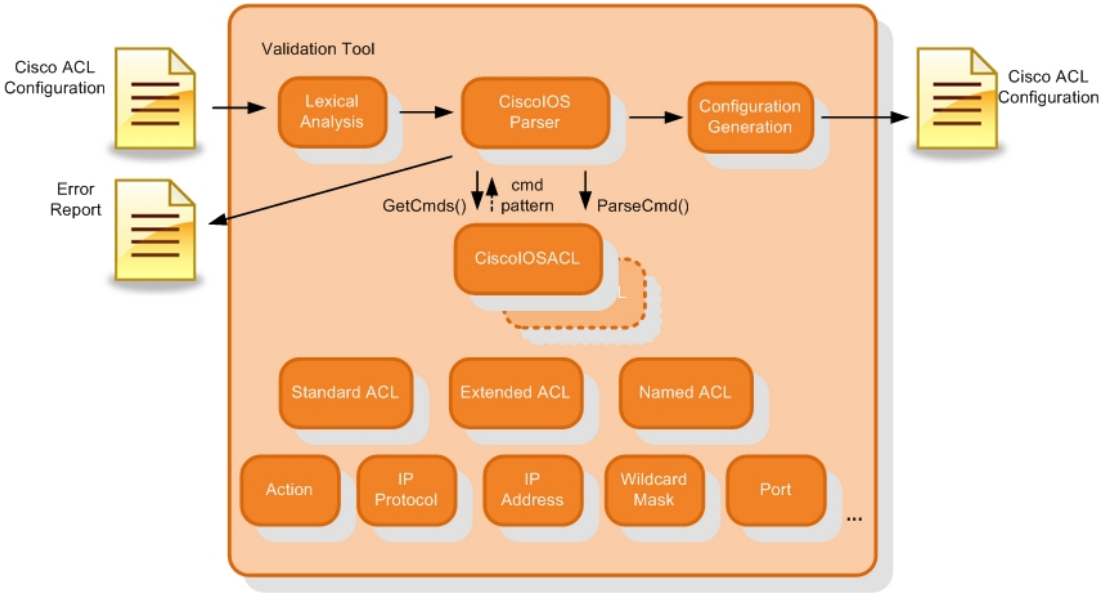

Figure 4.1 – CRIP Tool Implementation

4.2 CRIP Tool Implementation

The tool was implemented as a C# console application, to provide a CLI interface, but the main validation and parsing functionality was encapsulated in C# classes. This meant the methods of the classes could be utilised directly by Windows or Web applications, if a GUI was required.

4.2.1 Interface

The interface described in section 3.3.2 was implemented using some code to parse the CLI arguments. A snippet of the code is shown in listing 4.1. The static `Split()` method of the `Regex`, regular expression class, is used to split each argument into parts. This uses the regular expression capabilities provided by .NET, from the `System.Text.RegularExpressions` library. Depending on whether an argument is a parameter or a value, the switch statement either stores the parameter, or stores the value if the previous argument was a parameter, or stores the previous boolean parameter with the value 'true'.

Listing 4.1 – CRIP tool argument parsing code snippet, and the use of the regular expressions library

```
1 Using System.Text.RegularExpressions
2 ...
3 // Parse Arguments.
4 string[] sParts;
5 foreach (string arg in args)
6 {
7     // Split each cmd line arg.
```

```
8       sParts = Regex.Split(arg, @"^-{1,2}", RegexOptions.IgnoreCase);
9       switch (sParts.Length)
10      {
11          case 1:
12              // A Parameter Value.
13              if (sParam != null)
14              {
15                  // Set param value.
16                  if (!Params.ContainsKey(sParam))
17                      Params.Add(sParam, sParts[0]);
18                  sParam = null;
19              }
20              break;
21          case 2:
22              // A Parameter.
23              if (sParam != null)
24              {
25                  // Previous param has no value, (bool param) set it
26                  // to 'true'
27                  if (!Params.ContainsKey(sParam))
28                      Params.Add(sParam, "true");
29              }
30              sParam = sParts[1];
31              break;
32      }
33  }
34  // Previous param has no value, (bool params) set it
35  // to 'true'
36  if (sParam != null)
37  {
38      if (!Params.ContainsKey(sParam))
39          Params.Add(sParam, "true");
40  }
```

The next code snippet shows the next part of the code, were the console application is setting the parameters for the CRIP tool, from the parsed list. It simply ignores parameters which are not implemented yet. Future development should include a help parameter `-h`, as well as some additional validation to reject incorrect parameters.

Listing 4.2 – CRIP tool argument parsing code snippet

```
1   string configFile = "";
2   if (Params.ContainsKey("c"))
3       configFile = Params["c"];
4   else {
5       Console.WriteLine("No file to parse...");
6       Environment.Exit(1);
7   }
8   bool bVerbose = false;
9   if (Params.ContainsKey("v"))
10  {
```

```
11      if (Params["v"] == "true")
12          bVerbose = true;
13  }
14  bool bQuiet = false;
15  if (Params.ContainsKey("q"))
16  {
17      if (Params["q"] == "true")
18          bQuiet = true;
19  }
```

The CRIP tool running from the CLI is shown in figure 4.2. The tool has been run three times, with the `-c` argument to specify the same input configuration file. The second time the tool is run with the `-v` argument for verbose error reporting, and the third time it is passed the `-q` for quiet error logging. This is a similar style interface to typical text-based tools favored by system administrators [43].

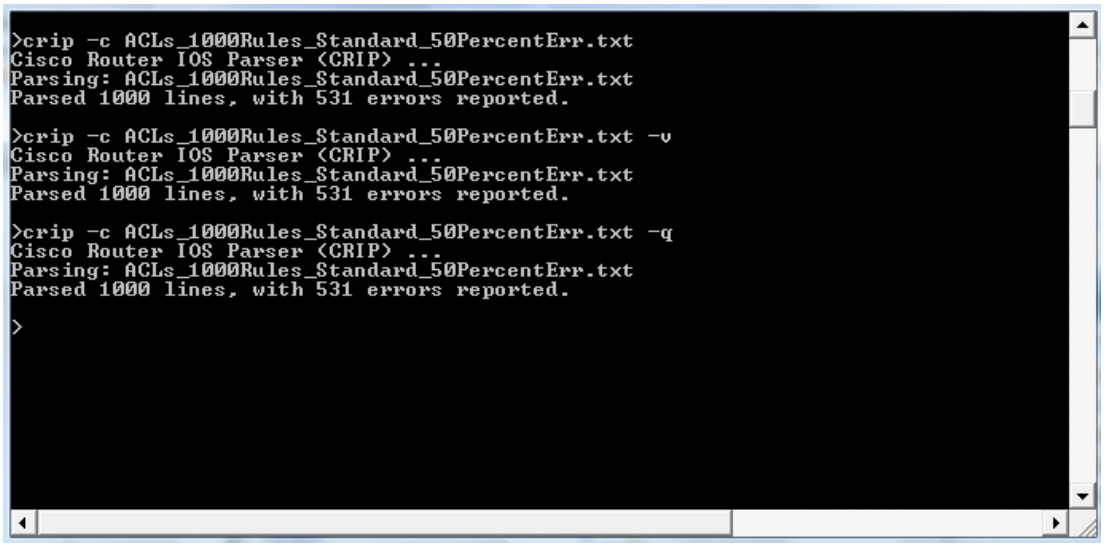

Figure 4.2 – CRIP Tool CLI Interface

4.2.2 Lexical Analysis

The lexical analysis module, implemented in the `Lexer` class, reads the entire configuration file into an array. During initial unit testing, it was shown the array can cope with over a million lines of configuration, so this is seemed reasonable for the purpose of validating ACLs. The literature reviewed discusses firewalls with tens of thousands of rules, but not millions [22, 35]. An alternative would be to read in a buffer of a maybe a thousand lines at a time, if larger configuration files were to be processed. The regular expressions RegEx class, Split() method, was used to split the configuration input file, into configuration lines.

Listing 4.3 – Lexer reading input configuration file into memory code snippet

```
1  public Lexer(
```

```
    string configFile // file to process.
    )
{
    if (configFile == null || configFile == "")
        throw new ArgumentException("[CiscoConfigParser.Lexer] null or blank
            parameter", "configFile");
    try {
        // Open input stream.
        this.config = new StreamReader( configFile );
        // Read in config, split into lines array.
        // TBA: "\r\n" is DOS EOL, need to add "\n" Unix, and "\r" Mac
        this.lines = Regex.Split( config.ReadToEnd(), @"\r\n" );
        linesLength = lines==null ? 0 : lines.Length-1;
        linesPos = -1;
        // Close input stream
        config.Close();
    }
}
```

Lexical analysis removes any whitespace, and comments from the input configuration lines as described in design Section 3.3.4. As specified, the configuration line returned to the Parser module is also converted to lower case. Listing 4.4 shows the regular expressions used to match comment lines and white space lines, and the code used to return a configuration line from the cached input configuration file.

Listing 4.4 – Lexer returning a configuration line code snippet

```
// Regex Patterns.
const string COMMENTLINE = @"^\s*!.*$";
const string WHITESPACELINE = @"^\s*$";
...
public string ReadLine()
{
    string line;

    if ( lines.Length == 0 )
        return null; // no lines.
    if ( linesPos == linesLength )
        return null; // end of config.
    // return next config line.
    while ( ++linesPos < linesLength )
    {
        line = lines[linesPos];
        // Skip comment lines or empty lines in the input stream.
        if ( Regex.IsMatch(line, COMMENTLINE) | Regex.IsMatch(line, WHITESPACELINE
            ) )
            continue;
        else {
            // remove comments at end of line
            line = Regex.Replace( line, @"\s!+.*$", "" );
    // remove whitespace at end of line
```

```
24      //line = Regex.Replace( line, @"\s*$", "" );
25      // Pass lowercase config line back to parser.
26      curLine = line.Trim().ToLower();
27      return curLine;
28          }
29      }
30      // no line to return
31      return null;
32  }
```

Parsing Tokens

The parsing of the tokens was carried out using regular expressions. The `IsMatch()` static method of the `RegEx` class is used to match the token being parsed, against the regular expression pattern. Listing 4.5 shows the code which parses the IP Protocol filtering field. The `IsMatch` method matches the current token against the regular expression for the protocol, returning a boolean true if there is a match. The static method is used, as opposed to the overhead of creating an instance of a `RegEx` object and using the `Match()` method.

Listing 4.5 – Lexer returning a configuration line code snippet

```
1  // protocol 0-255|named protocol
2  const string PROTOCOLS      = @"^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]|ahp|eigrp|
       esp|gre|icmp|igmp|ip|ipinip|nos|ospf|pcp|pim|tcp|udp)$";
3  ...
4  private int ParseProtocol( string[] tokens, ref int idx )
5  {
6      idx++; // next token
7      if ( idx == tokens.Length )
8      {
9          traceLog.WriteLine("\n[ParseProtocol] % Incomplete Command - Protocol
               Expected" );
10         errorLog.WriteLine("% Incomplete Command - Protocol Expected", lexer, "EOL
               " , bVerbose );
11         return NO_TOKENS;
12     }
13     string protocol = tokens[idx];
14     //
15     // Name or number of an IP Protocol.
16     //
17     if ( Regex.IsMatch(protocol, PROTOCOLS) )
18     {
19         this.Protocol = protocol;
20     }
21     else
22     {
23         traceLog.WriteLine("[ACL.ParseProtocol] % Invalid Input Detected: IP
               Protocol or <0-255> expected");
```

```
24          errorLog.WriteLine("% Invalid Input Detected: IP Protocol or <0-255>
                expected", lexer, protocol, bVerbose );
25          return INVALID;
26      }
27      // Valid Protocol.
28      traceLog.WriteLine("[ACL.ParseProtocol] valid Protocol: " + protocol);
29      return SUCCESS;
30  }
```

4.2.3 Syntactic Analysis

The syntactic analysis is implemented mainly in the `CiscoIOSACL` class. It encapsulates the functionality for dealing with Cisco ACL commands, within the Cisco IOS. The `ParseCmd()` method takes a command line from the calling `CiscoIOS.Parse()` method and provides the grammer analysis, for Cisco ACL commands, described in design Section 3.3.5.

Commands and State

The code snippet shown in 4.6 shows part of the `CiscoIOS` class `Parse()` method. On line 7 the `CiscoIOSACL` class, `GetCmds()` method, is called to return a regular expression string pattern, which is then matched against the current input configuration line, the 'line' variable. In this way the `CiscoIOSACL` module can announce which commands it has functionality to deal with, via the regular expressions patterns shown in Listing 4.7. Other modules can be plugged in by simply adding lines to this `Parse()` method to call the `GetCmds()` methods of the added modules. No commands are hard coded anywhere using this type of interface. If a new class 'CiscoIOSRouting', to provide Cisco IOS Routing command validation, was added - such as suggested in figure 3.4 - another if statement calling CiscoIOSRouting.GetCmds() would simply be added. In this way classes can be inserted, to add validation of other services such as routing commands. Modules could be added to provide functionality for all services which make up the configuration of the OS, thus providing the same functionality as the proprietary CLI. This was inspired by a technique used for plug-in modules in the FIRECROCODILE firewall parsing system Lehmann et al. [94].

Listing 4.6 – Cisco IOS Command Parsing code snippet - CiscoIOS.Parse() mothod

```
1  // Get the next line of the config.
2  while ((line = lexer.ReadLine()) != null)
3  {
4  // Parse IOS commands.
5
6  // Check if class can handle command, if so call handler to Parse cmd line.
7  if (Regex.IsMatch(line, CiscoIOSACL.GetCmds()))
```

```
{
    CiscoIOSACL ACL = new CiscoIOSACL(lexer, errorLog, traceLog, bVerbose);
    if (ACL.ParseCmd(ref cmdMode) != SUCCESS)
        iErrors++;
}
else {
    // Command not implemented.
    errorLog.WriteLine('%UNKNOWNCOMMAND', lexer, bVerbose);
    iErrors++;
}
iLinesParsed++;
}
```

Listing 4.7 – Cisco IOS Command Parsing code snippet - CiscoIOSACL.GetCmds() method

```
// Commands class can parse.
const string ACCESS_LIST    = @"^access-list\s+";
const string IP_ACCESS_LIST = @"^ip access-list\s+";
const string PERMIT         = @"^permit\s+";
const string DENY           = @"^deny\s+";
const string REMARK         = @"^remark\s+";
const string COMMANDS       = ACCESS_LIST+"|"+IP_ACCESS_LIST+"|"+PERMIT+"|"+DENY;
...
public static string GetCmds()
{
return COMMANDS;
}
```

ACL Types

The processing of the grammar for the various ACL rule set types is performed in the `CiscoIOSACL` class, using the `ParseStdACL()`, `ParseExtACL()`, `ParseNamedACL()` methods. These methods deal with the rule set header data, such as ACL number or name. The rules are then parsed using shared functionality, implemented in the StdACLRule and ExtACLRule ACL modules. Named ACLs can use either functionality as they can contain standard or extended rules. The method implementing the syntactic analysis for a standard rule, calls the methods to parse the appropriate filtering fields: Action, IP Address, IP Wildcard Mask, and Log (defined in Listing 3.4). The common filtering field modules are shared between the standard and extended ACL modules. The functionality is encapsulated within the moduels, and no code should be repeated.

Error Output Implementation

Error output is implemented in the `ErrorLog` class. When an error in the configuration is detected by the parser modules, the ErrorOutput object is used to store information about the errors. This class is then used to write the error report output file once the parsing is complete.

Listing 4.8 – Cisco IOS error output generation code snippet - ErrorLog.WriteLine() method

```
1   public void WriteLine( string errorMsg, Lexer lexer, string token, bool
        bVerbose )
2   {
3       errorCnt++;
4       if (this.bQuiet)
5           return;
6       string sErrorNo = errorCnt.ToString();
7       if (bVerbose == true) {
8           errors.Add(sErrorNo + ". " + "Error at line " + lexer.inputLineNo
                + " - " + errorMsg);
9           errors.Add(lexer.inputLine);
10          if (token == "EOL" )
11              errors.Add(padPointer(lexer.line, token));
12          else
13              errors.Add(padPointer(lexer.inputLine, token));
14      }
15      else {
16          errors.Add(sErrorNo + "." + "\tError at line " + lexer.inputLineNo
                + "\t" + errorMsg
17                      + ". '" + lexer.inputLine + "'");
18      }
19  }
```

The `PadPointer()` method is used if verbose error reporting is required. It generates a '^' symbol under the position of the token in error. The output for each error is three lines. The configuration line with the error, the line generated by `PadPointer()` with a '^' symbol underneath the token in error, and an error message line detailing the problem with the token. This gives the administrator a familiar system of describing errors within ACL rules, as this is similar to the output from Cisco filtering device CLIs.

