Generic Firewall Rule Compiler
And Modeller

CO42019 Honours Project

UNDERGRADUATE PROJECT DISSERTATION

Christopher Geeringh
Supervisor: Professor William J. Buchanan

Submitted in partial fulfilment of the requirements of
Napier University for the degree of
Bachelor of Science with Honours in Network Computing
School of Computing
May 2007
Authorship Declaration

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

1. Where I have consulted the published work of others this is always clearly attributed.
2. 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.
3. I have acknowledged all main sources of help.
4. 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.
5. I have read and understand the penalties associated with plagiarism.

Signed

Name: Christopher Geeringh
Matriculation No.: 05001842
Abstract

Firewalls are one of the most widely deployed network devices to provide security to a network. The firewall provides the first line of defence to a network, and its configuration is critical to security. However, the configuration of firewalls is an error prone task. This can be attributed to the need to understand the specific hardware being deployed, and due to the misinterpretation of the security policy of the organisation.

The concept of firewall modelling arises as an attempt to develop an abstract model of firewall configurations. By modelling a firewall, it is possible to identify the relationships between rules, thus identifying errors. Identification of errors within firewall rule-sets is an essential requirement to ensure that a firewall policy does reflect the requirements of the organisations security policy.

The presented work gives an investigation into existing work into firewall modelling. Furthermore, the concept of a firewall anomaly and the various types of anomalies is presented. A new and novel approach to firewall optimisation is presented - rule crunching. Rule crunching is explained, and its applicability in firewall rule optimisation is discussed. The presented work incorporates an implementation of the findings, in the form of a prototype application – The Generic Firewall Rule Compiler and Modeller. This prototype system is evaluated and the findings of the evaluation process are reviewed and discussed.

A final overview of the work is presented, highlighting the shortcomings, and further work to be undertaken.
# Contents

Authorship Declaration .................................................................................................. 2  
Abstract .......................................................................................................................... 3  
Contents ......................................................................................................................... 4  
List of Figures and Tables .............................................................................................. 6  
Acknowledgements ........................................................................................................ 7  
1. Introduction ................................................................................................................ 8  
   1.1 Project Overview ................................................................................................. 8  
   1.2 Background ......................................................................................................... 9  
   1.3 Aims and Objectives .......................................................................................... 10  
   1.4 Thesis Structure ................................................................................................. 11  
2. Theory ...................................................................................................................... 12  
   2.1 Introduction ........................................................................................................ 12  
   2.2 IP Addressing ..................................................................................................... 12  
   2.2.1 Classification ............................................................................................... 12  
   2.2.2 Sub-netting .................................................................................................. 13  
   2.2.3 Wildcard Mask ............................................................................................ 14  
   2.3 Bitwise Operations ............................................................................................. 14  
   2.3.1 AND Operator ............................................................................................. 14  
   2.3.2 OR Operator ................................................................................................ 14  
   2.3.3 NOT Operator ............................................................................................. 15  
   2.3.4 XOR Operator ............................................................................................. 15  
   2.4 TCP/UDP Ports .................................................................................................. 15  
   2.5 Firewall Background .......................................................................................... 16  
   2.6 Conclusion ......................................................................................................... 17  
3. Literature Review ..................................................................................................... 18  
   3.1 Introduction ........................................................................................................ 18  
   3.2 Security – An abstract approach ........................................................................ 18  
   3.3 Firewall Modelling ............................................................................................. 20  
   3.4 Policy Anomalies ............................................................................................... 21  
   3.4.1 Shadowing Anomaly ................................................................................... 21  
   3.4.2 Correlation Anomaly .................................................................................. 21  
   3.4.3 Generalisation Anomaly ............................................................................. 22  
   3.4.4 Redundancy Anomaly ................................................................................. 23  
   3.6 Computer Law ................................................................................................... 23  
   3.7 Performance and Accountability ........................................................................ 25  
   3.8 Conclusion ......................................................................................................... 26  
4. Design ...................................................................................................................... 27  
   4.1 Introduction ........................................................................................................ 27  
   4.2 Generic Firewall Compiler and Modeller Design .............................................. 27  
   4.2.1 Generic Syntax ............................................................................................ 27  
   4.2.2 Anomaly Discovery .................................................................................... 28  
   4.2.3 Rule Crunching ........................................................................................... 28  
   4.3 Experiment Design ............................................................................................. 30  
   4.3.1 Experiment 1 – Rule Verification Timing .................................................. 30  
   4.3.2 Experiment 2 – Rule Anomaly Discovery Timing ..................................... 31  
   4.3.3 Experiment 3 – Rule Crunching Timing .................................................... 31
List of Figures and Tables