Listing 4.9 – Cisco IOS Error output generation code snippet - ErrorLog.PadPointer() mothod

```
1   private string padPointer( string line, string token )
2   {
3       string[] tokens = Lexer.GetTokens(line);
4       line = line.ToLower();
5       string s = "^", padded  = "^";
6       token.Trim();
7       try {
```

```
8                if (token != "") {
9                    if (token == "EOL")
10                       padded = s.PadLeft(line.Length + 2);
11                   else {
12                           int idx = line.IndexOf(token);
13                       if (idx < 0)
14                               idx = 1;
15                       padded = s.PadLeft(1 + idx);
16                   }
17               }
18           }
19           catch {}
20           return(padded);
21       }
```

```
access-list 2285 XXX 17 host 134.148.165.29 host 121.142.194.152  ! INVALID action
access-list 146 deny 6 204.98.27.195 62.90.185.200 XXX 59907 144.109.37.73 156.121.226.25  ! INVALID srcPort
access-list 2472 permit 240 any 128.42.232.35 120.224.222.82
access-list 2305 deny icmp host 224.148.135.58 any dscp 32
access-list 181 deny tcp any any
access-list 2169 XXX 17 154.178.22.78 126.83.109.8 116.77.25.121 76.8.1.6  ! INVALID action
access-list 121 permit 6 109.115.110.101 129.136.27.198 207.149.247.212 216.167.47.218
access-list 2269 permit udp any any eq XXX  ! INVALID destPort
access-list 185 permit tcp host 237.86.23.93 66.114.180.83 11.81.204.200
access-list 2198 permit 6 any 117.204.238.23 1.47.191.108 eq pop2
access-list 2312 deny tcp 30.153.240.251 125.83.9.217 49.24.92.122 61.210.173.29
access-list 100 permit -1 host 242.226.11.199 host 240.191.18.95  ! INVALID protocol
access-list 2358 deny 6 219.186.128.136 215.66.19.80 7.103.252.90 105.151.50.10 lt 4624 established
access-list 2620 permit tcp 131.240.92.220 111.98.233.234 XXX discard any log-input  ! INVALID srcPort
access-list 178 XXX 6 any 143.205.108.132 139.119.48.54  ! INVALID action
access-list 103 permit tcp 171.28.250.65 143.38.25.221 any
access-list 2114 permit udp any XXX  ! INVALID destIP
access-list 159 deny tcp 170.129.185.97 163.221.221.90 39.93.215.129 199.101.154.140
access-list 128 deny tcp 219.69.96.100 60.139.209.206 host 148.23.171.109
access-list 148 permit XXX 65.10.61.1 243.137.85.43 host 195.109.165.197  ! INVALID protocol
access-list 2332 deny 6 any 100.54.254.23 191.121.223.1 dscp cs5
access-list 2105 permit pcp any any
access-list 98 permit 6 host 30.86.99.251 host 87.100.88.81  ! INVALID aclId
access-list 126 deny 17 218.127.88.97 172.191.29.15 host 249.198.97.154 XXX 28656  ! INVALID destPort
access-list 2325 deny icmp 68.229.36.12 178.22.253.184 host XXX.1.79.53  ! INVALID destIP
access-list 2241 deny 17 host 232.212.43.253 100.192.173.109 97.207.217.42 XXX echo  ! INVALID destPort
                                                                                                30,103        0%
```

Figure 4.3 – CRIP Tool ACLs Input Configuration

```
Parsed ACLs_1000Rules_Extended_50PercentErr.txt - 1000 lines, with 506 errors detected.

1.      Error at line 1 % Invalid Input Detected . 'access-list 2104 permit tcp host 174.99.210.113 130.117.146.22 194.173.204.41 XXX  ! INVALID log
'
2.      Error at line 3 % Invalid Input Detected - IP Address. 'access-list 2324 deny tcp 113.144.221.105 16.186.75.134 XXX.1.185.141 117.83.72.146
 ! INVALID destIP'
3.      Error at line 5 % Invalid Input Detected: {permit|deny} expected. 'access-list 2285 XXX 17 host 134.148.165.29 host 121.142.194.152  ! INVAL
ID action'
4.      Error at line 6 % Invalid Input Detected - IP Address. 'access-list 146 deny 6 204.98.27.195 62.90.185.200 XXX 59907 144.109.37.73 156.121.2
26.25  ! INVALID srcPort'
5.      Error at line 10        % Invalid Input Detected: {permit|deny} expected. 'access-list 2169 XXX 17 154.178.22.78 126.83.109.8 116.77.25.121
76.8.1.6  ! INVALID action'
6.      Error at line 12        % Invalid Input Detected - Port. 'access-list 2269 permit udp any any eq XXX  ! INVALID destPort'
7.      Error at line 16        % Invalid Input Detected: IP Protocol or <0-255> expected. 'access-list 100 permit -1 host 242.226.11.199 host 240.1
91.18.95  ! INVALID protocol'
8.      Error at line 18        % Invalid Input Detected - IP Address. 'access-list 2620 permit tcp 131.240.92.220 111.98.233.234 XXX discard any lo
g-input  ! INVALID srcPort'
9.      Error at line 19        % Invalid Input Detected: {permit|deny} expected. 'access-list 178 XXX 6 any 143.205.108.132 139.119.48.54  ! INVALI
D action'
10.     Error at line 21        % Invalid Input Detected - IP Address. 'access-list 2114 permit udp any XXX  ! INVALID destIP'
11.     Error at line 24        % Invalid Input Detected: IP Protocol or <0-255> expected. 'access-list 148 permit XXX 65.10.61.1 243.137.85.43 host
 195.109.165.197  ! INVALID protocol'
12.     Error at line 27        % Invalid Input Detected - Source IP . 'access-list 98 permit 6 host 30.86.99.251 host 87.100.88.81  ! INVALID aclId
'
13.     Error at line 28        % Invalid Input Detected . 'access-list 126 deny 17 218.127.88.97 172.191.29.15 host 249.198.97.154 XXX 28656  ! INV
ALID destPort'
                                                                                                14,54-65      Top
```

Figure 4.4 – CRIP Tool Error Output - Error Reporting

```
Parsed ACLs_1000Rules_Extended_50PercentErr.txt - 1000 lines, with 506 errors detected.

1. Error at line 1 - % Invalid Input Detected
access-list 2104 permit tcp host 174.99.210.113 130.117.146.22 194.173.204.41 XXX  ! INVALID log
                                                                              ^

2. Error at line 3 - % Invalid Input Detected - IP Address
access-list 2324 deny tcp 113.144.221.105 16.186.75.134 XXX.1.185.141 117.83.72.146  ! INVALID destIP
                                                         ^

3. Error at line 5 - % Invalid Input Detected: {permit|deny} expected
access-list 2285 XXX 17 host 134.148.165.29 host 121.142.194.152  ! INVALID action
                 ^

4. Error at line 6 - % Invalid Input Detected - IP Address
access-list 146 deny 6 204.98.27.195 62.90.185.200 XXX 59907 144.109.37.73 156.121.226.25  ! INVALID srcPort
                                                   ^

5. Error at line 10 - % Invalid Input Detected: {permit|deny} expected
access-list 2169 XXX 17 154.178.22.78 126.83.109.8 116.77.25.121 76.8.1.6  ! INVALID action
                 ^

6. Error at line 12 - % Invalid Input Detected - Port
access-list 2269 permit udp any any eq XXX  ! INVALID destPort
                                       ^

7. Error at line 16 - % Invalid Input Detected: IP Protocol or <0-255> expected
access-list 100 permit -1 host 242.226.11.199 host 240.191.18.95  ! INVALID protocol
                       ^

8. Error at line 18 - % Invalid Input Detected - IP Address
access-list 2620 permit tcp 131.240.92.220 111.98.233.234 XXX discard any log-input  ! INVALID srcPort
                                                           ^

9. Error at line 19 - % Invalid Input Detected: {permit|deny} expected
access-list 178 XXX 6 any 143.205.108.132 139.119.48.54  ! INVALID action
                ^

10. Error at line 21 - % Invalid Input Detected - IP Address
access-list 2114 permit udp any XXX  ! INVALID destIP
                                ^

                                                                        1,1           Top
```

Figure 4.5 – CRIP Tool Error Output - Verbose Error Reporting

```
Parsed ACLs_1000Rules_Extended_50PercentErr.txt - 1000 lines, with 506 errors detected.
~
~
~
~
~
~
~
~
~
:browse confirm saveas                                                  2,0-1         All
```

Figure 4.6 – CRIP Tool Error Output - Quiet Error Reporting

4.2.4 Intermediate Code Generation and Configuration Output Implementation

Two classes were implemented to store an intermediate representation of the ACL rules once they have successfully been parsed. These are the `PacketFilter` and `PacketFilterRule` class. The `PacketFilter` class contains a collection of `PacketFilterRule`, implemented as a C#.NET `ArrayList`. The `ArrayList` type differs from an array in C# as the list it contains can vary in size. An array has to redeclare the memory its using whenever its size changes. The `Arraylist` can be added to or deleted from dynamically, and encapsulates the memory reallocation process.

4.3 CRIPE Tool Impementation

An evaluation tool was developed, also with the Visual C# programming language, running on the .NET runtime framework. The CRIPE evaluation application has a GUI front-end which can be used to specify a range of rule set generation, testing, and evaluation parameters can be set. It can generate a selection of configuration files, for testing the CRIP tool. Synthetic ACLs can be generated using a random

rule generation algorithm. The rules sets can be created with a certain specified percentage of rules containing errors for testing purposes. CRIPE can then test the CRIP tool, by running it with the generated input files, and record metrics to evaluate the parsing and validation performance.

4.3.1 Evaluation User Interface

The GUI interface to the CRIPE tool is shown in Figure 4.7 and Figure 4.8. The first screen shot shows the synthetic rule generate interface, and the second shows the automated testing mechanism.

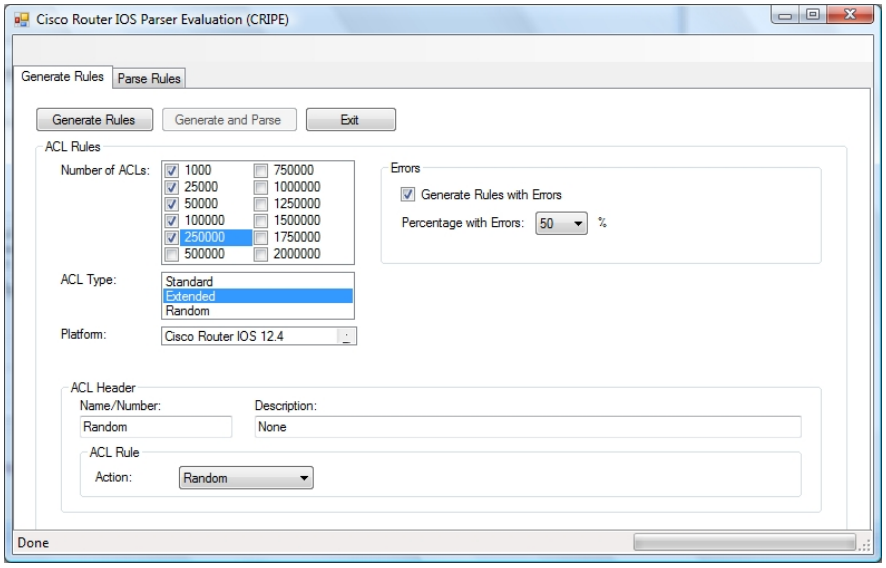

Figure 4.7 – CRIPE tool GUI - Synthetic rule generation tab

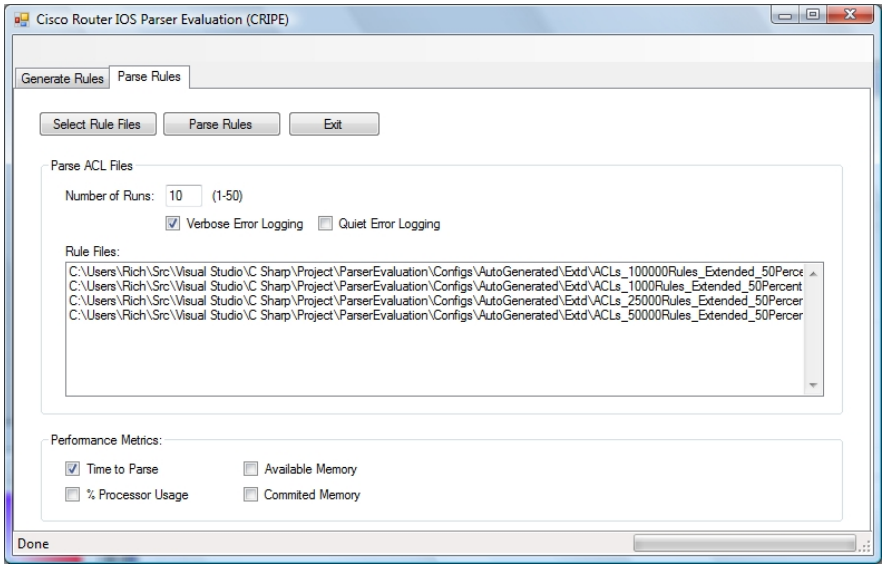

Figure 4.8 – CRIPE tool GUI - CRIP evaluation tab

The random algorithm for generating synthetic rules borrows from the rule grammar

parsing functionality. Cisco standard and extended rules can be automatically generated, creating the appropriate commands, keywords, and parameters, and using the algorithm to assign random values to the tokens which make up the rule. The synthetic rules can also be created with errors randomly assigned to any of the tokens within the rule.

4.4 Implementation Conclusions

This chapter has shown the successful implementation of the CRIP ACL parsing and validation tool, as well as the CRIPE tool which provides an evaluation harness for CRIP. A fully functional tool was developed, using the object oriented C#.NET platform. This was found to have an excellent regular expression code library, which was used extensively in both tools, although Microsoft's on-line documentation for this was a little disappointing. The CRIP tool has been implemented for maximum code reuse, and extensibility, using modular development techniques. This is important when working with languages such as for network device configuration, as they tend to change rapidly [75]. The CRIP tool can parse and validate all the major ACL types from the Cisco IOS, which is implemented on filtering routers and firewalls. The CRIP tool was able to process realistic sized rule sets, and produce useful error reporting output about problems with the rule set syntax and semantics. The CRIPE tool can generate configuration files, for testing, which contain synthetic ACLs generated using a random algorithm. It can then automate evaluation of the CRIP tool, by running it a configurable number of times and collecting performance metrics.

Chapter 5

Evaluation

5.1 Introduction

This chapter provides an analysis of the implementation of the CRIP tool, described in Chapter 4. A review of the results of a thorough evaluation of the tool is detailed in Sections 5.2 and 5.3, along with discussion of any highlighted problems. The evaluation was carried out using the CRIPE evaluation application, described in Chapter 4, to automate the necessary experiments. The evaluation is split into two parts. First an evaluation of the CRIP tool's validation algorithms, for both the OSI layer 3 based - standard ACL - algorithm, and the layer 3 and 4 based - extended ACL - algorithm. The second part, is a performance evaluation, based around the timings of the CRIP tool to process various types of rule sets.

5.2 CRIP Validation Evaluation

5.2.1 Hand Crafted Rules

The CRIP tool's validation algorithms were evaluated during continual unit testing with hand crafted ACL test configurations, as part of the implementation stage. Bounds testing was carried out for all tokens being parsed, within all modules which perform validation. An example of a test file is shown in Listing 5.1. This test configuration file checks the token pattern matching, and validation algorithm, for the icmp-type and icmp-code filtering fields (see Listing 3.6 for the ICMP extended ACL definition, and the Cisco documentation for the filtering field valid values [92]).

Listing 5.1 – bounds testing snippet]Hand crafted rules - Cisco extended ACL - icmp-type [icmp-code] bounds testing snippet

```
! icmp bounds testing
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 -1
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1  256
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 0
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 255
```

```
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.10 5
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 echo
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 echohobbes
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 0 0
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 0 -1
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 0 256
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 1 1
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 8 0 log
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 echo-reply calvin
access-list 169 permit icmp 172.168.1.1 0.0.0.254 host 192.168.1.1 redirect fragments
```

5.2.2 Synthetic Rules

The following experiments were carried out to evaluate the CRIP tool's parsing and validation algorithm. The CRIPE evaluation harness was used to generate various sizes of ACL rule sets, containing randomly generated synthetic rules. Both standard ACL rule sets, and extended ACL rule sets were generated to evaluate the parsing and validation algorithms for the different types of rules. The algorithm for the standard rules, validates the OSI layer 3 filtering fields, and the extended rule algorithm, validates OSI layers 3 and 4 filtering fields. Three different types of input rule sets were used. Rule sets with no errors in any of the rules, rule sets with approximately 50% random errors, and rule sets with 100% errors.

Error Free Rule Sets

The first experiment ran the CRIP tool, automatically from the CRIPE application with standard ACL rules sets, containing no errors. For each of the rule sets, shown in Table 5.1 CRIP was run 10 times, and the number of errors reported was averaged. Listing 5.2 shows a snippet of the synthetic rules used in the experiment. The results in the Table show, no anomalies were highlighted and the tool correctly parsed and validated the error free rules sets.

Listing 5.2 – Synthetic Rules generated by CRIPE application - Cisco standard ACL snippet

```
access-list 3 deny host 72.25.93.164
access-list 79 deny 80.133.132.6 239.202.200.89 log
access-list 1860 permit host 23.176.205.235
access-list 70 permit 108.60.250.100 202.134.179.29
access-list 1989 permit 246.88.58.74 254.62.161.218 log
access-list 1306 deny any
access-list 1781 deny host 13.150.155.26 log
access-list 54 permit any
access-list 73 deny host 17.63.186.116
access-list 1896 deny host 78.97.240.229 log
```

```
access-list 1344 permit 254.252.250.169 156.185.159.170
```

Table 5.1 – OSI Layer 3, Standard ACL validation algorithm evaluation - no errors in rule sets.

Errors	Number of rules in rule set				
	1,000	25,000	50,000	100,000	250,000
Errors in L3 Synthetic Rules	0	0	0	0	0
Errors reported by CRIP tool	0	0	0	0	0

A similar experiment was carried out for extended ACL rule sets, and again, the CRIP tool correctly reported no errors for any of the rule sets. The results are shown in Table 5.2. The Listing 5.3 shows a snippet of the layer 4, extended ACL, synthetic rules used in the experiment.

Table 5.2 – OSI Layer 4, Extended ACL validation algorithm evaluation - no errors in rule sets.

Errors	Number of rules				
	1,000	25,000	50,000	100,000	250,000
Errors in L4 Synthetic Rules	0	0	0	0	0
Errors reported by CRIP tool	0	0	0	0	0

Listing 5.3 – Synthetic Rules generated by CRIPE application - Cisco extended ACL snippet

```
access-list 109 deny 6 176.143.218.3 160.244.201.144 100.223.55.183 215.78.226.255
access-list 2452 permit tcp 5.215.49.105 164.145.202.25 any
access-list 148 permit tcp any any
access-list 2590 deny tcp any 163.152.227.197 157.194.190.3 established
access-list 168 deny 6 42.80.216.117 100.63.134.193 207.203.237.135 231.157.216.175
access-list 2660 deny 6 any 50.4.26.55 217.87.168.0 gt 26226 dscp 11
access-list 151 deny 161 255.69.83.149 166.167.170.19 host 35.109.5.171
access-list 184 deny tcp host 207.162.53.70 any lt talk tos min-monetary-cost
access-list 102 permit ip any 107.131.230.217 165.41.68.240
access-list 2509 deny udp 19.76.251.123 79.152.141.196 any
access-list 2018 permit udp 3.210.199.72 99.58.206.229 168.22.19.71 139.204.79.94 log-
    input
```

Rule Sets With 50% Errors

The second experiment in this series, involved calling the CRIP tool standard and extended ACL rules sets, containing approximately 50% errors. Listing 5.4 shows a snippet of the standard ACL rules used in the experiment, some of the rules having

random errors. The results in Table 5.3 show, the tool correctly reported the errors in the rules sets.

Table 5.3 – OSI Layer 3, Standard ACL validation algorithm evaluation - approx. 50% errors in rule sets.

Errors	Number of rules in rule set				
	1,000	25,000	50,000	100,000	250,000
Errors in L3 Synthetic Rules	520	12,501	25,069	50,050	125,206
Errors reported by CRIP tool	520	12,501	25,069	50,050	125,206

Listing 5.4 – Synthetic Rules generated by CRIPE application - Cisco standard ACLs with 50% of rules containing errors

```
access-list 1703 deny 97.63.75.210 55.176.96.25
access-list 57 deny host 223.44.178.243
access-list 101 deny 193.56.137.216  ! INVALID aclId
access-list 1668 permit any
access-list 60 permit 128.164.21.57
access-list 92 deny 45.208.147.134
access-list XXX permit 39.135.63.76 49.225.135.199  ! INVALID aclId
access-list 46 deny 171.63.182.215 226.209.156.1 XXX  ! INVALID log
access-list 1964 XXX host 151.152.128.124  ! INVALID action
access-list 1889 deny host 7.47.45.137
access-list 1542 deny any
```

Listing 5.5 – CRIP tool error report - Cisco standard ACL with 50% errors

```
Parsed C:\Users\Rich\Src\Visual Studio\C Sharp\Project\ParserEvaluation\Configs\
    AutoGenerated\Std\ACLs_25000Rules_Standard_50PercentErr.txt - 25000 lines, with
    12501 errors detected.

1. Error at line 3 - % Invalid Input Detected: IP Protocol or <0-255> expected
access-list 101 deny 193.56.137.216  ! INVALID aclId
                    ^
2. Error at line 7 - % Invalid Numbered ACL Id
access-list XXX permit 39.135.63.76 49.225.135.199  ! INVALID aclId
           ^
3. Error at line 8 - % Invalid Input Detected: log expected
access-list 46 deny 171.63.182.215 226.209.156.1 XXX  ! INVALID log
                                                  ^
4. Error at line 9 - % Invalid Input Detected: {permit|deny} expected
access-list 1964 XXX host 151.152.128.124  ! INVALID action
                ^
```

Layer 4 extended ACL filtering rule sets were also tested against the tool, and again every error in the input configurations was detected correctly. The results are shown in Table 5.4 and samples from the input and error report files are shown in Listings 5.6 and 5.7.

Table 5.4 – OSI Layer 4, Extended ACL validation algorithm evaluation - approx. 50% errors in rule sets.