Figure 2-1. IP Packet Header 12
Table 2-1. IP Classification of Class A, B, and C 13
Table 2-2. Classes A, B, and C Subnet Information 14
Table 2-1. AND Operator 14
Table 2-2. OR Operator 15
Table 2-5. NOT Operator 15
Table 2-3. XOR Operator 15
Figure 2-2. TCP Layer 4 Packet Header Information 16
Table 2-7. Figure 2. Basic firewall concept 17
Figure 3-1. Novel Framework for Automated Security Abstraction, Modelling, Implementation and Verification (Saliou et al, 2005) 19
Figure 3-2. Generic Syntax (Al-Shaer et al. 2004) 20
Figure 3-3. The use of the generic syntax to define an example firewall policy 20
Figure 3-4. Shadowing Anomaly 21
Figure 3-5. Correlation Anomaly 22
Figure 3-6. Generalisation Anomaly 22
Figure 3-7. Redundancy Anomaly 1 23
Figure 3-8. Redundancy Anomaly 2 23
Figure 3-9. The Balance of needs (Graves 2006) 25
Figure 3-10. Example of Redundant Rules 26
Figure 4-1. Generic Syntax (Al-Shaer et al, 2004) 27
Table 4-2. Proposed Syntax Parameters 27
Figure 4-3. Example Firewall Policy 28
Figure 4-3. Example Firewall Policy 28
Figure 4-4. Source Addresses Converted to Binary 29
Figure 4-5. Result of AND operation on binary addresses 29
Figure 4-6. Creating the new crunch rule 29
Figure 4-7. Crunch Rule 29
Figure 4-8. Crunching worst case scenario 30
Figure 4-9. Worst case scenario rule crunch result 30
Figure 4-10. Evaluation Environment (Saliou et al. 2007) 33
Table 5-1. Syntax Definition 34
Figure 5-2. User Interface 35
Figure 5-3 Verification Flow 35
Figure 5-4. Rule Anomalies Tab 37
Figure 5-5. Rule Crunching Tab 38
Figure 6-1. Rule Verification Timing Results 39
Figure 6-2. Anomaly Discovery Timing Results 40
Figure 6-3. Rule Crunching Timing Results 41
Figure 6-4. Platform Specific Rule Generation Timing Results 41
Figure 6-5. Firewall Throughput performance 42
Acknowledgements

I would like to extend thanks to Professor William Buchanan for his supervision and guidance during the formation of this dissertation. I would also like to thank Lionel Saliou for his support and help during the evaluation stage. Finally, I would like to thank Andrew Cumming for being part of the marking process.
1. Introduction

1.1 Project Overview

The firewall is the first line of defence to a network, and controls the flow of traffic entering and exiting network boundaries. However, this commonly deployed device frequently exhibits configuration errors, which are unknown to the administrator (Wool, 2004). These errors can cause unexpected traffic behaviour within a network.

The aim of this project is to investigate work in the field of firewall modelling, and develop a system which can effectively model firewall policies. Through investigation of existing work, policy anomaly definitions allow for the creation of anomaly discovery algorithms, to find configuration errors in an automated manner. Furthermore A novel approach to firewall rule management is taken, with the concept of rule crunching. The system will provide a complete and novel approach to firewall modelling and management for network and security administrators.
1.2 Background

Computers are becoming an increasingly prominent aspect to our everyday lives, such as in our jobs and at home for entertainment. As computer prominence increases, so do the networks they connect to grow in size. The world’s biggest computer network, the Internet, has grown exponentially in the past decade with more than 2 billion users worldwide today.

With the phenomenal growth of the Internet and computer networks, so the amount of threats and vulnerabilities has increased. Malicious software, know as Malware poses a constant threat to organisations. Some example of malware includes Viruses, Trojans, Worms and Spyware. In addition, Hacking and Denial of Service (DoS) attacks are further threats that need to be taken into consideration when deploying and administrating a network. In organisations today administrators are in a constant battle against these threats and finding improvements to security systems currently implemented. One of the most deployed systems for mitigating these threats and regulating network usage is the firewall.

The firewall regulates the flow of traffic entering and exiting network boundaries. A firewall policy is used to define the traffic regulation. Various firewall vendors are available, each with their own configuration languages and policy implementations. Hence, administrators are required to have a working knowledge of the hardware, in order to successfully implement policies (Caldwell et al. 2004). Furthermore, the task of writing firewall policies is error prone (Wool, 2004). Errors in firewall policies lead to anomalies. Thus, tools are available to the administrator to facilitate the job of firewall rule administration, such as the Firewall Policy Advisor (Al-Shaer et al. 2004), and FIREMAN (Yuan et al. 2007)

The presented work investigates firewall modelling, and the tools available to administrators. Through the analysis of the findings, a prototype firewall modeller is presented, which uses language abstraction to reduce the burden needed to configure specific vendor hardware.
1.3 Aims and Objectives

The aim of the project is to investigate current work into the field of firewall modelling, and the development of a generic syntax compiler. Furthermore, the work will be done in the field of firewall anomaly discovery through an evaluation of existing work, and finally develop a novel approach to further firewall policy management.

Thus, the following aims can be identified:

(a) Research previous work in the field.
(b) Develop a prototype compiler.
(c) Evaluate the prototype, and refine where necessary.
(d) Investigate rule crunching.
(e) Build up the prototype to incorporate some functionality of the work of Al-Shaer.

In order to accomplish these aims the following objectives were identified:

(a) Through investigation of existing research, define a generic syntax format.
(b) Choose a programming language suitable for the development of the system.
(c) Define methods for testing and evaluation.
(d) Propose future work and research.
1.4 Thesis Structure

Chapter 1  Introduction. This chapter provides the overview of the work undertaken.

Chapter 2  Theory. During this section underlying theory is discussed giving a foundation to the remainder of the project.

Chapter 3  Literature Review. Here a review of previous and current work into the field of firewall modelling is discussed. A discussion of configuration anomalies is presented.

Chapter 4  Design. This chapter provides a design overview of the prototype.

Chapter 5  Implementation. This sections provides the details behind the implementation of the system and experiments to evaluate it.

Chapter 6  Experiment Analysis. The results from experiments are discussed here, together with an evaluation of the findings.

Chapter 7  Discussion, Conclusions and Future Work. This chapter summarises the entire project, presents findings and suggest further work.

Chapter 8  References.

Chapter 9  Appendices.
2. Theory

2.1 Introduction

Firewalls are devices which have become common place in networks and their operation is often taken for granted. This chapter gives background to firewalls, and the concept of fire-walling. An investigation into existing firewalls within the scope of this project is undertaken to give further background and understanding to this work.

2.2 IP Addressing

In order for a node to be connected to any network, including the internet, it needs to have an Internet Protocol (IP) address. An IP address is a unique numerical identifier signifying the logical location of a node on a network. IP addresses are found within layer three, the network layer, of the Open System Interconnect (OSI) model. Figure 2-1 illustrates IP header information associated with this layer.