Errors	Number of rules				
	1,000	25,000	50,000	100,000	250,000
Errors in L4 Synthetic Rules	506	12,479	25,169	50,083	124,755
Errors reported by CRIP tool	506	12,479	25,169	50,083	124,755

Listing 5.6 – Synthetic Rules generated by CRIPE application - Cisco extended ACL with 50% of rules containing errors

```
access-list 138 deny icmp 213.114.63.17 105.201.61.167 15.70.18.108 120.33.170.131 XXX
      ! INVALID log
access-list 2168 deny tcp 146.152.31.57 60.199.80.172 198.27.198.34 78.155.112.XXX  !
    INVALID destIP
access-list 186 permit udp host 41.17.152.166 171.15.59.211 9.74.28.204 neq XXX  !
    INVALID destPort
access-list 2074 deny tcp host 34.65.198.226 host 132.64.76.17 dscp 49
access-list 2257 permit 46 any any precedence 5
access-list 114 deny 6 55.91.169.220 149.142.127.77 host 90.11.65.124
access-list 141 deny 6 42.55.58.209 87.61.117.121 host 89.158.169.117
access-list 2559 permit tcp any any
access-list 193 permit 6 host 93.25.197.201 host 6.198.201.137 XXX  ! INVALID log
access-list 2236 deny tcp 101.102.126.8 238.111.220.33 any
```

Listing 5.7 – CRIP tool error report - Cisco extended ACL with 50% errors

```
Parsed C:\Users\Rich\Src\Visual Studio\C Sharp\Project\ParserEvaluation\Configs\
    AutoGenerated\Extd\ACLs_25000Rules_Extended_50PercentErr.txt - 25000 lines, with
    12479 errors detected.

1. Error at line 1 - % Invalid Input Detected
access-list 138 deny icmp 213.114.63.17 105.201.61.167 15.70.18.108 120.33.170.131 XXX
      ! INVALID log
                                                                                ^
2. Error at line 2 - % Invalid Input Detected - Wildcard Mask
access-list 2168 deny tcp 146.152.31.57 60.199.80.172 198.27.198.34 78.155.112.XXX  !
    INVALID destIP
                                                          ^
3. Error at line 3 - % Invalid Input Detected - Port
access-list 186 permit udp host 41.17.152.166 171.15.59.211 9.74.28.204 neq XXX  !
    INVALID destPort
                                                                    ^
```

Rule Sets With 100% Errors

Similar results were obtained for rule sets with every rule containing an error, and are shown in Tables 5.5 and 5.6. This shows the tool has correctly detected over a million and a half, randomly generated, errors in this experiment alone. This may be partly due to the extremely thorough unit testing procedures, described in Section 5.2.1, which were carried out, in part, during the implementation. The process to create the synthetic rules is closely based on the CRIP tool's validation algorithms, and token regular expression patterns. This testing should possibly be based more on data, reverse engineered, from actual device OS if possible. this would mean a detachment from the CRIP tool's development process, and also from the vendor documentation which has been shown to be incomplete.

Table 5.5 – OSI Layer 3, Standard ACL validation algorithm evaluation - 100% errors in rule sets.

	Number of rules in rule set				
Errors	1,000	25,000	50,000	100,000	250,000
Errors in L3 Synthetic Rules	1,000	25,000	50,000	100,000	250,000
Errors reported by CRIP tool	1,000	25,000	50,000	100,000	250,000

Table 5.6 – OSI Layer 4, Extended ACL validation algorithm evaluation - 100% errors in rule sets.

	Number of rules				
Errors	1,000	25,000	50,000	100,000	250,000
Errors in L4 Synthetic Rules	1,000	25,000	50,000	100,000	250,000
Errors reported by CRIP tool	1,000	25,000	50,000	100,000	250,000

5.3 Performance Evaluation

For the performance evaluation of the CRIP tool, realistic rule set sizes were used. From the literature reviewed in Chapter 2 rule sets used in industry range seem to range from just a few rules, up to tens of thousands of rules [10, 35]. Therefor, files containing synthetic rules in the quantities shown in Table 5.7, were created using the CRIPE application's random rule generation module.

The system used for the performance evaluation experiments was an Intel Core2 Duo T6400 2GHz processor, with 4GB of RAM, running Windows Vista Home Premium, with the Microsoft .NET Runtime Framework - Version 3.5 SP1. The performance metric recorded by the evaluation tool CRIPE, is the time taken to process a rule set file. The timings were taken with a precision stopwatch, and results shown in

the following tables have been rounded to three decimal places. All unnecessary application processes were closed on the system while experiments were taking place.

Table 5.7 – Synthetic rule sets created for performance evaluations.

ACL Type	Number of rules 1,000	25,000	50,000	100,000	250,000
Standard with no errors	Rule Set 1	Rule Set 2	Rule Set 3	Rule Set 4	Rule Set 5
Standard with 50% errors	Rule Set 6	Rule Set 7	Rule Set 8	Rule Set 9	Rule Set 10
Standard with 100% errors	Rule Set 11	Rule Set 12	Rule Set 13	Rule Set 14	Rule Set 15
Extended with no errors	Rule Set 16	Rule Set 17	Rule Set 18	Rule Set 19	Rule Set 20
Extended with 50% errors	Rule Set 21	Rule Set 22	Rule Set 23	Rule Set 24	Rule Set 25
Extended with 100% errors	Rule Set 26	Rule Set 27	Rule Set 28	Rule Set 29	Rule Set 30

Baseline Performance

Quiet error reporting mode was used as a base line, to compare other results against. The Quiet reporting option -q, when passed to the CRIP tool, results in no output files being produced The number of rules found to have errors is reported to the CLI, but no error report file is created. This should, in theory, should return the best performance results. Two different input rule set types were evaluated: Rule Sets 1-5 Standard ACLs with no errors, and Rule Sets 26-30 Extended ACLs with 100% errors. The experimental conditions were the 5 different sizes of the rule sets with numbers of rules ranging from 1,000 to 250,000. The time to process metric was recorded for 5 separate runs, and an average calculated.

Table 5.8 – Average processing time for OSI Layer 3, Standard ACL rules no errors in rule sets - quiet reporting.

	Number of rules 1,000	25,000	50,000	100,000	250,000
Standard ACL No Errors	0.040	0.987	2.306	4.493	10.860
Extended ACLs 100% Errors	0.736	17.999	37.229	71.408	182.197

The results, shown in Table 5.8, show linear growth in the average time taken to process the rule sets, as the size increases. This is shown in Figure 5.1. There is no overhead for writing the error report to the output file in this experiment, so other experiments carried out on this system should have increased processing times on the standard ACL with no errors results. If the average time to process a single record

is calculated from the Standard ACL base line timings, it is on average 0.0000429 seconds. This can be used to compare other results against.

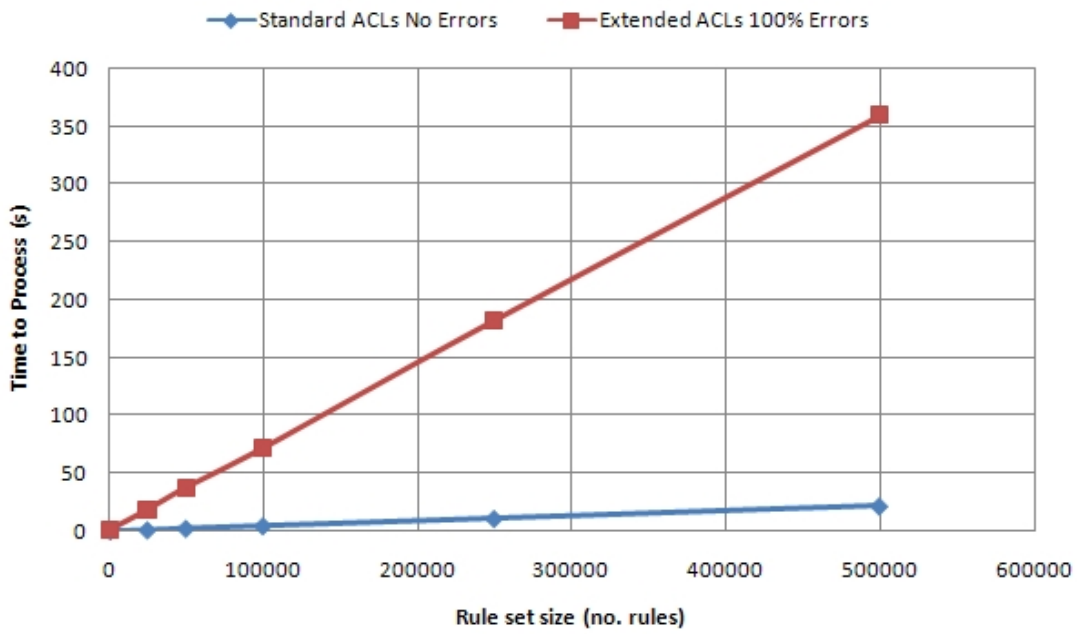

Figure 5.1 – Average Processing Time for CRIP tool, in quiet mode.

Standard vs Extended Algorithm Performance

Performance testing of the Standard ACL validation algorithm was carried out, for verbose error logging, using data sets 1-15. This is compared against the base line standard ACL timings discussed previously, in Table 5.9. As suspected, the base line timings from the quiet mode experiment, are the fastest.

Table 5.9 – Average processing time for OSI Layer 3, Standard ACL rules in rule sets - verbose error reporting.

	Number of rules				
	1,000	25,000	50,000	100,000	250,000
Standard ACL No Errors Base Line	0.040	0.987	2.306	4.493	10.860
Standard ACL No Errors	0.090	2.337	2.930	7.267	16.961
Standard ACL 50% Errors	0.539	12.825	27.708	55.929	129.287
StandardACLs 100% Errors	1.006	24.029	51.558	99.647	265.085

A direct comparison can be made between these timings and the verbose logging with no errors and the same rule set sizes (rule sets 1-5 in row 2 of the table). The average time to process a single record, for the standard ACLs with verbose logging

experiment, is 0.000078, 0.000532, and 0.001012, for no errors, 50% errors, and 100% errors respectively. This is compared to the average average of 0.0000429 seconds for the base line experiment.

The same linear growth as the base line experiment for the average time taken to process the rule sets, as the size increases, is shown in Figure 5.2.

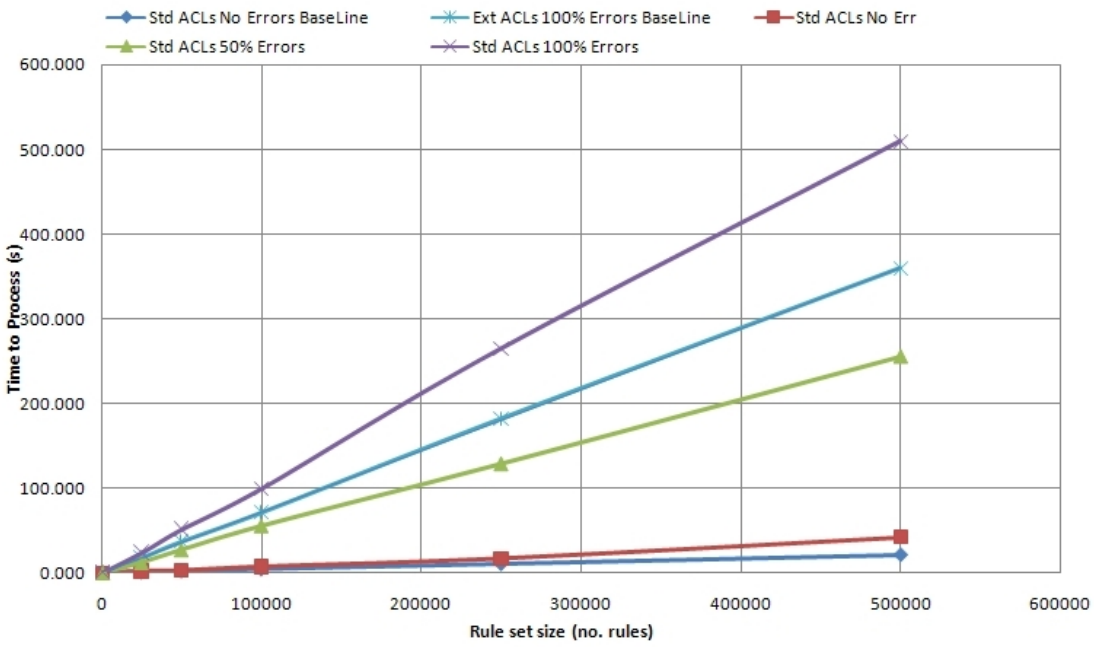

Figure 5.2 – Average Processing Time for CRIP tool, Standard ACLs - verbose error reporting.

Performance testing of the Extended ACL validation algorithm, was also carried out similarly for verbose error logging. This time data sets 16-30 are being used to evaluate the CRIP tools. This is again compared against the base line standard, but this time the extended ACL timings base line. The results are shown in Table 5.10. In this experiment a direct comparison can be made between the rule sets with 100% errors and the 100% error base line reuslts (first and last row o fthe table). These results are very similar, more so than the standard ACLs base line comparison.

The average time to process a single record, for the extended ACLs with verbose logging experiment, is 0.00016312, 0.000458482, and 0.00075855, for no errors, 50% errors, and 100% errors respectively. When compared to the standard ACL average times to process a single rule, we notice that for rule sets with no erros the average timing is smaller for the extended rule algorithm, whereas the average time is larger for the rule sets with 100% errors. This difference in processing speed of the algorithms can be seen clearly if the rule sets with 100% errors are compared in Figures 5.2 and 5.3 The standard algorithm takes over 500 seconds to process 500,000 ACLs, and the extended algorithm only takes 364 seconds.

Table 5.10 – Average processing time for OSI Layer 4, Extended ACL rules in rule sets - verbose error reporting.

	Number of rules				
	1,000	25,000	50,000	100,000	250,000
Extended ACLs 100% Errors Base Line	0.736	17.999	37.229	71.408	182.197
Extended ACL No Errors	0.175	3.947	7.947	16.025	39.925
Extended ACL 50% Errors	0.484	10.847	22.210	48.337	104.868
Extended ACL 100% Errors	0.753	18.392	40.988	76.974	185.907

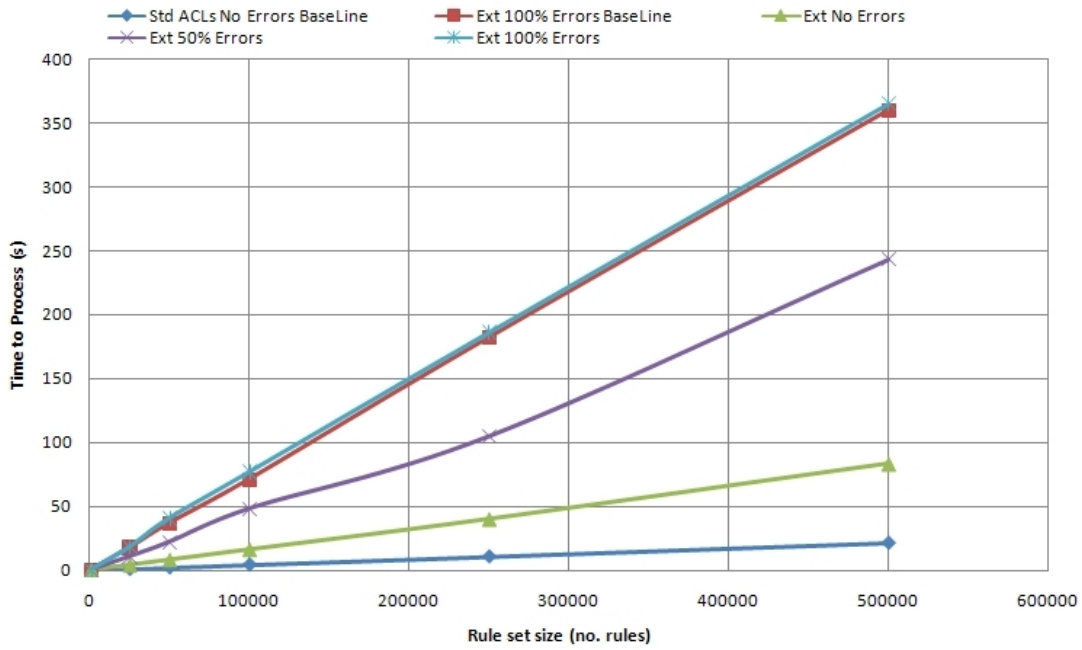

Figure 5.3 – Average Processing Time for CRIP tool, Extended ACLs - verbose error reporting.

5.4 Evaluation Conclusions

This chapter provides an evaluation of the implementation of the CRIP ACL validation tool. The tool was evaluated for the completeness of its validation algorithms. This was carried in thoroughly, using a range of different experiments for both the OSI layer 3 standard ACLs, and for layer 3 and 4 extended ACLs. The tool was also evaluated using a range of different experiments to record the tools performance. The metric used to record the tools performance was an accurate timing of the length of time taken to process a rule set.

The validation algorithm evaluation was inconclusive. The results show the tool performs well against the test data used, but these synthetic ACL rule sets were created

based around the same Cisco documentation that the validation algorithms are based on. The test data should perhaps be based more on real firewall configurations, from on-line network devices. Or the synthetic rule creation algorithm should be based around reverse engineering the command definitions from the actual devices. This would mean a detachment from the development process, and also from the vendor documentation, which has been shown to be flawed.

The performance results were all linear when the results were plotted on graphs. This seems ideal for the performance of a tool like this, and larger rule sets than would probably ever occur in real networks were used. Some interesting results were highlighted when comparing the standard ACL and extended ACL validation algorithms performances. The standard algorithm executes less lines of code, and always validates less tokens, than the extended algorithm, due to the rules having less filtering fields. The results showed as the rules sets got larger, the extended algorithm consistently outperformed the standard algorithm.

The Performance from the tool was shown to be usable for realistic sized rule sets. The slowest of the algorithms, with 100% errors in the rule sets can still validate 1,000 ACL rules in around a second.

Chapter 6

Conclusion

6.1 Aim and Objectives

The aim of this thesis is to produce a tool which can parse and validate firewall rule sets. To implement and evaluate it with realistic test data. Objectives to support the overall aim of the thesis are as follows:

1. Investigate and review the extensive literature in the fields of policy-based security and in particular firewall policy management.
2. Design a tool to perform the non-trivial task of off-line firewall policy validation, based around firewall device configurations.
3. Implement and test the system, using appropriate tools to realise the design sepecifications.
4. Evaluate the performance of the prototype system, validating its performance using experiments with realistic data sets.
5. Investigate and propose a framework, which the policy validation tool could integrate with, to support system administrators in the management of firewall policies.

6.1.1 Objective 1. Investigate and review the extensive literature in the fields of policy-based security and in particular firewall policy management

The first objective was accomplished and reported on in Chapter 2. Literature was reviewed in the areas of network security, security policies, firewall policies, and policy modeling systems. The different types, and abstractions, of security polices were explored through a wide range of literature. A comprehensive study of firewall policy based systems was then undertaken, comparing these systems across various attributes which were potentially useful in the development of the validation tool. A novel taxonomy of these systems, was produced, and is reported on in Chapter 2.