![Figure 2-1. IP Packet Header](image)

IP addresses are made up of 32 binary bits, which are broken down into eight octets. Typically though, IP addresses are written in dotted decimal notation where the decimal equivalent is used. An example of an IP address is 72.14.207.99 which provides the logical location of the google.com web server. A certain number of binary bits in the IP address make up the network portion and the remaining bits make up the host portion. The Internet Corporation for Assigned Names and Numbers (ICANN) is the body responsible for assigning IP addresses to organisations and individuals.

2.2.1 Classification

When the original IP specification was developed, five different classes for IP addresses were defined. The classification of IP addresses is by the number of bits which are used for the network portion of the address. Class A, Class B, Class C, Class D, and Class E. Each class differs in the range of addresses they cover. Typically, Class A addresses are assigned to large organisations with millions of hosts. Class B addresses, are usually assigned to organisations with thousands of hosts, such as universities. Finally, the assignment of Class C addresses is to organisations only requiring a few hundred host addresses. Class D addresses are for multicasting purposes, while Class E addresses are reserved addresses.
addresses are for experimental purposes. Table 2-1, shows the address ranges for
classes A, B, and C and the corresponding number of networks and hosts. Classes A
and B contain the largest number of hosts and in order to obtain an address in either of
these two classes, sufficient justification would need to be given to ICANN.

<table>
<thead>
<tr>
<th>Class</th>
<th>Initial Byte</th>
<th>Networks</th>
<th>Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0 – 127</td>
<td>126</td>
<td>16777214</td>
</tr>
<tr>
<td>Class B</td>
<td>128 - 191</td>
<td>16384</td>
<td>65536</td>
</tr>
<tr>
<td>Class C</td>
<td>192 - 233</td>
<td>2097152</td>
<td>254</td>
</tr>
</tbody>
</table>

Table 2-1. IP Classification of Class A, B, and C

2.2.2 Sub-netting

Due to the rapid expansion of the internet and the inefficient allocation of address
blocks due to class-full addressing. A more efficient approach was required, Sub-
etting. Subnets provide a range of logical addresses within an address space. The
number of hosts which a subnet may contain, is defined by the subnet mask. The
subnet mask consists of 32 binary bits. The number of bits, which are set to “1”,
defines subnet masks. As with IP addresses, subnet mask representation can be in
dotted decimal format. Furthermore, subnet masks can be represented in Classless
Inter-Domain Routing (CIDR) notation, indicating the number of bits being used for
the mask written after a slash (“/”). Class A, B, and C addresses have default subnet
masks (Table 2-2). For example, if an organisation consists of 12 hosts requiring
publicly routable addresses, it would only require a subnet mask of /28. This would
leave 4 bits remaining for host addresses, which would provide 16 contiguous
addresses, for use by the organisation. Previously, if an organisation was granted an
address, a class C address would have been assigned. Thus, giving the organisation
255 hosts addresses, this is considered a waste of address space. Appendix 1 provides
a full table of subnet masks.

* Throughout the presented work, the position of a single bit is counted from left to
right. For example when referring to the bit in position 10, would be the second bit in
octet 2.

Privately Routable Addresses

The use of privately routable addresses, is another way in which the rapid reduction in
available IP addresses was overcome. Privately routable addresses are used within
organisations, and are not accessible over the internet. The use of Network Address
Translation allows many organisations to utilise the same address space on their
internal networks, while still having access to external resources. There are three
privately routable address ranges, they are:

- 10.0.0.0 to 10.255.255.255 (10.0.0.0/8) – Class A
- 172.16.0.0 to 172.31.255.255 (172.16.0.0/12) – Class B
- 192.168.0.0 to 192.168.255.255 (192.168.0.0/16) – Class C
### 2.2.3 Wildcard Mask

The wildcard mask can be considered to be the inverse of the subnet mask. The subnet mask defined the network portion with “1”s and the host portion with “0”s. The wildcard mask represents host portion with “1”s. Wildcard masks are used in the creation of Cisco Access control lists (ACLs).

### 2.3 Bitwise Operations

Bitwise operations are carried out at a hardware level by logic gates. Some examples of logic gates include AND, OR, NOT and XOR.

#### 2.3.1 AND Operator

During the bitwise binary AND operation on two binary values, the logical AND operation is applied. The result is true, if both values are true. In computer programming the bitwise AND operator is represented with the ampersand (&), while the logical AND operator is represented with two ampersands (&&).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2-3. AND Operator

#### 2.3.2 OR Operator

During the bitwise binary OR operation on two binary values, the logical OR operator is applied. The result is true, if either of the values are true. If neither are true, the result is false. In computer programming the bitwise OR operator is represented with the pipe character (|), while the logical OR operator is represented with two pipe characters (||).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2-3. OR Operator
2.3.3 NOT Operator

During the NOT binary *unary* operation, the result is the *negation* of the given value. If the input is true, the output is false and vice-versa. In computer programming the bitwise NOT operator is represented with the tilde character (~), while the logical NOT operator is represented with the exclamation mark character (!).

<table>
<thead>
<tr>
<th>A</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-5. NOT Operator

2.3.4 XOR Operator

During the XOR binary operation, the XOR logical operator is used. The result is true if one of the two values is *true*. If both values are true, or both are false, the result is *false*. In computer programming the bitwise XOR operator is represented with the caret character (^).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-6. XOR Operator

2.4 TCP/UDP Ports

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), are two layers found on the transport layer of the TCP/IP stack, and the OSI layer. Figure ?? illustrates the header information contained within a TCP packet. TCP is considered a connection oriented protocol, as it sets up a connection between two hosts before any data is sent. Furthermore TCP uses a method of acknowledging receipt of packets to
ensure lost packets are resent. UDP however is considered a connectionless protocol as there is no error correction. UDP packets are sent with a hope that they arrive at their destination.

Both of these protocols make use of “ports”. Ports are used to identify specific applications accessing the network. Common applications operate on specific ports, such as web servers operate on port 80, while the Domain Name Server (DNS) operates on port 53. If a host wishes to access a web server, the TCP packet’s destination IP address, will be that of the web server, while the destination port will be 80. Thus it is possible to establish what applications or services a user is operating by investigating the ports used for communication. Table 2-7 shows some common ports and their associated services.

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web server</td>
<td>80</td>
</tr>
<tr>
<td>Telnet</td>
<td>23</td>
</tr>
<tr>
<td>File Transfer</td>
<td>21</td>
</tr>
<tr>
<td>DNS</td>
<td>53</td>
</tr>
</tbody>
</table>

Table 2-7 Common Services and Ports

2.5 Firewall Background

A firewall allows for the filtering of packets as they traverse the boundary of a network. A firewall may also be placed on a host, this is usually done by means of a software firewall package. A host based firewall, controls the access specific applications have to the network, and the connections applications can receive.

Firewalls make their filtering decisions based on packet header information, as shown in Section 2.1 and Section 2.4. Firewalls match the information in the header fields against rules defined by administrator to make filtering decisions. Filtering decisions can either be to allow the packet to traverse the network boundary, or to block its traversal. Figure ?? illustrates a basic firewall system in a host-to-host situation. However far more complex firewall configurations and implementation need consideration, particularly within organisations. Firewalls in organisations control far more than single end-to-end connections between hosts, as organisational networks can be logically segmented. Logical segmentation of the network places hosts with certain IP addresses into different logical segments on the network. This is usually
done to ease administration and control network access organisation departments may have.

![Figure 1. Basic firewall concept](image)

2.6 Conclusion

This chapter has provided a level of foundation needed to understand how the Internet Protocol works. The concept of an IP address, and the use of ports provides foundations to how a firewall operates. An outline of Cisco and Linux firewalls provides some background to some commonly deployed firewall systems. This chapter gives the foundations required to understand the research and concepts discussed in the presented work.
3. Literature Review

3.1 Introduction

This chapter covers existing work into the field of firewall modelling, the concept of firewall anomalies and optimisation. A look at the Novel Framework for Automated Security Abstraction, Modelling, Implementation and Verification by Saliou et al 2005 is approached, and the concept of an abstract security framework investigated. Chapter 2 provides a foundation for many of the concepts presented.