From the research reviewed, it was recognised that security policies are the central pillar in network security, and that security should be built around policies. High level policies are created by management, in coordination with senior administrators, and are normally written in a natural language. These policies are then mapped to the low level technical policies, such as firewall policies, which network devices enforce. The mapping of high level policies to low level device configurations is typically performed by network administrators, and is normally a manual process. This task is complex and error prone as there is a large conceptual gap between the high level policies and the low level policies. Low level firewall policies are complex, even if only containing small numbers of rules. Lots of research details the complexity problems with 'first match' rule sets, which depend on the ordering of the rules. These complexities mean low level firewall policies are difficult to understand, by simply reviewing the rules, due to the interactions between the rules. Larger rule sets, or if rule sets are deployed on multiple firewalls and filtering devices, make for even more complexity. Many systems have been proposed to help with the understanding and management of these policies.

Policy-based systems can be categorised as high level abstract policy languages, query-based rule set testing systems, and other firewall rule set auditing systems. High level languages, are used to create an abstract layer of policy between the high level policies and low level firewall configurations. Many high level language systems were developed before it was realised that system administrators did not favor these abstract languages. System administrators seem to prefer to work in the lowest possible language, and at the lowest level. The tools at this level have no layers of abstraction and tend to be faster and more robust, so they are trusted more. This makes a lot on sense, as administrators have to rely on these tools to work correctly to be able to fix problems quickly. Some of these abstract policy languages actually needed specialists to configure them. Some of the languages also have limitations with the features they support. Many of these systems are a clean-slate approach, using new high level languages and abstract models. These are detached from the network management of most organisations. Also, no high level language has been created which seems to have been used by more than a single research group. No common language or model seems to exist for this extra layer of policy abstraction.

Other auditing tools were reviewed, including query-based analysis systems some of which have been produced specifically for firewall auditing. They can audit the firewall rule set by simulating network traffic passing through the firewall. The administrator can enter queries and the system provides an answer showing which traffic can get where. Similar problems were encountered with these systems, as administrators testing the system had to learn the query language which was not popular. A typical reaction to this was to add a GUI to the system, but this is not always helpful, especially when developing tools for administrators. GUIs generally

seem to be good for less experienced users, or for visualising complex parts of data sets, but textual CLIs seem to be preferred by administrators for most jobs.

Some systems overcame the problems of abstract language based systems and models, by hiding these from the user. The model would be created from low level device configurations automatically. These are the types of systems which have been successfully commercialised. Other off-line auditing systems were reviewed, including rule set anomaly detection systems, and visualisation systems for firewall policies, and high level policy inferencing systems. Reverse engineering a high level security policy into a natural language which management can read, is an interesting research area. Not very much research has been carried out in this field, probably as it tends to be built on top of other research into policy modeling.

From the review, it was highlighted that off-line configuration analysis is the main area were these systems have become commercialised, such as automated firewall configuration analysis, and routing configuration analysis. These passive systems have advantages over active testing systems, such as vulnerability testing tools or penetration testing tools, as they can be performed before the policies are deployed. With active testing, once a security flaw is found the network is vulnerable till a suitable patch can be deployed.

6.1.2 Objective 2. Design a tool to perform the non-trivial task of off-line firewall policy validation, based around firewall device configurations

This objective was met by producing a design for the CRIP tool, and this is detailed in Chapter 3. Background research in system administration tool design, firewalls, and firewall policy analysis systems helped accomplish this objective, as well as previous knowledge of systems analysis and design methods.

The tool was loosely based on the structure of a typical compiler. It first performs lexical analysis, than syntactic analysis, reporting any errors to the user.

The CRIP tool was designed generally with extensibility in mind. By adhering to modular object oriented design techniques, there should be maximum encapsulation of functionality, and code reuse. This is especially important when working with network device configuration languages. These languages have a tendency to change, as new features are added, or devices are upgraded.

The validation tool was designed to integrate into the proposed framework, and compliment other existing tools. This is important as system administrators tend to use tools in this way. Combining tools together to run batch jobs, and to build their own sets of utilities using scripts.

The tool was designed to be flexible, as system administration tools tend to be used

by different types of users. A text based scriptable interface is provided for administrators, but the tool could also interface with GUI applications for less experienced users. This allows the system administrator to use the low level vendor specific ACL language they typically prefer, and intentionally not introducing new abstract, untrusted languages.

6.1.3 Objective 3. Implement and test the system, using appropriate tools to realise the design specifications.

Objective 4 was accomplished, and is described in Chapter 4. It describes the successful implementation of the Cisco Router IOS ACL Parser and validation tool, CRIP. The tool was implemented following the design in chapter 3. A fully working tool was created, to assist the network administrator when creating Cisco ACL policies.

The tool was developed using the object oriented C#.NET platform, which allowed a structured approach to the coding. The .NET framework provides a large variety of code libraries which were used during the implementation, such as the excellent regular expressions library. There was an issue with Microsoft's documentation for this functionality, as it was quite poor, but some other excellent resources were available for regular expressions. An object oriented implementetion was attempted, based on the object oriented design. This was only a partial success, due to shortcuts taken with the prototype, as some of the class hierarchy was implemented in a single class. The `vi` editor was used extensively as it has built in regular expression functionality, which was useful for testing some patterns as they were developed. Also on-line resources are available for this. The editor was also used for manipulation of the automatically generated ACL configuration files. A task such as finding how many rules with errors had actually been created would have been difficult without it. A simple search for the '`! INVALID`' comment and then print the number of instances in a file, is easy in `vi`, but other simple text editors do not have such functionality.

The CRIP tool was created with a CLI, which accepts a small number of input arguments. This interface was not fully implemented. Just enough to evaluate the tool was created. For example, the -h help function was not created, which is poor, as this is the first thing a user would need.

The CRIP tool can successfully parse and validate the main types of rule sets used on Cisco's filtering devices, and firewalls. It was able to process realistic sized configuration files, and can cope easily with processing very large firewall rule sets. It produces different types of error reports to the user, depending on how much detail is required on the validation errors found in individual rules.

The tool currently does not perform all of its lexical analysis using regular expressions. In a few places where it was significantly easier to use a conditional statement,

rather than creating and testing a complex regular expression, this was done.

The resulting tool was fully tested, using a hand crafted test file for each of the objects which contained validation processing. This was done via the tools CLI, and also from custom Web and Windows GUI-based test applications. A basic windows GUI was used to perform unit testing on the tool as each component was finished. A web application was created and used to allow students at the university to use the tool as part of Cisco ACL tutorials. This provided some feedback on small problems with the validation algorithms, which were corrected.

The second part of the implementation was the creation of the CRIPE performance evaluation application. The CRIPE tool was developed to create evaluation test data to use with the CRIP tool, during its evaluation. The application can generate configuration files, which contain synthetic ACL rule sets, generated using a novel random algorithm. C# was an excellent tool to create the GUI based application with. It was extremely quick to pick up the tools needed to create basic Windows applications. A very basic web application was also created, with no previous knowledge of ASP.NET, just using a single book chapter as reference.

The CRIPE application can then automate the process of running the CRIP tool for evaluation purposes. It can generate a configurable set of evaluation runs of the CRIP tool, while collecting performance metrics about the runs. It automates the process of running the CRIP tool with a number of the different synthetic test configuration files. An average is generated across the performance metrics recorded during the multiple runs.

The tool has a flexible design and contains comprehensive functionality, as opposed to some of the other tools which perform across multiple vendor languages, but do not implement a deep range of options for any of the languages. It compliments existing systems, such as policy compliance tools, and abstract policy analysis systems.

An issue highlighted during the implementation phase, was incomplete Cisco documentation on their configuration language. Several instances of incomplete documentation was found within the small number of ACL commands that were used in the tool. Since the tool has been implemented other non-documented functionality has also been found. For example, the keywords for the `fin, psh, syn, urg` flags are not dealt with by the tool currently, as they are not in any of Cisco's documentation.

6.1.4 Objective 4. Evaluate the performance of the prototype system, validating its performance using experiments with realistic data sets

Chapter 5 reports on Objective 4. An evaluation of the implementation of the CRIP validation tool. The tool was evaluated in two different ways. Firstly the different validation algorithms were evaluated using a simple, but thorough method. This was

done by using a large range of different experiments, where the CRIP tool is required to validate both OSI layer 3 standard ACLs, and for layer 3 and 4 extended ACLs. Secondly the performance of the tool was evaluated using realistic sized rule sets. For both experiments, synthetic rules were created by a custom evaluation application, which was created for this purpose.

The validation algorithm evaluation seemed a success, as the tool had a perfect record of identifying errors in the test files. Millions of errors were correctly detected, and none missed. The results show the tool performs well against the test data used, but these synthetic ACL rule sets were created based around the same Cisco documentation that the validation algorithms are based on. Real rule sets should be used to get a better idea of the completeness of the algorithms. Or synthetic rules could be generated from the device command definitions some how. Perhaps some type of capture tool could automatically interrogate a device's CLI and the commands and parameters could be reverse engineered from the device. This may produce a more accurate algorithm by which to generate the synthetic rules.

The validation algorithms have been evaluated, and seem to work well. There is however a major drawback to these algorithms being hard coded, in isolation, inside the tool. When Cisco change the syntax of ACLs functionality, the algorithms are immediately out of date. A process of matching the algorithm automatically with a copy of the appropriate Cisco OS could solve this problem, but this would require creating such a tool.

The tool was also evaluated using a range of different experiments to record the tools performance. The metric used to record the tools performance was an accurate timing of the length of time taken to process a rule set.

Many performance evaluation experiments were carried out, with different types and sizes of ACL configuration fields. The timing performance results were all linear as the rule sets got bigger in size. Very large rule sets were used and the linear trend continued, until the system ran out of memory when creating the storage the tool uses. This was trying to process several million rules, and this is larger rule set than would ever be used in a real network. The linear performance results are ideal, and so no optimisation of the tool was deemed unnecessary.

Interesting results were highlighted when comparing the standard ACL and extended ACL validation algorithms performances. The standard algorithm, would be expected to be the quicker to run, as it executes many less lines of code, and always validates less tokens. This is due to the standard ACL rules being based around only OSI layer 3 header fields, and not layer 4 fields like the extended can use. The standard rules start off quicker to process than the extended rules, but then at larger rules sets, the extended are significatly faster to process. The results showed as the rules sets got larger, the extended algorithm consistently outperformed the standard

algorithm.

The Performance from the tool was shown to be usable for realistic sized rule sets. The slowest of the algortihms, with 100% errors in the rule sets can still validate 1,000 ACL rules in around a second, or 50,000 rules in just under a minute.

6.1.5 Objective 5. Investigate and propose a framework, which the policy validation tool could integrate with, to support system administrators in the management of firewall policies

An overview of a simple framework to integrate existing policy-based management systems and the CRIP tool, with a central XML-based modeling language, is proposed. An XML-based model would be open rather than the closed models such as with commercial systems such as Cisco, or Algosec are based on. The framework would be based on abstracting the network device configurations, into XML-based language descriptions. The XML-based Netconf protocol could possibly be used as the language, which would provide retrieval and deployment functionality.

This type of policy description language may not be powerful enough to describe entire security policies however, such as user authentication and access control policies. Other frameworks have been suggested for this overall security policy modeling, and a simple configuration based model may prove to be more achievable. The frameworks proposed to encompass all security policies tend to be left as theory, and never implemented.

6.2 Future Work

6.2.1 XML-Based Framework

There are many research ideas for the an overall policy management framework. Firstly investigate the Netconf protocol to find out if it can be used as a way of describing configurations abstractly. Whether or not the language it uses can be parsed and used by other systems, such as would be necessary to create auditing or simulation systems on top of it. Investigating the retrieval and deployment of configurations with Netconf, such as which vendors, and on which of their platforms, have the protocol been implemented on.

Complete device configurations could be modeled in the XML-based language, or at least enough to be able to perform almost all configuration management on top of the model, similar to Caldwell's EDGE system, but using an XML database of configuration information. If this could be realised, a full configuration management system could be integrated with the model, as well as a network device simulator

system, and auditing systems.

6.2.2 CRIP Tool

A usability experiment, based around networking students using the tool for various tutorials was planned, but was not carried out. This would provide valuable information on the implementation of the tool, and its usefulness in a teaching environment.

Modules could be added to the tool to validate PIX and ASA firewall specific syntax, and evaluations between the different algorithms used.

A module could be added, or perhaps a separate tool could be integrated, to take valid configurations and abstract an XML-based policy specification of firewall rule sets.

Listings

References

[1] C. System, "Cisco security manager." [Online]. Available: http://www.cisco.com/en/US/products/ps6498/index.html

[2] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *IEEE Symposium on Security and Privacy*, vol. 0, pp. 17–31, 1999.

[3] A. Mayer, A. Wool, and E. Ziskind, "Offline firewall analysis," *International Journal of Information Security*, vol. 5, no. 3, pp. 125–144, 2006.

[4] E. Al-Shaer and H. Hamed, "Modeling and management of firewall policies," *IEEE Transactions on Network and Service Management*, vol. 1-1, pp. 2–10, 2004.

[5] ——, "Design and implementation of firewall policy advisor tools," DePaul University, CTI, Tech. Rep., 2002.

[6] BERR, "2008 information security breaches survey." [Online]. Available: http://www.berr.gov.uk/files/file45714.pdf

[7] B. Blakley, "The emperor's old armor," in *Proceedings of the 1996 workshop on New security paradigms*. ACM Press New York, NY, USA, 1996, pp. 2–16.

[8] M. Bishop and S. Peisert, "Your security policy is what," University of California at Davis, Tech. Rep., 2006.

[9] T. Wong, "On the usability of firewall configuration," in *Symposium on Usable Privacy and Security*, 2008.

[10] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[11] B. Fraser, J. P. Aronson, N. Brownlee, and F. Byrum, "Site security handbook (rfc 2196)," Sep 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2196.txt?number=2196

[12] A. D. Rubin, D. Geer, and M. J. Ranum, *Web Security Sourcebook*. Wiley, 1997.

[13] S. Lee, T. Wong, and Kim, "To automate or not to automate: On the complexity of network configuration," in *IEEE International Conference on Communications (ICC)*, 2008, pp. 5726 – 5731.

[14] Y. Bhaiji, *CCIE Professional Development - Network Security Technologies and Solutions*. Cisco Press, 2008.

[15] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miege, "A formal approach to specify and deploy a network security policy," in *Formal Aspects in Security and Trust*, ser. IFIP International Federation for Information Processing, vol. 173/2005. Springer Boston, 2004, p. 203Ű218.

[16] T. Corbitt, "Protect your computer system with a security policy," *Management Services*, vol. 46(5), pp. 20–21, 2002. [Online]. Available: http://findarticles.com/p/articles/mi_qa5428/is_200205/ai_n21313131/pg_2?tag=artBody;col1

[17] C. P. Pfleeger, *Security in Computing 4th Edition*. Prentice Hall, 2006.

[18] S. Bellovin and W. Cheswick, "Network firewalls," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 50–57, 1994.

[19] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls, Second Edition*, 2nd ed. O'Reilly Media, Inc., Jun 2000. [Online]. Available: http://oreilly.com/catalog/9781565928718/index.html

[20] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, *Firewalls and Internet Security: Repelling the Wiley Hacker, 2nd Edition*. Addison-Wesley, Feb 2003.

[21] A. Wool, *Packet Filtering and Stateful Firewalls*. Wiley, 2006, ch. Firewall Architectures, p. 526.

[22] C. C. Zhang, M. Winslett, and C. A. Gunter, "On the safety and efficiency of firewall policy deployment," *Security and Privacy, IEEE Symposium on*, vol. 0, pp. 33–50, 2007.

[23] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 0177, 2000.

[24] E. Al-Shaer and H. Hamed, "Taxonomy of conflicts in network security policies," *IEEE Communications Magazine*, vol. 44, no. 3, pp. 134– 141, 2006.

[25] F. Cuppens, N. Cuppens-Boulahia, and J. G. Alfaro, "Detection and removal of firewall misconfiguration," in *Conference On Communication, Network, and Information Security (CNIS)*, 2005, pp. 154–162.

[26] J. D. Guttman, "Rigorous automated network security management," *International Journal of Information Security*, vol. 4, pp. 29–48, 2005.

[27] B. Schneier, *Secrets and Lies - Digital Security in a Networked World*. Wiley, 2000.

[28] D. Danchev, "Building and implementing a successful information security policy," online at www.windowsecurity.com, 2003.

[29] E. Guttman, L. Leong, and G. Malkin, "Users' security handbook (rfc 2504)," Feb 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2504.txt?number=2504

[30] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," *Lecture Notes in Computer Science*, vol. 2171, pp. 137–196, 2000.

[31] C. Systems, "Cisco safe," 2009. [Online]. Available: http://www.cisco.com/en/US/netsol/ns954/index.html

[32] E. M. Madigan, C. Petrulich, and K. Motuk, "The cost of non-compliance: when policies fail," in *SIGUCCS '04: Proceedings of the 32nd annual ACM SIGUCCS conference on User services*. New York, NY, USA: ACM, 2004, pp. 47–51.

[33] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, pp. 30–50, 2000.

[34] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting edge of ip router configuration," in *Proceedings of 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, 2003.

[35] S. Hinrichs, "Policy-based management: Bridging the gap," in *Computer Security Applications Conference, Annual*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 1999, pp. 209–218.

[36] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, 2004.

[37] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith, "Implementing a distributed firewall," in *Proceedings of the 7th ACM conference on Computer and communications security*. ACM New York, NY, USA, 2000, pp. 190–199.

[38] M. Abrams and D. Bailey, "Abstraction and refinement of layered security policy," in *Information Security: An Integrated Collection of Essays*. IEEE Computer Society Press, 1995, pp. 126–136.

[39] R. Sandhu, "The typed access matrix model," in *1992 IEEE Computer Society Symposium on Research in Security and Privacy, 1992. Proceedings.*, 1992, pp. 122–136.

[40] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li, "An automated framework for validating firewall policy enforcement," in *POLICY '07: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 151–160.

[41] NIST, "Generally accepted principles and practices for securing information technology systems," NIST, Tech. Rep., 1996.

[42] ——, "Special publication 900-64 - security considerations in the system development life cycle," NIST, Tech. Rep., 2008.

[43] E. M. Haber and J. Bailey, "Design guidelines for system administration tools developed through ethnographic field studies," in *CHIMIT '07: Proceedings of the 2007 symposium on Computer human interaction for the management of information technology*. New York, NY, USA: ACM, 2007, p. 1.

[44] P. Jaferian, D. Botta, F. Raja, K. Hawkey, and K. Beznosov, "Guidelines for designing it security management tools," in *CHiMiT '08: Proceedings of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*. New York, NY, USA: ACM, 2008, pp. 1–10.

[45] V. Tsoumas and T. Tryfonas, "From risk analysis to effective security management: towards an automated approach," *Information Management & Computer Security*, vol. 12, pp. 91–101, 2004.

[46] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Workshop on Policies for Distributed Systems and Networks*, 2001.

[47] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, "Specifications of a high-level conflict-free firewall policy language for multi-domain networks," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2007, pp. 185–194.

[48] V. Zaliva, "Firewall policy modeling, analysis and simulation: a survey," SourceForge, Tech. Rep., 2008.

[49] J. D. Guttman, "Filtering postures: local enforcement for global policies," vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 1997, p. 0120.

[50] ——, "Security goals: Packet trajectories and strand spaces," *Lecture Notes in Computer Science*, pp. 197–261, 2001.

[51] C. Systems, "Cisco secure policy manager." [Online]. Available: http://www.cisco.com/en/US/products/sw/secursw/ps2133/prod_technical_reference09186a00800a9ebc.html

[52] T. Uribe and S. Cheung, "Automatic analysis of firewall and network intrusion detection system configurations," *Journal of Computer Security*, vol. 15, no. 6, pp. 691–715, 2007.

[53] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello, "Configuration management at massive scale: system design and experience," in *2007 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–14.