3.2 Security – An abstract approach

Security is a major issue for organisations, both the legal issues associated with computer and network security as well as its implementation. Threats such as malware and DoS constantly test organisations' security. Thus, it is necessary to take the concept of security as an ideal, rather than an implementation applied to various components of a system. It is necessary to take the entire system as an entity and the application of security to the entire network. A simplistic and specific approach of applying security measures to each individual component on a “what seems best” principle can result in an inconsistent security deployment. Security policy implementations quite often do not reflect the security policy requirements, due to errors in policy interpretation (Danchev, 2004). The framework presented by Saliou et al, 2005 needs consideration. By taking the approach of a framework, security can be conceptualised at a higher level. Through this higher-level abstraction process, the specific vendor hardware found at the physical implementation level does not need consideration. Figure ?? illustrates the abstraction of security as a framework (Saliou et al, 2005). Within this framework, the security policy definition is at a higher level, in organisation this can be by both administrators and managers. The work by Saliou in the framework allows for the development of a security policy from the organisational, legal and social requirements and existing policy templates. In this context the firewall is not the complete solution to security, it is a single component of the system. Within the Novel Framework presented by Saliou et al. 2005, the policy is processed within a security compiler where it is further refined. The security policy is modelled and then can be considered for deployment with the use of a “Security Implementer” (Figure 3-1).
Figure 3-1. Novel Framework for Automated Security Abstraction, Modelling, Implementation and Verification (Saliou et al, 2005)
3.3 Firewall Modelling

The configuration language used by various firewall vendors differs significantly, as well as the rule organisation and interaction between rules. For example, both Linux and Cisco firewalls operate by taking a decision based on the first matched rule, however FreeBSD bases its action on the last matched rule (www.freebsd.org). The concept of a firewall is common to all the various vendors, each rule contains a set of parameters, and the header information contained in each packet is tested to see if a match is found. Keeping this in mind it is possible to model a firewall policy. The relationships between rules can be established, and it is these relationships, which allows for the analysis of rules on an individual bases, as well as their impact on the policy as a whole. In the work on the Firewall Policy Advisor, a generic syntax definition is presented (Al-Shaer et al, 2004). Previous work done into firewall modelling used Cisco syntax. The use of a specific syntax becomes a limitation when other vendor firewalls are deployed (Guttman et al. 1997). Furthermore, Guttman used Cisco syntax based on the Cisco Internetwork Operating System (IOS) release 10; this further tied the work to Cisco devices running a specific version of the operating system. The work done by Al-Shaer et al. used a generic syntax, allowing for a level of abstraction away from the specification of vendor hardware, and allows for a higher-level approach to modelling. However, the proposed syntax by Al-Shaer et al. in the Firewall Policy Advisor, did not allow for the use of Classless Inter Domain Routing (CIDR) notation, thus limiting the applications ability for use with classless networks.

<order><protocol><s ip><s port><d ip><d port><action>

Figure 3-2. Generic Syntax (Al-Shaer et al. 2004)

This syntax allows rule filtering based on five parameters:

- **Protocol:** This is the protocol being used, either IP, TCP, or UDP
- **Source IP Address:** The IP address of the source of the data packet
- **Source Port:** The port the source host is using for communication, this can be a number between 0 and 65534, or can be any.
- **Destination IP Address:** The IP address of the destination of the data packet
- **Destination Port:** The port the destination host is using for communication, this can be a number between 0 and 65534, or can be any.
- **Action:** The action the firewall is to take, this can be either deny or accept.