[54] S. Pozo, R. Ceballos, and R. M. Gasca, "Afpl, an abstract language model for firewall acls," in *ICCSA '08: Proceedings of the international conference on Computational Science and Its Applications, Part II*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 468–483.

[55] S. Hazelhurst, A. Fatti, and A. Henwood, "Binary decision diagram representations of firewall and router access lists," Department of Computer Science, University of the Witwatersrand, Tech. Rep., 1998.

[56] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," University of theWitwatersrand, Tech. Rep., July 1999.

[57] R. Marmorstein and P. Kearns, "A tool for automated iptables firewall analysis," in *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC)*. Berkeley, CA, USA: USENIX Association, 2005, pp. 44–44.

[58] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (snmp) (rfc 1157)," May 1990. [Online]. Available: http://tools.ietf.org/html/rfc1157

[59] J. Schoenwaelder, "2002 iab network management workshop (rfc 3535)," 2003.

[60] C. Ehret, "From snmp deception to verinecŠs cisco service," Master's thesis, University of Fribourg, Switzerland, 2005.

[61] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy core information model - version 1 specification (rfc 3460)," 2001. [Online]. Available: http://tools.ietf.org/html/rfc3460

[62] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The cops (common open policy service) protocol (rfc2748)," Jan 2000. [Online]. Available: http://tools.ietf.org/html/rfc2748

[63] A. Bierman, K. Crozier, R. Enns, T. Goddard, E. Lear, P. Shafer, S. Waldbusser, and M. Wasserman, "Netconf configuration protocol (rfc 4741)," Dec 2006. [Online]. Available: http://tools.ietf.org/html/rfc4741

[64] S. Halle, R. Deca, O. Cherkaoui, R. Villemaire, and D. Puche, "A formal validation model for the netconf protocol," *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, pp. 147–158, 2004.

[65] M. Choi, H. Choi, J. Hong, H. Ju, and S. POSTECH, "Xml-based configuration management for ip network devices," *Communications Magazine, IEEE*, vol. 42, no. 7, pp. 84–91, 2004.

[66] C. Systems, "Network configuration protocol," Jun 2009. [Online]. Available: http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_cns_netconf.pdf

[67] J. Networks, "Netconf api guide," 2008. [Online]. Available: http://www.juniper.net/techpubs/software/junos/junos91/netconf-guide/netconf-guide.pdf

[68] G. Munz, A. Antony, F. Dressler, and G. Carle, "Using netconf for configuring monitoring probes," in *IEEE/IFIP Network Operations & Management Symposium (IEEE/IFIP NOMS 2006), Poster Session, Vancouver, Canada, Apr*, 2006.

[69] A. Wool, "Architecting the lumeta firewall analyzer," in *Proceedings of the 10th conference on USENIX Security Symposium*, USENIX Association Berkeley, CA, USA. USENIX Association, 2001, pp. 7–7.

[70] L. W. Wai, "Sans security life cycle," 2001. [Online]. Available: http://www.sans.org/reading_room/whitepapers/testing/security_life_cycle_1_diy_assessment_260?show=260.php&cat=testing

[71] Algosec, "Algosec firewall analyser," 2009. [Online]. Available: http://www.algosec.com/en/products/firewall_analyzer.php

[72] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Integrated Network Management*, 2003, p. 17Ű30.

[73] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, "Towards global verification and analysis of network access control configuration," DePaul University, Chicago, IL, USA, Tech. Rep., 2008.

[74] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "Fireman: A toolkit for firewall modeling and analysis," *Security and Privacy, IEEE Symposium on*, vol. 0, pp. 199–213, 2006.

[75] D. Caldwell, S. Lee, and Y. Mandelbaum, "Learning to talk cisco ios: Inferring the ios command language from router configuration data," AT&T, Tech. Rep., 2007.

[76] E. Al-Shaer and H. Hamed, "Management and translation of filtering security policies," in *2003 IEEE International Conference on Communications*. IEEE Press, 2003.

[77] A. Tongaonkar, N. Inamdar, and R. Sekar, "Inferring higher level policies from firewall rules," in *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–10.

[78] K. Golnabi, R. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, 2006.

[79] T. Tran, E. Al-Shaer, and R. Boutaba, "Policyvis: firewall security policy visualization and inspection," in *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–16.

[80] D. Botta, R. Werlinger, A. Gagné, K. Beznosov, L. Iverson, S. Fels, and B. Fisher, "Towards understanding it security professionals and their tools," in *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security*. New York, NY, USA: ACM, 2007, pp. 100–111.

[81] L. Saliou, W. J. Buchanan, J. Graves, and J. Munoz, "Novel framework for automated security abstraction, modelling, implementation and verification," in *4th European Conference on Information Warfare and Security*, 2005, pp. 303–312.

[82] L. Saliou, "Network firewall dynamic performance evaluation and formalisation," Ph.D. dissertation, Edinburgh Napier University, 2009.

[83] N. Damianou, "A policy framework for management of distributed systems," Ph.D. dissertation, Imperial College, 2002.

[84] W. W. W. Consortium, "Extensible markup language (xml)," 2000. [Online]. Available: http://www.w3.org/TR/2000/REC-xml-20001006#sec-origin-goals

[85] E. T. Ray, *Learning XML*, 2nd ed., S. St.Laurent, Ed. O'Reilly Media, Inc., Sep 2003. [Online]. Available: http://oreilly.com/catalog/9780596004200/index.html

[86] M. Fitzgerald, *Learning XSLT*. O'Reilly Media, Inc., Nov 2003. [Online]. Available: http://oreilly.com/catalog/9780596003272/

[87] J. Sedayao, *Cisco IOS access lists*. O'Reilly Media, Inc., 2001.

[88] R. Barrett, E. Kandogan, P. Maglio, E. Haber, L. Takayama, and M. Prabaker, "Field studies of computer system administrators: analysis of system management tools and practices," in *Proceedings of the 2004 ACM conference on computer supported cooperative work*. ACM New York, NY, USA, 2004, pp. 388–395.

[89] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[90] T. Parsons, *Introduction to compiler construction*. Computer Science Press, Inc. New York, NY, USA, 1992.

[91] J. E. F. Friedl, *Mastering Regular Expressions, 2nd Edition*. O'Reilly Media, Inc., Aug 2006. [Online]. Available: http://oreilly.com/catalog/9780596528126/

[92] C. Systems, "Cisco ios software releases 12.4 command references," 2008.

[93] J. Levine, T. Mason, and D. Brown, *lex & yacc*. O'Reilly Media, Inc., Oct 1992. [Online]. Available: http://oreilly.com/catalog/9781565920002/index.html

[94] N. Lehmann, R. Schwarz, and J. Keller, "Firecrocodile: A checker for static firewall configurations."

Acronyms

AFPL	**A**bstract **F**irewall **P**olicy language
ACL	**A**ccess **C**ontrol **L**ist
BERR	Department for **B**usiness **E**nterprise & **R**egulatory **R**eform
BDD	**B**inary **D**ecision **D**iagrams
CIM	**C**ommon **I**nformation **M**odel
COBIT	**C**ontrol **OB**jectives for **I**nformation and Related **T**echnology
COPS	**C**ommon **O**pen **P**olicy **S**ervice
CLI	**C**ommand **L**ine **I**nterface
CRIP	**C**isco **R**outer **I**OS **P**arser
CRIPE	**C**isco **R**outer **I**OS **P**arser **E**valuation Engine
CSPM	**C**isco **S**ecure **P**olicy **M**anager
DMZ	**D**e**M**ilitarised **Z**one
FANG	**F**irewal **AN**alysis en**G**ine
FPA	**F**irewall **P**olicy **A**dvisor
GUI	**G**raphical **U**ser **I**nterface
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment
IDS	**I**ntrusion **D**etection **S**ystem
IDPS	**I**ntrusion **D**etection and **P**revention **S**ystem
IGMP	**I**nternet **G**roup **M**anagement **P**rotocol
IOS	**I**nternetwork **O**perating **S**ystem
IETF	**I**nternet **E**ngineering **T**ask **F**orce

IP	**I**nternet **P**rotocol
MDL	**M**odel **D**efinition **L**anguage
MDD	**M**ulti-way **D**ecision **D**iagram
MIB	**M**anagement **I**nformation **B**ase
NIDS	**N**etwork **I**ntrusion **D**etection **S**ystem
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NPT	**N**etwork **P**olicy **T**ool
NSA	**N**ational **S**ecurity **A**gency
OS	**O**perating **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection
SAFE	**S**ecure **A**rchitecture **F**or **E**nterprise
SDLC	**S**ystem **D**evelopment **L**ife **C**ycle
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SSH	**S**ecure **SH**ell
TCP	**T**ransport **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
VPN	**V**irtual **P**rivate **N**etwork
XML	e**X**tensible **M**arkup **L**anguage
XSL	e**X**tensible **S**tylesheet **L**anguage
XSLT	e**X**tensible **S**tylesheet **L**anguage **T**ransformations

Appendix A

Project Management

NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rich Macfarlane
Supervisor: Prof. Bill Buchanan
Date: 21/08/2009
Last diary date: 24/07/2009

Objectives:

- Draft copy of literature review completed.
- Experiments completed.
- Changes to literature review made.

Progress:

- Draft literature review complete.
- Experiments almost complete - problems with stress testing experiments to be discussed.

Supervisor's Comments:

- Changes recommended for literature review.
- Consider reasons behind stress test failure, as part of evaluation.

Appendix B

CRIP Tool Source Code

Listing B.1 – CRIP tool CiscoIOS class

```
1  //#define TRACE
2  using System;
3  using System.Diagnostics;
4  using System.Threading;
5  using System.Text.RegularExpressions;
6  using System.Collections.Specialized;
7
8  namespace CiscoConfigParser
9  {
10
11     /// <summary>
12     /// CiscoIOS - Temp Shell Class to call Modules from.
13     /// </summary>
14     public class CiscoIOS
15     {
16         // Cisco Command Modes
17         // TBA: stuct/static class for these
18         const int GLOBAL_CONFIG = 0;
19         const int SUCCESS = 0;
20
21
22         /// <summary>
23         /// Parse a Cisco IOS Configuration.
24         /// TBA:
25         ///     - [-o outputFile] [-t traceLogFile]
26         /// </summary>
27         [STAThread]
28         static int Main(string[] args)
29         {
30             StringDictionary Params;
31             Params = new StringDictionary();
32             string sParam = null;
33
34             Console.WriteLine( "Cisco Router IOS Parser (CRIP) ...");
35
36             // Parse Arguements.
37             string[] sParts;
38             foreach (string arg in args)
```

```
39              {
40                  // Split each cmd line arg.
41                  sParts = Regex.Split(arg, @"^-{1,2}", RegexOptions.IgnoreCase);
42                  switch (sParts.Length)
43                  {
44                      case 1:
45                          // A Parameter Value.
46                          if (sParam != null)
47                          {
48                              // Set param value.
49                              if (!Params.ContainsKey(sParam))
50                                  Params.Add(sParam, sParts[0]);
51                              sParam = null;
52                          }
53                          break;
54                      case 2:
55                          // A Parameter.
56                          if (sParam != null)
57                          {
58                              // Previous param has no value, (bool params) set it
59                              // to 'true'
60                              if (!Params.ContainsKey(sParam))
61                                  Params.Add(sParam, "true");
62                          }
63                          sParam = sParts[1];
64                          break;
65                  }
66              }
67              // Previous param has no value, (bool params) set it
68              // to 'true'
69              if (sParam != null)
70              {
71                  if (!Params.ContainsKey(sParam))
72                      Params.Add(sParam, "true");
73              }
74
75              string configFile = "";
76              if (Params.ContainsKey("c"))
77                  configFile = Params["c"];
78              else
79              {
80                  Console.WriteLine("No file to parse...");
81                  Environment.Exit(1);
82              }
83              bool bVerbose = false;
84              if (Params.ContainsKey("v"))
85              {
86                  if (Params["v"] == "true")
87                      bVerbose = true;
88              }
89              bool bQuiet = false;
```

```
90              if (Params.ContainsKey("q"))
91              {
92                  if (Params["q"] == "true")
93                      bQuiet = true;
94              }
95
96              Console.WriteLine("Parsing: " + configFile);
97
98              int[] outs = new int[2];
99              try
100             {
101                 outs = CiscoIOS.Parse(configFile, bVerbose, bQuiet);
102             }
103             catch (Exception e)
104             {
105                 Console.WriteLine("Exception: {0}", e.Message);
106                 Environment.Exit(2);
107             }
108             Console.WriteLine("Parsed " + outs[0].ToString() + " lines, with "
109                               + outs[1].ToString() + " errors reported." );
110
111             //Console.WriteLine("\nPress Enter to Exit");
112             return (0);
113         }// end of Main()
114
115
116         /// <summary>
117         /// Parses Cisco Router IOS Commands, calling Handler Classes Parse
118         /// methods to deal with different IOS commands.
119         ///
120         /// </summary>
121         /// <returns>
122         ///     0   Success.
123         ///     1
124         /// </returns>
125         public static int[] Parse(
126             string configFile,  // Input config file to be parsed.
127             bool bVerbose,      // Verbose error output or not.
128             bool bQuiet         // Error file written or not
129             )
130         {
131             string line = "";   // Current config line.
132             int iLinesParsed = 0;
133             int iErrors = 0;
134             int cmdMode = GLOBAL_CONFIG;
135
136             // Set up Input stream.
137             Lexer lexer = new Lexer(configFile);
138             // Set up Output streams.
139             ErrorLog errorLog = new ErrorLog( bQuiet);
140             TraceLog traceLog = new TraceLog();
```

```
141             traceLog.WriteLine("[CiscoIOS.Parse] starts");
142 
143             //
144             // Parse config file.
145             //
146             try
147             {
148                 // Get the next line of the config.
149                 while ((line = lexer.ReadLine()) != null)
150                 {
151                     // Parse IOS commands.
152 
153                     // Check if class can handle command, if so call handler to
                          Parse cmd line.
154                     if (Regex.IsMatch(line, CiscoIOSACL.GetCmds()))
155                     {
156                         CiscoIOSACL ACL = new CiscoIOSACL(lexer, errorLog,
                              traceLog, bVerbose);
157                         if (ACL.ParseCmd(ref cmdMode) != SUCCESS)
158                             iErrors++;
159                     }
160                     else
161                     {
162                         // Command not implemented.
163                         traceLog.WriteLine("\n[CiscoIOS.Main] %UNKNOWNCOMMAND: "
                               + line);
164                         errorLog.WriteLine("%UNKNOWNCOMMAND", lexer, bVerbose);
165                         iErrors++;
166                     }
167                     iLinesParsed++;
168                 }
169             }
170             catch (Exception e)
171             {
172                 traceLog.WriteLine("[CiscoIOS.Parse] " + e.Message);
173                 throw e;
174             }
175 
176 #if TRACE
177             traceLog.WriteToConsole();
178 #endif
179             //
180             // Write to output file.
181             //
182             errorLog.WriteToFile(configFile + ".errors", configFile, iLinesParsed)
                  ;
183 
184             int[] outValues = {iLinesParsed,iErrors};
185             return (outValues);
186         } // end of Parse Method
187 
```

```
188     }//end of Class
189 }
```

Listing B.2 – CRIP tool CiscoIOSACL class

```
1   using System;
2   using System.IO;
3   using System.Text.RegularExpressions;
4   using System.Collections;
5
6   namespace CiscoConfigParser
7   {
8       /// <summary>
9       /// Description:
10      ///     Class to parse/validate Cisco IOS Access Control List Commands.
11      ///     Based on Cisco IOS v12.4
12      ///
13      /// Properties:
14      ///     Public:
15      ///
16      ///     Private:
17      ///         lexer
18      ///         errorLog
19      ///         traceLog
20      ///
21      /// Methods:
22      ///     Public:
23      ///         ParseCmd
24      ///
25      ///     Private:
26      ///         ParseStdACLRule
27      ///         ParseExtACL
28      ///         ParseExtACLRule
29      ///         ParseNamed
30      /// Notes:
31      ///     TBA:
32      ///     - check protocols patterns against what router does - in depth
33      ///     - IP Addr/Wildcard Mask Validation improved? -Bills isMask() routine?
34      ///     - Error Msgs passed back up to ACL level, or written to output file.
            Not written to console window from
35      ///       low level methods. Normal mode should just report an error on line n
             of config. -v verbose mode could
36      ///       spit out extra stuff?
37      ///     - Src/Dest parser procs into one
38      ///     - remark command - ACL & ACE levels
39      ///     - fragments param not always available e.g. use "tcp ... eq finger"
            then not available (on router)
40      ///     - if we want to parse like the Cisco IOS, then we need to take at
            least optional params in any order!
41      ///
42      /// Admendment History:
43      ///     1.  Aug-2008    RJM     Release ?.??
```

```
44      ///         Created.
45      /// </summary>
46      public class CiscoIOSACL
47      {
48          private Lexer       lexer;      // input stream; for now a configuration
                txt file.
49          private TraceLog    traceLog;   // output trace stream.
50          private ErrorLog    errorLog;   // output error stream.
51          private string      line;       // current line of consig.
52          private bool        bVerbose;   // Verbose Error logging or not.
53          private bool        bQuiet;     // Write to an Error logging file or not.
                Quiet=true, just return total
54                                          // lines parsed & no. of errors.
55
56          //private static ArrayList errorList = new ArrayList();
57
58          // Output list of abstract filter rules.
59          private PacketFilter PacketFilet;
60
61          // tmp
62          private string      ACLName;
63          private string      DynamicACLName;
64          private string      Action;
65          private string      Protocol;
66          private string      SourceIP;
67          private string      SourceWildcard;
68          private string      DestIP;
69          private string      DestWildcard;
70          private bool        Log;
71
72          // Cisco Command Modes
73          // TBA: stuct/static class for these
74          const int   GLOBAL_CONFIG   = 1;
75          const int   ACL_CONFIG_STD  = 11;
76          const int   ACL_CONFIG_EXT  = 12;
77
78          // Commands class can parse.
79          const string ACCESS_LIST    = @"^access-list\s+";
80          const string IP_ACCESS_LIST = @"^ip access-list\s+";
81          const string PERMIT         = @"^permit\s+";
82          const string DENY           = @"^deny\s+";
83          const string REMARK         = @"^remark\s+";
84          const string COMMANDS       = ACCESS_LIST+"|"+IP_ACCESS_LIST+"|"+PERMIT+"|
                "+DENY;
85
86          // Patterns to match TOKENS
87          const string WORD           = @"^[a-zA-Z]+$"; //{1,5} 1 to 5 chars long
88          const string NUMBER         = @"^[0-9]+$";
89          const string ALPHANUMERIC   = @"^[\w.]+$";
90          const string ALPHASTART     = @"^\w+.*$";
91          // NO QUOTES?? ^' ^"?
```

```
92          //                          = @"^[a-zA-Z0-9]+$";
93
94          const string IP_STD_ACL     = @"^([1-9]|[1-9][0-9]|1[3-9][0-9][0-9])$"; //
                IP standard numbered. 1-99 or 1300-1999.
95          const string IP_EXT_ACL     = @"^(1[0-9][0-9]|2[0-6][0-9][0-9])$"; // IP
                extended numbered. 100-199 or 2000-2699.
96          const string STANDARD       = @"^standard$";
97          const string EXTENDED       = @"^extended$";
98          //const string IP_ACL_TYPE  = @"^(standard|extended)$";
99          const string IP_ACL_TYPE    = STANDARD+"|"+EXTENDED;
100
101         const string ACTION         = @"^(permit|deny)$";
102
103         // protocol 0-255|named protocol
104         const string PROTOCOLS      = @"^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]|ahp
                |eigrp|esp|gre|icmp|igmp|ip|ipinip|nos|ospf|pcp|pim|tcp|udp)$";
105         //const string PROTOCOLS        = @"^(ahp|eigrp|esp|gre|icmp|igmp|ip|
                ipinip|nos|ospf|pcp|pim|tcp|udp)$";
106         const string ICMP           = @"^icmp$";
107         const string IGMP           = @"^igmp$";
108         const string TCP            = @"^(tcp|6)$";
109         const string UDP            = @"^(udp|17)$";
110
111         // IpAddress 0-255.0-255.0-255.0-255
112         const string IPv4ADDRESS    = @"^(([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
                \.){3}"
113                                     +   @"([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])$"
                                            ;
114         //const string MaskCIDR     = @"^([0-2]?[0-9]|3[0-2])$";
115         //const string IPv4ADDRESS  = @"^([0-2]?[0-5]?[0-5]\.){3}[0-2]?[0-5]?[0-5]
                $";
116
117         // ports
118         const string PORT_OPERATORS = @"^(lt|gt|eq|neq|range)$";
119         const string TCP_PORT_NAMES = @"^(bgp|chargen|cmd|daytime|discard|domain|
                echo|finger|ftp|ftp-data|gopher|hostname|ident|irc|klogin|kshell|lpd|
                nntp|pop2|pop3|smtp|sunrpc|syslog|tacacs|talk|telnet|time|uucp|whois|
                www)$";
120         const string UDP_PORT_NAMES = @"^(biff|bootpc|bootps|discard|dnsix|domain|
                echo|mobile-ip|nameserver|netbios-dgm|netbios-ns|non500-isakmp|ntp|
                rip|snmp|snmptrap|sunrpc|syslog|tacacs-ds|talk|tftp|time|who|xdmcp)$"
                ;
121
```

```
122         const string ICMP_NAME      = @"^(administratively-prohibited|alternate-
                address|conversion-error|dod-host-prohibited|dod-net-prohibited|echo|
                echo-reply|general-parameter-problem|host-isolated|host-precedence-
                unreachable|host-redirect|host-tos-redirect|host-tos-unreachable|host
                -unknown|host-unreachable|information-reply|information-request|mask-
                reply|mask-request|mobile-redirect|net-redirect|net-tos-redirect|net-
                tos-unreachable|net-unreachable|network-unknown|no-room-for-option|
                option-missing|packet-too-big|parameter-problem|port-unreachable|
                precedence-unreachable|protocol-unreachable|reassembly-timeout|
                redirect|router-advertisement|router-solicitation|source-quench|
                source-route-failed|time-exceeded|timestamp-reply|timestamp-request|
                traceroute|ttl-exceeded|unreachable)$";
123         const string ICMP_TYPE      = @"^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])$";
                // icmp-type 0-255
124         const string ICMP_CODE      = @"^([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])$";
                // icmp-code 0-255
125
126         // igmp-type 0-15|named
127         const string IGMP_TYPE      = @"^([0-9]|1[0-5]|dvmrp|host-query|host-
                report|pim|trace)$";
128
129         // Precedence 0-7|named precedence
130         const string PRECEDENCE     = @"^([0-7]|critical|flash|flash-override|
                immediate|internet|network|priority|routine)$";
131         // DSCP 0-63|named dscp
132         const string DSCP           = @"^([0-5]?[0-9]|6[0-3]|default|ef|af11|af12|
                af13|af21|af22|af23|af31|af32|af33|af41|af42|af43|cs1|cs2|cs3|cs4|cs5
                |cs6|cs7)$";
133
134         // protocol 0-15|named tos
135         const string TOS            = @"^([0-9]|1[0-5]|max-reliability|max-
                throughput|min-delay|min-monetary-cost|normal)$";
136         const string LOG            = @"^(log|log-input)$";
137
138         // Common Return Values for private methods
139         const   int SUCCESS         = 0;
140         const   int NO_TOKENS       = 1;
141         const   int INVALID         = 2;
142         const   int PARSE_ERROR     = 3;
143
144
145         /// <summary>
146         /// Constructor for Cisco IOS ACL Parser Class.
147         /// </summary>
148         public CiscoIOSACL(
149             Lexer lexer,
150             ErrorLog errorLog,
151             TraceLog traceLog,
152             bool bVerbose
153             )
154         {
```

```
155             this.lexer = lexer;
156             this.errorLog = errorLog;
157             this.traceLog = traceLog;
158             this.line = lexer.line;
159             this.bVerbose = bVerbose;
160         }
161         /// <summary>
162         /// Returns string containing patterns of cmds this class can parse.
163         /// </summary>
164         public static string GetCmds()
165         {
166             return COMMANDS;
167         }
168 
169 
170         /// <summary>
171         /// Parses Cisco IOS ACL Commands - {access-list, ip access-list, permit,
                deny} (Standard,Extended,Named)
172         /// Check the input is well formed, and validate to similar extent as the
                Cisco IOS parser does.
173         ///
174         /// </summary>
175         /// <returns>
176         ///     1   Incomplete command params
177         ///     2   Invalid command
178         ///     3   Error in line.
179         /// </returns>
180         public int ParseCmd(
181             ref int     cmdMode     // Current Cisco command mode
182             )
183             // prob' want to pass in Security Policy object for the current
184             // device, so we can add PacketFilter obj's to it.
185         {
186             int idx;    // current token index
187 
188             try
189             {
190                 // TBA: Need to change to deal with sub-commands? Based on
191                 // indent? Lexer can do all that
192                 string[] tokens = Lexer.GetTokens(line);
193                 idx = 0; // first token
194                 //
195                 // Match ACL Commands
196                 //
197                 if ( Regex.IsMatch(line, ACCESS_LIST) )
198                 {
199                     //
200                     // Numbered acl-name.
201                     //
202                     idx++; // next token.
203                     if ( idx == tokens.Length )
```

```
204                     {
205                         // no more tokens
206                         traceLog.WriteLine("\n[ACL.ParseCmd] - " + line);
207                         traceLog.WriteLine("[ACL.ParseCmd] % Incomplete Command -
                                Expecting acl-name" );
208                         errorLog.WriteLine("% Incomplete Command - Expecting acl-
                                name", lexer, "EOL", bVerbose);
209                         return 1;
210                     }
211                     if ( Regex.IsMatch(tokens[idx], IP_STD_ACL) )
212                     {
213                         //
214                         // IP Standard.
215                         //
216                         this.ACLName = tokens[idx];
217                         traceLog.WriteLine("\n[ACL.ParseCmd] Std ACL - " + line);
218                         if (this.ParseStdACLRule(tokens, idx) == SUCCESS)
219                         {
220                             cmdMode = GLOBAL_CONFIG;
221                             // On change of ACL create new ACL,
222                             // Or Create new ACL Line.
223                         }
224                         else
225                         {
226                             // Error on line.
227                             return PARSE_ERROR;
228                         }
229                     }
230                     else if ( Regex.IsMatch(tokens[idx], IP_EXT_ACL) )
231                     {
232                         //
233                         // IP Extended.
234                         //
235                         this.ACLName = tokens[idx];
236                         traceLog.WriteLine("\n[ACL.ParseCmd] Extd ACL " + line);
237                         if ( this.ParseExtACL(tokens, idx) == SUCCESS )
238                         {
239                             cmdMode = GLOBAL_CONFIG;
240                             // On change of ACL create new ACL,
241                             // Or Create new ACL Line.
242                         }
243                         else
244                         {
245                             // Error on line.
246                             return PARSE_ERROR;
247                         }
248                     }
249                     else
250                     {
251                         traceLog.WriteLine("\n[ACL.ParseCmd] % Invalid Numbered
                                ACL Id: " + line);
```

```
252                     errorLog.WriteLine("% Invalid Numbered ACL Id ", lexer,
                            tokens[idx] , bVerbose );
253                     return 2;
254                 }
255             }
256             //else if ( tokens[idx] == "ip" && tokens[++idx] == "access-
                    list")
257             else if ( Regex.IsMatch(line, IP_ACCESS_LIST) )
258             {
259                 idx++; // this is "access-list"
260                 //
261                 // Named ACL.
262                 //
263                 traceLog.WriteLine("\n[ACL.ParseCmd] Named ACL - " + line);
264                 if ( this.ParseNamedACL(tokens, idx, ref cmdMode) == SUCCESS )
265                 {
266                     //
267                     //traceLog.WriteLine("\n[ACL.ParseCmd] Named ACL: " + line
                            );
268                 }
269             else if ( (Regex.IsMatch(line, PERMIT) || Regex.IsMatch(line, DENY
                    )) && cmdMode == ACL_CONFIG_STD )
270             {
271                 //
272                 // IP Standard Named ACL.
273                 //
274                 traceLog.WriteLine("\n[ACL.ParseCmd] Named Std ACE - " + line)
                        ;
275                 if ( this.ParseStdACLRule(tokens, --idx) == SUCCESS )
276                 {
277                     // Create new ACL Line.
278                 }
279             }
280             else if ( (Regex.IsMatch(line, PERMIT) || Regex.IsMatch(line, DENY
                    )) && cmdMode == ACL_CONFIG_EXT )
281             {
282                 //
283                 // IP Standard Named ACL.
284                 //
285                 traceLog.WriteLine("\n[ACL.ParseCmd] Named Extd ACE - " + line
                        );
286                 if ( this.ParseExtACLRule(tokens, --idx) == SUCCESS )
287                 {
288                     // Create new ACL Line.
289                 }
290             }
291             //
292             // TBA: "access-list remark" and "remark" cmds
293             //
294             else
295             {
```

```
296                 // Invalid Command/not implemented.
297                 traceLog.WriteLine("\n[ACL.ParseCmd] - " + line);
298                 traceLog.WriteLine("[ACL.ParseCmd] % Invalid Input Detected" )
                        ;
299                 errorLog.WriteLine("% Invalid Input Detected", lexer, bVerbose
                        );
300                 return 2;
301             }
302         }
303         catch(Exception e)
304         {
305             traceLog.WriteLine( "[ACL.ParseCmd] " + e.Message );
306             errorLog.WriteLine( e.Message, lexer , bVerbose );
307         }
308         return SUCCESS;
309     }
310 
311 
312     /// <summary>
313     /// Parse an IP Extended ACL.
314     /// This method parses the header, i.e. the bit before the action:
315     ///     access-list acl-name [dynamic dynamic-name [timeout timeout-mins]]
316     ///
317     /// </summary>
318     /// <param name="tokens"></param>
319     /// <param name="idx"></param>
320     /// <returns>
321     ///     0   Success.
322     ///     1   No tokens left
323     ///     2   Invalid Dynamic ACL params.
324     ///     3   Invalid Rule
325     /// </returns>
326     public int ParseExtACL( string[] tokens, int idx )
327     {
328         //
329         // Optional [dynamic dynamic-name [timeout timeout-mins]] params.
330         //
331         int ret = this.ParseDynamic(tokens, ref idx ) ;
332         if ( ret == NO_TOKENS )
333         {
334             traceLog.WriteLine("[ParseExtACL] % Incomplete Command" );
335             errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                    ;
336             return NO_TOKENS;
337         }
338         else if ( ret == 3 || ret==6 )
339         {
340             // No Tokens, after consuming valid dynamic|timeout param.
341             traceLog.WriteLine("[ParseExtACL] % Incomplete Command" );
342             errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                    ;
```

```
343                 return NO_TOKENS;
344             }
345             else if( ret == 4 || ret == 7 )
346             {
347                 // Invalid dynamic-name|timeout-mins, after consuming valid
                        dynamic|timeout keyword.
348                 return 2;
349             }
350             //
351             // Parse Extended ACL Rule.
352             //
353             if ( this.ParseExtACLRule(tokens, idx) != SUCCESS )
354             {
355                 return 3;
356             }
357             //
358             // Create new Abstract Packet Filtering Rule Object.
359             // Attach it to the Current Devices Security Policy object.
360             //
361             //PacketFilterRule rule = new PacketFilterRule( this.Action, this.
                    Protocl etc );
362             //PacketFilter.Rules.Add( rule );
363 
364             return SUCCESS;
365         }
366 
367 
368         /// <summary>
369         /// Parse the Optional [dynamic dynamic-name [timeout timeout-mins]]
                Params.
370         /// </summary>
371         /// <param name="tokens"></param>
372         /// <param name="idx"></param>
373         /// <returns>
374         ///     0   Success
375         ///     1   No Tokens
376         ///     2   Invalid dynamic param
377         ///     3   No Tokens, No dynamic-name
378         ///     4   Invalid dynamic-name value, after consuming valid dynamic
                param
379         ///     5   Invalid timeout param
380         ///     6   No Tokens, No timeout-mins
381         ///     7   Invalid timeout-mins value, after consuming valid timeout
                param
382         /// </returns>
383         private int ParseDynamic( string[] tokens, ref int idx )
384         {
385             //
386             // Optional [dynamic] param.
387             //
388             idx++; // next token
```

```
389             if ( idx == tokens.Length )
390             {
391                 return NO_TOKENS;
392             }
393             if ( tokens[idx] != "dynamic" )
394             {
395                 idx--; // not dynamic param
396                 return 2;
397             }
398             // TBA: Check dynamic ACL not defined already for this ACL. (1 at most
                    allowed)
399             //
400             // dynamic-name
401             //
402             idx++; // next token
403             if ( idx == tokens.Length )
404             {
405                 return 3; // no tokens after valid dynamic param consumed
406             }
407             if ( Regex.IsMatch(tokens[idx], ALPHASTART) )
408             {
409                 // Valid dynamic-name.
410                 traceLog.WriteLine("[ACL.ParseDynamic] valid dynamic ACL: " +
                        tokens[idx]);
411                 this.DynamicACLName = tokens[idx];
412             }
413             else
414             {
415                 // Not a valid dynamic-name.
416                 traceLog.WriteLine("[ACL.ParseDynamic] % Invalid Input Detected -
                        dynamic-name: " + tokens[idx]);
417                 errorLog.WriteLine("% Invalid Input Detected - dynamic-name ",
                        lexer, tokens[idx] , bVerbose );
418                 return 4;
419             }
420
421             //
422             // Optional [timeout timeout-mins] param (timeout-value in minutes
                    1-9999).
423             //
424             idx++; // next token
425             if ( idx == tokens.Length )
426             {
427                 return NO_TOKENS;
428             }
429             if ( tokens[idx] != "timeout" )
430             {
431                 idx--; // not timeout param
432                 return 5;
433             }
434             //
```

```
435             // timeout-mins
436             //
437             idx++; // next token
438             if ( idx == tokens.Length )
439             {
440                 return 6; // no tokens after valid dynamic param consumed
441             }
442             if ( Regex.IsMatch(tokens[idx], NUMBER) &&
443                 (Int32.Parse(tokens[idx]) >=1 && Int32.Parse(tokens[idx]) <=9999)
                        )
444             {
445                 // Valid timeout-mins.
446                 traceLog.WriteLine("[ACL.ParseDynamic] valid timeout: " + tokens[
                        idx]);
447             }
448             else
449             {
450                 // Not a timeout-mins value.
451                 traceLog.WriteLine("[ACL.ParsePorts] % Invalid Input Detected -
                        timeout: " + tokens[idx]);
452                 errorLog.WriteLine("% Invalid Input Detected - timeout ", lexer,
                        tokens[idx], bVerbose );
453                 return 7;
454             }
455             // TBA: Store dynamic params?
456             return SUCCESS;
457         }
458
459
460
461         /// <summary>
462         /// Parse a Named ACL.
463         /// This method, first parses the header, then calls parseStdACLRule() or
                parseExtACLRule() for each line.
464         ///     ip access-list acl-type:{standard|extended} acl-name
465         ///
466         /// </summary>
467         /// <param name="tokens"></param>
468         /// <param name="idx"></param>
469         /// <returns>
470         ///     0   Success.
471         ///     1   No tokens left
472         ///     2   Invalid acl-type or acl-name, or invalid token at end of line.
473         /// </returns>
474         public int ParseNamedACL( string[] tokens, int idx, ref int cmdMode )
475         {
476             //
477             // acl-type
478             //
479             idx++; // next token
480             if ( idx == tokens.Length )
```

```
481             {
482                 traceLog.WriteLine("[ACL.ParseNamedACL] % Incomplete Command -
                        Expecting acl-type" );
483                 errorLog.WriteLine("% Incomplete Command - Expecting acl-type",
                        lexer, "EOL", bVerbose );
484                 return NO_TOKENS;
485             }
486             if ( !Regex.IsMatch(tokens[idx], IP_ACL_TYPE) )
487             {
488                 traceLog.WriteLine("[ACL.ParseNamedACL] % Invalid Input Detected:
                        " + tokens[idx]);
489                 errorLog.WriteLine("% Invalid Input Detected ", lexer, tokens[idx
                        ], bVerbose );
490                 return INVALID;
491             }
492             string aclType = tokens[idx];
493 
494             //
495             // acl-name
496             //
497             idx++; // next token
498             if ( idx == tokens.Length )
499             {
500                 traceLog.WriteLine("[ACL.ParseNamedACL] % Incomplete Command -
                        Expecting acl-name" );
501                 errorLog.WriteLine("% Incomplete Command - Expecting acl-name",
                        lexer, "EOL", bVerbose);
502                 return NO_TOKENS;
503             }
504             if ( !Regex.IsMatch( tokens[idx], ALPHASTART) ) // TBA: add numbered
                     acl-name checks
505             {
506                 traceLog.WriteLine("\n[ACL.ParseNamedACL] % Invalid Input Detected
                        : acl-name: " + tokens[idx] );
507                 errorLog.WriteLine("% Invalid Input Detected ",lexer ,tokens[idx]
                        , bVerbose );
508                 return INVALID;
509             }
510             this.ACLName = tokens[idx];
511 
512             //
513             // Check for invalid token at end of line.
514             //
515             if ( ++idx < tokens.Length )
516             {
517                 traceLog.WriteLine("[ACL.ParseNamedACL] % Invalid Input Detected:
                        " + tokens[idx] );
518                 errorLog.WriteLine("% Invalid Input Detected ",lexer ,tokens[idx]
                        , bVerbose );
519                 return INVALID;
520             }
```

```
521             // Valid Named ACL hdr parsed.
522             if ( Regex.IsMatch( aclType, STANDARD) )
523                 cmdMode = ACL_CONFIG_STD;
524             else if ( Regex.IsMatch( aclType, EXTENDED) )
525                 cmdMode = ACL_CONFIG_EXT;
526             //
527             // Create new ACL obj.
528             // Loop until end/change of acl
529             //  {
530             //      Parse optional Seq No.
531             //      Parse each Line ( Std or Extd )
532             //      Create ACL Line objs
533             //  }
534             //
535             return SUCCESS;
536         }
537 
538 
539         /// <summary>
540         /// Parse an IP Standard ACL Rule.
541         /// The Policy rule itself from the Action token onwards:
542         ///     access-list acl-name action:{permit|deny} source [source-wildcard-
                mask] [log]
543         ///
544         /// </summary>
545         /// <param name="tokens"></param>
546         /// <param name="idx"></param>
547         /// <returns>
548         ///     0   Success
549         ///     1   Invalid Action
550         ///     2   Invalid Src
551         ///     3   Invalid Log param
552         /// </returns>
553         private int ParseStdACLRule( string[] tokens, int idx )
554         {
555             //
556             // Action.
557             //
558             if ( this.ParseAction(tokens, ref idx ) != SUCCESS)
559             {
560                 return 1;
561             }
562             //
563             // Source - Network, Host or Range of IP Address.
564             //
565             if ( this.ParseSrc(tokens, ref idx, true ) != SUCCESS)
566             {
567                 return 2;
568             }
569             //
570             // Option [log] param
```

```
571             //
572             if ( this.ParseLog(tokens, ref idx ) == INVALID )
573             {
574                 // Invalid log param.
575                 traceLog.WriteLine("[ACL.ParseStdACLRule] % Invalid Input Detected
                        : log expected");
576                 errorLog.WriteLine("% Invalid Input Detected: log expected", lexer
                        , tokens[++idx] , bVerbose );
577                 return 3;
578             }
579             //
580             // Valid ACL Line.
581             // Create new Abstract Packet Filtering Rule Object.
582             //PacketFilterRule rule = new PacketFilterRule( this.Action, this.
                    Protocl etc );
583             //PacketFilter.Rules.Add( rule );
584 
585             return SUCCESS;
586         }
587 
588 
589         /// <summary>
590         /// Parse a an IP Extended ACL Rule.
591         /// The Policy rule itself, from the Action token onwards:
592         ///     action:{permit|deny} protocol source source-wildcard destination
                    destination-wildcard
593         ///     [precedence precedence-value | dscp dscp-value] [tos tos-value] [
                    log | log-input]
594         ///     [time-range time-range-name] [fragments]
595         /// </summary>
596         /// <param name="tokens"></param>
597         /// <param name="idx"></param>
598         /// <returns>
599         ///     0   Success
600         ///     1   NO_TOKENS, Invalid Rule
601         ///     2   Invalid Action
602         ///     3   Invlaid Protocol
603         ///     4   Invalid Src
604         ///     5   Invalid Src Port param
605         ///     6   Invalid Dest
606         ///     7   Invalid Dest Port Params
607         ///     8   Invalid ICMP type
608         ///     9   Invalid IGMP type
609         ///     10  Invalid precedence
610         ///     11  Invalid dscp
611         ///     12  Invalid tos
612         ///     13  Invalid time-range
613         ///     14  Invalid Fragment param
614         ///     15  Invalid token at end of line
615         /// </returns>
616         private int ParseExtACLRule( string[] tokens, int idx )
```

```
617         {
618             //
619             // Action.
620             //
621             if ( this.ParseAction(tokens, ref idx ) != SUCCESS )
622             {
623                 return 2;
624             }
625 
626             //
627             // Protocol - Internet Protocol.
628             //
629             if ( this.ParseProtocol(tokens, ref idx ) != SUCCESS )
630             {
631                 return 3;
632             }
633 
634             //
635             // Source - Network, Host or Range of IP Address.
636             //
637             if ( this.ParseSrc(tokens, ref idx, false ) != SUCCESS )
638             {
639                 return 4;
640             }
641 
642             //
643             // Optional Src [operator port] params; for UPD, TCP protocols.
644             //
645             int ret = this.ParsePorts(tokens, ref idx ) ;
646             if ( ret == NO_TOKENS )
647             {
648                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
649                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
650                 return NO_TOKENS;
651             }
652             else if ( ret == 3 )
653             {
654                 // No Tokens, after consuming valid operator.
655                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
656                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
657                 return NO_TOKENS;
658             }
659             else if( ret == 4 )
660             {
661                 return 5; // Invalid port no, after consuming valid operator.
662             }
663 
664             //
665             // Destination - Network, Host or Range of IP Address.
```

```
666             //
667             if ( this.ParseDest(tokens, ref idx, false ) != SUCCESS )
668             {
669                 return 6;
670             }
671 
672             //
673             // Optional Dest [operator port] params, for UPD; TCP protocols.
674             //
675             ret = this.ParsePorts(tokens, ref idx ) ;
676             if ( ret == NO_TOKENS )
677             {
678                 return SUCCESS; // no mand params left
679             }
680             else if ( ret == 3 )
681             {
682                 // No Tokens, after consuming valid operator.
683                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
684                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
685                 return NO_TOKENS;
686             }
687             else if( ret == 4 )
688             {
689                 return 7; // Invalid port no, after consuming valid operator.
690             }
691 
692             //
693             // Optional [established] param; for TCP protocol only.
694             //
695             if ( this.ParseEstablished(tokens, ref idx ) == NO_TOKENS )
696             {
697                 return SUCCESS; // no mand params left
698             }
699 
700             //
701             // Optional [icmp-type [icmp-code]|icmp-message] params; for ICMP
                    protocol only.
702             //
703             ret = this.ParseICMPType(tokens, ref idx ) ;
704             if ( ret == NO_TOKENS )
705             {
706                 return SUCCESS; // no mand params left
707             }
708             else if ( ret == 3 )
709             {
710                 // No Tokens, after consuming valid icmp-type.
711                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
712                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
713                 return NO_TOKENS;
```

```
            }
            else if( ret == 4 )
            {
                return 8; // Invalid icmp-code, after consuming valid icmp-type.
            }

            //
            // Optional [igmp-type igmp-type] params; for IGMP protocol only.
            //
            ret = this.ParseIGMPType(tokens, ref idx ) ;
            if ( ret == NO_TOKENS )
            {
                return SUCCESS; // no mand params left
            }
            else if ( ret == 3 )
            {
                // No Tokens, after consuming valid igmp keyword.
                traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
                errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                    ;
                return NO_TOKENS;
            }
            else if( ret == 4 )
            {
                return 9; // Invalid igmp-type, after consuming valid igmp keyword
                    .
            }

            //
            // Optional [precedence precedence-value | dscp dscp-value] params
            //
            ret = this.ParsePrecedence(tokens, ref idx ) ;
            if ( ret == NO_TOKENS )
            {
                return SUCCESS; // no mand params left
            }
            else if ( ret == 3 )
            {
                // No Tokens, after consuming valid precedence keyword.
                traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
                errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                    ;
                return NO_TOKENS;
            }
            else if( ret == 4 )
            {
                return 10; // Invalid precedence value, after consuming valid
                    precedence keyword.
            }
            else if( ret == 2 ) // not precedence keyword, so check for dscp.
            {
```

```
761                 //
762                 // Optional [dscp dscp-value] params
763                 //
764                 ret = this.ParseDSCP(tokens, ref idx ) ;
765                 if ( ret == NO_TOKENS )
766                 {
767                     return SUCCESS; // no mand params left
768                 }
769                 else if ( ret == 3 )
770                 {
771                     // No Tokens, after consuming valid dscp keyword.
772                     traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
773                     errorLog.WriteLine("% Incomplete Command", lexer, "EOL",
                            bVerbose);
774                     return NO_TOKENS;
775                 }
776                 else if( ret == 4 )
777                 {
778                     return 11; // Invalid dscp value, after consuming valid dscp
                        keyword.
779                 }
780             }
781 
782             //
783             // Optional [tos [tos]] params
784             //
785             ret = this.ParseTos(tokens, ref idx ) ;
786             if ( ret == NO_TOKENS )
787             {
788                 return SUCCESS; // no mand params left
789             }
790             else if ( ret == 3 )
791             {
792                 // No Tokens, after consuming valid precedence keyword.
793                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
794                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
795                 return NO_TOKENS;
796             }
797             else if( ret == 4 )
798             {
799                 return 12; // Invalid tos value, after consuming valid precedence
                        keyword.
800             }
801 
802             //
803             // Optional [log | log-input] param
804             //
805             if ( this.ParseLog(tokens, ref idx ) == NO_TOKENS )
806             {
807                 return SUCCESS; // no mand params left
```

```
808             }
809
810             //
811             // Optional [timerange timerange-name] param
812             //
813             ret = this.ParseTimeRange(tokens, ref idx ) ;
814             if ( ret == NO_TOKENS )
815             {
816                 return SUCCESS; // no mand params left
817             }
818             else if ( ret == 3 )
819             {
820                 // No Tokens, after consuming valid time-range keyword.
821                 traceLog.WriteLine("[ParseExtACLRule] % Incomplete Command" );
822                 errorLog.WriteLine("% Incomplete Command", lexer, "EOL", bVerbose)
                        ;
823                 return NO_TOKENS;
824             }
825             else if( ret == 4 )
826             {
827                 return 13; // Invalid time-range-name, after consuming valid time-
                        range keyword.
828             }
829
830             //
831             // Optional [fragments] param
832             //
833             if ( this.ParseFragments(tokens, ref idx ) == INVALID )
834             {
835                 return 14; // last param, must match or invalid
836             }
837             if ( ++idx < tokens.Length )
838             {
839                 traceLog.WriteLine("[ACL.ParseExtACLRule] % Invalid Input Detected
                        : " + tokens[idx] );
840                 errorLog.WriteLine("% Incomplete Command", lexer, tokens[idx] ,
                        bVerbose );
841                 return 15; // Invalid token at end of line.
842             }
843             //
844             // Valid ACL Line.
845             //
846             return SUCCESS;
847         }
848
849
850         /// <summary>
851         /// Parse the Action param.
852         /// </summary>
853         /// <param name="tokens"></param>
854         /// <param name="idx"></param>
```

```
855         /// <returns>
856         ///     0   Success
857         ///     1   No tokens
858         ///     2   Invalid Action
859         /// </returns>
860         private int ParseAction( string[] tokens, ref int idx )
861         {
862             int r = SUCCESS;
863             try
864             {
865                 idx++; // Next token.
866                 if ( idx == tokens.Length )
867                 {
868                     //traceLog.WriteLine("[ACL.ParseAction] % Incomplete Command -
                          Expecting {permit|deny}");
869                     //errorLog.WriteLine("% Incomplete Command - Expecting {permit
                         |deny}", lexer, tokens[idx] , bVerbose );
870                     //return NO_TOKENS; // no more tokens
871                     r = NO_TOKENS;
872                     throw(new ApplicationException( "% Incomplete Command -
                         Expecting {permit|deny}" ));
873                 }
874                 string action = tokens[idx];
875                 if ( Regex.IsMatch(tokens[idx], ACTION) )
876                 {
877                     traceLog.WriteLine("[ACL.ParseAction] valid Action: " + action
                         );
878                     this.Action = action;
879                 }
880                 else
881                 {
882                     //traceLog.WriteLine("[ACL.ParseAction] % Invalid Input
                         Detected: {permit|deny} expected");
883                     //errorLog.WriteLine("% Invalid Input Detected: {permit|deny}
                         expected", lexer, tokens[idx] , bVerbose );
884                     //return INVALID;
885                     r = INVALID;
886                     throw(new ApplicationException( "% Invalid Input Detected: {
                         permit|deny} expected" ));
887                 }
888 
889             }
890             catch(ApplicationException e)
891             {
892                 if ( r != SUCCESS )
893                 {
894                     traceLog.WriteLine( "[ACL.ParseAction] " + e.Message );
895                     string sToken = r == NO_TOKENS ? "EOL" : tokens[idx];
896                     errorLog.WriteLine( e.Message, lexer, sToken , bVerbose );
897                     return r;
898                 }
```

```
899             }
900             return SUCCESS;
901         }
902 
903 
904         /// <summary>
905         /// Parse the Protocol.
906         /// </summary>
907         /// <param name="tokens"></param>
908         /// <param name="idx"></param>
909         /// <returns>
910         ///     0   Success
911         ///     1   No Tokens
912         ///     2   Invalid Protocol
913         /// </returns>
914         private int ParseProtocol( string[] tokens, ref int idx )
915         {
916             idx++; // next token
917             if ( idx == tokens.Length )
918             {
919                 traceLog.WriteLine("\n[ParseProtocol] % Incomplete Command -
                        Protocol Expected" );
920                 errorLog.WriteLine("% Incomplete Command - Protocol Expected",
                        lexer, "EOL" , bVerbose );
921                 return NO_TOKENS;
922             }
923             string protocol = tokens[idx];
924             //
925             // Name or number of an IP Protocol.
926             //
927             if ( Regex.IsMatch(protocol, PROTOCOLS) )
928             {
929                 this.Protocol = protocol;
930             }
931             else
932             {
933                 traceLog.WriteLine("[ACL.ParseProtocol] % Invalid Input Detected:
                        IP Protocol or <0-255> expected");
934                 errorLog.WriteLine("% Invalid Input Detected: IP Protocol or
                        <0-255> expected", lexer, protocol, bVerbose );
935                 return INVALID;
936             }
937             // Valid Protocal.
938             traceLog.WriteLine("[ACL.ParseProtocol] valid Protocol: " + protocol);
939             return SUCCESS;
940         }
941 
942 
943         /// <summary>
944         /// Parse the Layer 3 Source Address.
945         /// (Host, Network, or Range of IP Addresses)
```

```
946         /// </summary>
947         /// <param name="tokens"></param>
948         /// <param name="idx"></param>
949         /// <returns>
950         ///     0   Success
951         ///     1   No more tokens
952         ///     2   Invalid Src Address
953         ///     3   Invalid Src Wildcard
954         /// </returns>
955         private int ParseSrc( string[] tokens, ref int idx, bool isStdACL )
956         {
957             bool isHost = false;
958 
959             idx++; // next token
960             if ( idx == tokens.Length )
961             {
962                 traceLog.WriteLine("\n[ParseSrc] % Incomplete Command - IP Address
                        Expected" );
963                 errorLog.WriteLine("% Incomplete Command - IP Address Expected",
                        lexer, "EOL", bVerbose);
964                 return NO_TOKENS;
965             }
966             if ( tokens[idx] == "any" )
967             {
968                 traceLog.WriteLine("[ACL.ParseSrc] valid source: " + tokens[idx]);
969                 // Match any address.
970                 this.SourceIP       = "0.0.0.0";
971                 this.SourceWildcard = "255.255.255.255";
972                 return SUCCESS;
973             }
974             else if ( tokens[idx] == "host" )
975             {
976                 idx++; // next token
977                 if ( tokens.Length == idx )
978                 {
979                     traceLog.WriteLine("\n[ACL.ParseSrc] % Incomplete Command -
                            Host IP Address expected");
980                     errorLog.WriteLine("% Incomplete Command - Host IP Address
                            expected", lexer, "EOL", bVerbose);
981                     return NO_TOKENS;
982                 }
983                 isHost = true;
984                 //
985                 // TBA: Optional [Hostname] param.
986                 //
987             }
988             if ( Regex.IsMatch(tokens[idx], IPv4ADDRESS) )
989             {
990                 // TBA: extra validation for IP Addrs, host or network? (
                        1.0.0.1-255.255.255.254 ?)
```

```
991                 //      To set isHost flag or not. Can we use bills isip()
                        function?
992                 //
993                 // Lionels IP class: DOESN'T WORK ( range seems to be:
                        2.0.0.1-255.255.255.254 !?)
994                 //IPOperations v = new IPOperations();
995                 //if ( v.isValidIPv4( tokens[idx] ) )
996                 //
997                 traceLog.WriteLine("[ACL.ParseSrc] valid source IP Address: " +
                        tokens[idx]);
998                 this.SourceIP   = tokens[idx];
999
1000                if ( isHost )
1001                {
1002                        return SUCCESS; // valid rule.
1003                }
1004                idx++; // next token
1005                if ( isStdACL && tokens.Length == idx )
1006                {
1007                    // No more tokens, but its a Std ACL - implies a host address.
1008                    this.SourceWildcard = "0.0.0.0";
1009                    idx--;
1010                    // Valid Rule.
1011                    return SUCCESS;
1012                }
1013                if ( tokens.Length == idx )
1014                {
1015                    traceLog.WriteLine("[ACL.ParseSrc] % Incomplete Command -
                            Wildcard expected");
1016                    errorLog.WriteLine("% Incomplete Command - Wildcard expected",
                             lexer, "EOL", bVerbose);
1017                    return NO_TOKENS;
1018                }
1019                //
1020                // source-wildcard-mask; network wildcard mask.
1021                //
1022                if ( Regex.IsMatch(tokens[idx], IPv4ADDRESS) )
1023                {
1024                    // TBA: wildcard matching? and/or validation for wildcard mask
1025                    // (must be a.b.c.d; a <= b <= c <= d e.g. 0.0.7.255)?
1026                    //
1027                    // Could use Bills iswildcard() method?
1028                    traceLog.WriteLine("[ACL.ParseSrc] valid source Wildcard: " +
                            tokens[idx]);
1029                    this.SourceWildcard = tokens[idx];
1030                }
1031                else
1032                {
1033                    // Invalid Wildcard.
1034                    if (!isStdACL)
1035                    {
```

```
1036                        // Mandatory Wildcard.
1037                        traceLog.WriteLine("[ACL.ParseSrc] % Invalid Input
                                Detected - Source Wildcard: " + tokens[idx]);
1038                        errorLog.WriteLine("% Invalid Input Detected - Source
                                Wildcard ", lexer, tokens[idx], bVerbose);
1039                        return 3;
1040                    }
1041                    else
1042                    {
1043                        // Optional Wildcard, could be another param after.
1044                        idx--;
1045                    }
1046                }
1047            }
1048            else
1049            {
1050                traceLog.WriteLine("[ACL.ParseSrc] % Invalid Input Detected -
                        Source IP: " + tokens[idx]);
1051                errorLog.WriteLine("% Invalid Input Detected - Source IP ", lexer,
                         tokens[idx], bVerbose );
1052                return INVALID;
1053            }
1054            return SUCCESS;
1055        }
1056
1057
1058        /// <summary>
1059        /// Parse the Optional [operator[port]] Params.
1060        /// (Only valid for TCP and UDP filtering rules)
1061        /// </summary>
1062        /// <param name="tokens"></param>
1063        /// <param name="idx"></param>
1064        /// <returns>
1065        ///     0   Success
1066        ///     1   No Tokens
1067        ///     2   Invalid port operator
1068        ///     3   No Tokens, No Port
1069        ///     4   Invalid port
1070        /// </returns>
1071        private int ParsePorts( string[] tokens, ref int idx )
1072        {
1073            //
1074            // Optional [Port Operator].
1075            //
1076            idx++; // next token
1077            if ( idx == tokens.Length )
1078            {
1079                return NO_TOKENS;
1080            }
1081            if ( (Regex.IsMatch(this.Protocol, TCP) || Regex.IsMatch(this.Protocol
                     , UDP))
```

```
1082                && Regex.IsMatch(tokens[idx], PORT_OPERATORS) )
1083            {
1084                traceLog.WriteLine("[ACL.ParsePorts] valid Operator: " + tokens[
                        idx]);
1085                // TBA: Store port number operator?
1086            }
1087            else
1088            {
1089                // Not a valid port operator.
1090                idx--;
1091                return 2;
1092            }
1093            //
1094            // Port Number <0-65535>, or TCP/UPD Named Protocol. (mandatory if
                 Operator given]
1095            //  TBA: if "range" operator, need to get 2 x portnumbers.
1096            //
1097            idx++; // next token
1098            if ( idx == tokens.Length )
1099            {
1100                return 3;
1101            }
1102            if ( Regex.IsMatch(tokens[idx], NUMBER) &&
1103                (Int32.Parse(tokens[idx]) >=0 && Int32.Parse(tokens[idx]) <=65535)
                      )
1104            {
1105                // Valid port number.
1106                traceLog.WriteLine("[ACL.ParsePorts] valid Port Number: " + tokens
                        [idx]);
1107            }
1108            else if ( Regex.IsMatch(this.Protocol, TCP) && Regex.IsMatch(tokens[
                 idx], TCP_PORT_NAMES) )
1109            {
1110                // Valid TCP port name.
1111                traceLog.WriteLine("[ACL.ParsePorts] valid TCP Port Name: " +
                        tokens[idx]);
1112            }
1113            else if ( Regex.IsMatch(this.Protocol, UDP) && Regex.IsMatch(tokens[
                 idx], UDP_PORT_NAMES) )
1114            {
1115                // Valid UDP port name.
1116                traceLog.WriteLine("[ACL.ParsePorts] valid UDP Port Name: " +
                        tokens[idx]);
1117            }
1118            else
1119            {
1120                // Not a valid port number.
1121                traceLog.WriteLine("[ACL.ParsePorts] % Invalid Input Detected -
                        Port: " + tokens[idx]);
1122                errorLog.WriteLine("% Invalid Input Detected - Port", lexer,
                        tokens[idx], bVerbose );
```

```
1123                return 4;
1124            }
1125            // TBA: Store port number. Maybe only if operator "eq"?
1126            // this.Port = tokens[idx];
1127            return SUCCESS;
1128        }
1129
1130
1131        /// <summary>
1132        /// Parse the Layer 3 Dest Address.
1133        /// (Host, Network, or Range of IP Addresses)
1134        /// </summary>
1135        /// <param name="tokens"></param>
1136        /// <param name="idx"></param>
1137        /// <returns>
1138        ///     0   Success
1139        ///     1   No more tokens
1140        ///     2   Invalid Address
1141        ///     3   Invalid Wildcard
1142        /// </returns>
1143        private int ParseDest( string[] tokens, ref int idx, bool isStdACL )
1144        {
1145            bool isHost = false;
1146
1147            idx++; // next token
1148            if ( idx == tokens.Length )
1149            {
1150                traceLog.WriteLine("\n[ParseDest] % Incomplete Command - IP
                        Address Expected" );
1151                errorLog.WriteLine("\r% Incomplete Command - IP Address Expected",
                         lexer, "EOL", bVerbose);
1152                return NO_TOKENS;
1153            }
1154            if ( tokens[idx] == "any" )
1155            {
1156                traceLog.WriteLine("[ACL.ParseDest] valid source: " + tokens[idx])
                        ;
1157                // Match any address.
1158                this.SourceIP       = "0.0.0.0";
1159                this.SourceWildcard = "255.255.255.255";
1160                return SUCCESS;
1161            }
1162            else if ( tokens[idx] == "host" )
1163            {
1164                idx++; // next token
1165                if ( tokens.Length == idx )
1166                {
1167                    traceLog.WriteLine("\n[ACL.ParseDest] % Incomplete Command -
                            Host IP Address expected");
1168                    errorLog.WriteLine("% Incomplete Command - Host IP Address
                            expected", lexer, "EOL", bVerbose);
```

```
1169                    return NO_TOKENS;
1170                }
1171                isHost = true;
1172                //
1173                // TBA: Optional [Hostname] param.
1174                //
1175            }
1176            if ( Regex.IsMatch(tokens[idx], IPv4ADDRESS) )
1177            {
1178                // TBA: extra validation for IP Addrs, host or network? (
                        1.0.0.0-255.255.255.254 ?)
1179                //      To set isHost flag or not. Can we use bills isip()
                        function?
1180                //
1181                // Lionels IP class: DOESNT WORK ( range seems to be:
                        2.0.0.1-255.255.255.254 !?)
1182                //IPOperations v = new IPOperations();
1183                //if ( ! v.isValidIPv4( tokens[idx] ) )
1184                //{
1185                //  // TEMP
1186                //  traceLog.WriteLine("[ACL.ParseDest] % INVALID SOURCE IP: " +
                        tokens[idx]);
1187                //  errorLog.WriteLine("% INVALID SOURCE IP ", lexer, tokens[idx],
                         bVerbose );
1188                //  return 1;
1189                //}
1190                traceLog.WriteLine("[ACL.ParseDest] valid IP Address: " + tokens[
                        idx]);
1191                this.SourceIP   = tokens[idx];
1192                if ( isHost )
1193                {
1194                    this.SourceWildcard = "0.0.0.0";
1195                    return SUCCESS; // valid rule
1196                }
1197                idx++; // next token
1198                if ( tokens.Length == idx )
1199                {
1200                    traceLog.WriteLine("[ACL.ParseDest] % Incomplete Command -
                            Wildcard expected");
1201                    errorLog.WriteLine("% Incomplete Command - Wildcard expected",
                             lexer, "EOL" , bVerbose );
1202                    return NO_TOKENS;
1203                }
1204                //
1205                // dest-wildcard-mask; network wildcard mask.
1206                //
1207                if ( Regex.IsMatch(tokens[idx], IPv4ADDRESS) )
1208                {
1209                    // TBA: wildcard matching? and/or validation for wildcard mask
1210                    // (must be a.b.c.d; a <= b <= c <= d e.g. 0.0.7.255)??
1211                    // Could use Bill iswildcard() method?
```

```
1212                    traceLog.WriteLine("[ACL.ParseDest] valid Wildcard: " + tokens
                            [idx]);
1213                    this.SourceWildcard = tokens[idx];
1214                }
1215                else
1216                {
1217                    traceLog.WriteLine("[ACL.ParseDest] % Invalid Input Detected -
                            Wildcard Mask: " + tokens[idx]);
1218                    errorLog.WriteLine("% Invalid Input Detected - Wildcard Mask",
                            lexer, tokens[idx], bVerbose );
1219                    return 3;
1220                }
1221            }
1222            else
1223            {
1224                traceLog.WriteLine("[ACL.ParseDest] % Invalid Input Detected - IP
                        Address: " + tokens[idx]);
1225                errorLog.WriteLine("% Invalid Input Detected - IP Address", lexer,
                         tokens[idx], bVerbose );
1226                return INVALID;
1227            }
1228            return SUCCESS;
1229        }
1230
1231
1232        /// <summary>
1233        /// Parse the Optional Established Param; only valid for TCP protocol rule
                .
1234        /// </summary>
1235        /// <param name="tokens"></param>
1236        /// <param name="idx"></param>
1237        /// <returns>
1238        ///     0   Success
1239        ///     1   No tokens
1240        ///     2   Invalid log param
1241        /// </returns>
1242        private int ParseEstablished( string[] tokens, ref int idx )
1243        {
1244            idx++; // next token
1245            if ( idx == tokens.Length )
1246            {
1247                return NO_TOKENS; // no more tokens
1248            }
1249            if ( Regex.IsMatch(this.Protocol, TCP) && tokens[idx] == "established"
                     )
1250            {
1251                traceLog.WriteLine("[ACL.ParseEstablished] valid established param
                        " );
1252                this.Log = true;
1253            }
1254            else
```

```
1255            {
1256                idx--;
1257                return INVALID;
1258            }
1259            return SUCCESS;
1260        }
1261
1262
1263        /// <summary>
1264        /// Parse the Optional [icmp-type[icmp-code]|icmp-name] Params; only valid
                for ICMP protocol.
1265        /// </summary>
1266        /// <param name="tokens"></param>
1267        /// <param name="idx"></param>
1268        /// <returns>
1269        ///     0   Success
1270        ///     1   No Tokens
1271        ///     2   Invalid icmp param
1272        ///     3   No Tokens, No icmp-code
1273        ///     4   Invalid icmp-code value, after consuming valid icmp-type param
1274        /// </returns>
1275        private int ParseICMPType( string[] tokens, ref int idx )
1276        {
1277            //
1278            // Optional [icmp-type|icmp-name].
1279            //
1280            idx++; // next token
1281            if ( idx == tokens.Length )
1282            {
1283                return NO_TOKENS;
1284            }
1285            if ( Regex.IsMatch(this.Protocol, ICMP) && Regex.IsMatch(tokens[idx],
                    ICMP_NAME) )
1286            {
1287                // Valid icmp-name.
1288                traceLog.WriteLine("[ACL.ParseICMPType] valid icmp-name: " +
                        tokens[idx]);
1289                // TBA: Store icmp-name value?
1290                return SUCCESS;
1291            }
1292            else if ( Regex.IsMatch(this.Protocol, ICMP) && Regex.IsMatch(tokens[
                    idx], ICMP_TYPE) )
1293                {
1294                    // Valid icmp-type.
1295                    traceLog.WriteLine("[ACL.ParseICMPType] valid icmp-type: " +
                            tokens[idx]);
1296                }
1297            else
1298            {
1299                idx--; // not icmp param
1300                return 2;
```

```
1301            }
1302            //
1303            // icmp-code <0-255>; mandatory if icmp-type consumed.
1304            // TBA: validation for icmp-code (e.g. icmp-type=8 "echo" icmp-code=0.
1305            //
1306            idx++; // next token
1307            if ( idx == tokens.Length )
1308            {
1309                return 3; // no tokens after valid icmp-type param consumed
1310            }
1311            if ( Regex.IsMatch(tokens[idx], ICMP_CODE) )
1312            {
1313                // Valid icmp-code.
1314                traceLog.WriteLine("[ACL.ParseICMPType] valid icmp-code: " +
                        tokens[idx]);
1315            }
1316            else
1317            {
1318                // Not a valid icmp-code.
1319                traceLog.WriteLine("[ACL.ParseICMPType] % Invalid Input Detected -
                         icmp-code: " + tokens[idx]);
1320                errorLog.WriteLine("% Invalid Input Detected - icmp-code ", lexer,
                         tokens[idx], bVerbose );
1321                return 4;
1322            }
1323            // TBA: Store icmp-type & icmp-code values?
1324            return SUCCESS;
1325        }
1326
1327
1328        /// <summary>
1329        /// Parse Optional [igmp-type]; for IGMP protocol only.
1330        /// </summary>
1331        /// <param name="tokens"></param>
1332        /// <param name="idx"></param>
1333        /// <returns>
1334        ///     0   Success
1335        ///     1   No Tokens
1336        ///     2   Invalid igmp-type
1337        /// </returns>
1338        private int ParseIGMPType( string[] tokens, ref int idx )
1339        {
1340            idx++; // next token
1341            if ( idx == tokens.Length )
1342            {
1343                return NO_TOKENS;
1344            }
1345            if ( Regex.IsMatch(this.Protocol, IGMP) && Regex.IsMatch(tokens[idx],
                     IGMP_TYPE) )
1346            {
1347                // Valid igmp-type.
```

```
1348                traceLog.WriteLine("[ACL.ParseIGMPType] valid igmp-type: " +
                        tokens[idx]);
1349            }
1350            else
1351            {
1352                idx--; // not igmp param
1353                return 2;
1354            }
1355            // TBA: Store igmp-type?
1356            return SUCCESS;
1357        }
1358
1359
1360        /// <summary>
1361        /// Parse the Optional [precedence precedence-value] Params.
1362        /// </summary>
1363        /// <param name="tokens"></param>
1364        /// <param name="idx"></param>
1365        /// <returns>
1366        ///     0   Success
1367        ///     1   No Tokens
1368        ///     2   Invalid precedence param
1369        ///     3   No Tokens, No precedence value
1370        ///     4   Invalid precedence value, after consuming valid precedence
                param
1371        /// </returns>
1372        private int ParsePrecedence( string[] tokens, ref int idx )
1373        {
1374            //
1375            // Optional [precedence] keyword.
1376            //
1377            idx++; // next token
1378            if ( idx == tokens.Length )
1379            {
1380                return NO_TOKENS;
1381            }
1382            if ( tokens[idx] != "precedence" )
1383            {
1384                idx--; // not precedence param
1385                return 2;
1386            }
1387            //
1388            // precedence-value <0-7>, or Named precedence value
1389            //
1390            idx++; // next token
1391            if ( idx == tokens.Length )
1392            {
1393                return 3; // no tokens after valid precedence param consumed
1394            }
1395            if ( Regex.IsMatch(tokens[idx], PRECEDENCE) )
1396            {
```

```
1397                // Valid precedence number/name.
1398                traceLog.WriteLine("[ACL.ParsePrecedence] valid Precedence Value:
                        " + tokens[idx]);
1399            }
1400            else
1401            {
1402                // Not a valid precedence value.
1403                traceLog.WriteLine("[ACL.ParsePrecedence] % Invalid Input Detected
                        - Precedence: " + tokens[idx]);
1404                errorLog.WriteLine("% Invalid Input Detected - Precedence ", lexer
                        , tokens[idx], bVerbose );
1405                return 4;
1406            }
1407            // TBA: Store precedence value?
1408            return SUCCESS;
1409        }
1410
1411
1412        /// <summary>
1413        /// Parse the Optional [dscp dscp-value] Params.
1414        /// </summary>
1415        /// <param name="tokens"></param>
1416        /// <param name="idx"></param>
1417        /// <returns>
1418        ///     0   Success
1419        ///     1   No Tokens
1420        ///     2   Invalid precedence param
1421        ///     3   No Tokens, No precedence value
1422        ///     4   Invalid precedence value, after consuming valid precedence
                    param
1423        /// </returns>
1424        private int ParseDSCP( string[] tokens, ref int idx )
1425        {
1426            //
1427            // Optional [dscp] keyword.
1428            //
1429            idx++; // next token
1430            if ( idx == tokens.Length )
1431            {
1432                return NO_TOKENS;
1433            }
1434            if ( tokens[idx] != "dscp" )
1435            {
1436                idx--; // not DSCP keyword
1437                return 2;
1438            }
1439            //
1440            // DSCP value <6 bits: 0-63>, or Named DSCP value.
1441            //
1442            idx++; // next token
1443            if ( idx == tokens.Length )
```

```
1444            {
1445                return 3; // no tokens after valid DSCP keyword consumed
1446            }
1447            if ( Regex.IsMatch(tokens[idx], DSCP) )
1448            {
1449                // Valid DSCP number/name.
1450                traceLog.WriteLine("[ACL.ParseDSCP] valid DSCP Value: " + tokens[
                        idx]);
1451            }
1452            else
1453            {
1454                // Not a valid precedence value.
1455                traceLog.WriteLine("[ACL.ParseDSCP] % Invalid Input Detected -
                        DSCP: " + tokens[idx]);
1456                errorLog.WriteLine("% Invalid Input Detected - DSCP ", lexer,
                        tokens[idx], bVerbose );
1457                return 4;
1458            }
1459            // TBA: Store DSCP value?
1460            return SUCCESS;
1461        }
1462
1463
1464        /// <summary>
1465        /// Parse the Optional [tos tos-value] Params.
1466        /// </summary>
1467        /// <param name="tokens"></param>
1468        /// <param name="idx"></param>
1469        /// <returns>
1470        ///     0   Success
1471        ///     1   No Tokens
1472        ///     2   Invalid tos param
1473        ///     3   No Tokens, No tos value
1474        ///     4   Invalid tos value, after consuming valid tos param
1475        /// </returns>
1476        private int ParseTos( string[] tokens, ref int idx )
1477        {
1478            //
1479            // Optional [tos] keyword.
1480            //
1481            idx++; // next token
1482            if ( idx == tokens.Length )
1483            {
1484                return NO_TOKENS;
1485            }
1486            if ( tokens[idx] != "tos" )
1487            {
1488                idx--; // not TOS param
1489                return 2;
1490            }
1491            //
```

```
1492            // TOS Value <0-15>, or Named TOS value
1493            //
1494            idx++; // next token
1495            if ( idx == tokens.Length )
1496            {
1497                return 3; // no tokens after valid TOS keyword consumed
1498            }
1499            if ( Regex.IsMatch(tokens[idx], TOS) )
1500            {
1501                // Valid TOS number/name.
1502                traceLog.WriteLine("[ACL.ParseTos] valid tos Value: " + tokens[idx
                        ]);
1503            }
1504            else
1505            {
1506                // Not a valid tos value.
1507                traceLog.WriteLine("[ACL.ParseTos] % Invalid Input Detected - tos:
                         " + tokens[idx]);
1508                errorLog.WriteLine("% Invalid Input Detected - tos ", lexer,
                        tokens[idx], bVerbose );
1509                return 4;
1510            }
1511            // TBA: Store tos value?
1512            return SUCCESS;
1513        }
1514
1515
1516        /// <summary>
1517        /// Parse the Optional Log Param.
1518        /// </summary>
1519        /// <param name="tokens"></param>
1520        /// <param name="idx"></param>
1521        /// <returns>
1522        ///     0   Success
1523        ///     1   No tokens
1524        ///     2   Invalid log param
1525        /// </returns>
1526        private int ParseLog( string[] tokens, ref int idx )
1527        {
1528            idx++; // next token
1529            if ( idx == tokens.Length )
1530            {
1531                return NO_TOKENS; // no more tokens
1532            }
1533            if ( Regex.IsMatch(tokens[idx], LOG) )
1534            {
1535                traceLog.WriteLine("[ACL.ParseLog] valid log param" );
1536                this.Log = true;
1537            }
1538            else
1539            {
```

```
1540                idx--;
1541                return INVALID;
1542            }
1543            return SUCCESS;
1544        }
1545
1546
1547        /// <summary>
1548        /// Parse the Optional [time-range time-range-name] Params.
1549        /// </summary>
1550        /// <param name="tokens"></param>
1551        /// <param name="idx"></param>
1552        /// <returns>
1553        ///     0   Success
1554        ///     1   No Tokens
1555        ///     2   Invalid time-range param
1556        ///     3   No Tokens, No time-range-name
1557        ///     4   Invalid time-range-name value, after consuming valid time-
                range param
1558        /// </returns>
1559        private int ParseTimeRange( string[] tokens, ref int idx )
1560        {
1561            //
1562            // Optional [time-range].
1563            //
1564            idx++; // next token
1565            if ( idx == tokens.Length )
1566            {
1567                return NO_TOKENS;
1568            }
1569            if ( tokens[idx] != "time-range" )
1570            {
1571                idx--; // not time-range param
1572                return 2;
1573            }
1574            //
1575            // time-range-name
1576            //
1577            idx++; // next token
1578            if ( idx == tokens.Length )
1579            {
1580                return 3; // no tokens after valid time-range param consumed
1581            }
1582            // TBA: should really be checking a symbol table; time-range-name
                    should be defined before use
1583            if ( Regex.IsMatch(tokens[idx], ALPHASTART) )
1584            {
1585                // Valid time-range-name.
1586                traceLog.WriteLine("[ACL.ParseTimeRange] valid time-range: " +
                        tokens[idx]);
1587            }
```

```
1588            else
1589            {
1590                // Not a valid time-range-name.
1591                traceLog.WriteLine("[ACL.ParseTimeRange] % Invalid Input Detected
                        - time-range: " + tokens[idx]);
1592                errorLog.WriteLine("% Invalid Input Detected - time-range ", lexer
                        , tokens[idx], bVerbose );
1593                return 4;
1594            }
1595            // TBA: Store time-range?
1596            return SUCCESS;
1597        }
1598
1599
1600        /// <summary>
1601        /// Parse the Optional Fragments Param.
1602        /// </summary>
1603        /// <param name="tokens"></param>
1604        /// <param name="idx"></param>
1605        /// <returns>
1606        ///     0   Success
1607        ///     1   No tokens
1608        ///     2   Invalid param
1609        /// </returns>
1610        private int ParseFragments( string[] tokens, ref int idx )
1611        {
1612            idx++; // next token
1613            if ( idx == tokens.Length )
1614            {
1615                return NO_TOKENS; // no more tokens
1616            }
1617            if ( tokens[idx] == "fragments" )
1618            {
1619                traceLog.WriteLine("[ACL.ParseFragments] valid fragments param" );
1620                this.Log = true;
1621            }
1622            else
1623            {
1624                // Last param, so invalid even though its optional.
1625                traceLog.WriteLine("[ACL.ParseFragments] % Invalid Input Detected:
                        " + tokens[idx]);
1626                errorLog.WriteLine("% Invalid Input Detected ", lexer, tokens[idx
                        ], bVerbose );
1627                return INVALID;
1628            }
1629            return SUCCESS;
1630        }
1631
1632
1633    }
1634 }
```