1. tcp 10.0.0.1 80 200.150.150.150 any deny
2. tcp 10.0.0.1/16 any 200.150.150.150 any accept
3. tcp 192.168.0.5 any 123.100.100.100 80 deny
4. tcp 180.100.100.5/24 any any any accept
5. tcp 192.168.0.6 any 123.100.100.100 80 deny
6. tcp 140.192.37.0/24 161.120.33.40 80 accept
7. tcp 192.168.0.7 any 123.100.100.100 80 deny

Figure 3-3. The use of the generic syntax to define an example firewall policy.
3.4 Policy Anomalies

Errors in the rules within firewall policies can produce anomalies, furthermore research suggests that all firewalls contain anomalies (Wool, 2004). Anomalies are discovered within firewall policies during the modelling process. The process of modelling a firewall involves the establishment of relationships between rules. Once relationships are defined between the rules, policy violations can be determined. If two rules are completely unrelated to each other and all other rules in the rule set, their order is irrelevant (Al-Shaer et al, 2004), conversely if rules are in relation to each other, or other rules in the set, their order needs to be considered, as they may cause anomalies. Al-Shaer et al., and Yuan et al., both define the following anomalies, shadowing anomaly, correlation anomaly, generalisation anomaly, and redundancy anomaly.

3.4.1 Shadowing Anomaly

A rule can be considered shadowed when a preceding rule in the policy matches all the packets that the rule attempts to match, such that the rule is never processed and both rules have contradictory actions (Yuan et al, 2006. Al-Shaer et al, 2004). In the example illustrated in Figure 3-4, rule 10 will never be processed. Rule 10 is specific to a host within the network, while rule 1 applies its action to all traffic from the entire network range.

Shadowing anomaly defined by Al-Shaer et al., can be expressed using the following formal definition: Rule \( R_y \) is shadowed by rule \( R_x \) if it is a subset match to \( R_y \) and the action of rule \( R_x \) is opposite to \( R_y \).

\[ \forall: R_x[a] \subseteq R_y[a] \text{ and } R_x[b] \subseteq R_y[b] \text{ and } R_x[c] \neq R_y[c] \]

where \( a, b \in \{ \text{Source IP, Source Port, Destination IP, Destination Port} \} \)

and \( c \in \{ \text{action} \} \)

1. tcp 192.168.0.0/24 any 10.100.100.100 80 deny
   ...
   ...
   ...
10. tcp 192.168.0.30 any 10.100.100.100 80 accept

Figure 3-4. Shadowing Anomaly

3.4.2 Correlation Anomaly

If one rule matches some of the packets that another rule may capture, and that other rule matches some packets, which the original rule captures, the rules may be in correlation. However, the actions of the two rules need to be different in a correlation anomaly (Yuan et al, 2006. Al-Shaer et al, 2004). Figure 3-5 illustrates an example of a correlation anomaly, rule 1 matches some of the packets which rule 10 matches, and conversely rule 10 matches some packets, which rule 1 matches.
Correlation anomaly defined by Al-Shaer et al., can be expressed using the following formal definition: Rule Rx is in correlation rule Ry, if rule Rx has a partial superset match to Ry, and Ry has a partial Superset match to Rx, and their actions are different.

∀: \( Rx[a] \subseteq Ry[a] \) and \( Rx[b] \supseteq Ry[b] \) and \( Rx[c] \neq Ry[c] \)  
where \( a, b \in \{\text{Source IP, Source Port, Destination IP, Destination Port}\} \)  
and \( c \in \{\text{action}\} \)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Protocol</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>tcp</td>
<td>192.168.0.5</td>
<td>any</td>
<td>80</td>
<td>deny</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>tcp</td>
<td>any</td>
<td>10.100.100.100</td>
<td>80</td>
<td>accept</td>
</tr>
</tbody>
</table>

Figure 3-5. Correlation Anomaly

Arguably, correlation is not an anomaly, and it is not necessary for correlations between rules to be identified. Figure 3-5 illustrated the definition of a correlation anomaly; however, there is nothing fundamentally wrong with rules 1 and 10. Rule 1 denies a specific host access to all web servers, while rule 10 allows the rest of the subnet access to a single specific web server.

3.4.3 Generalisation Anomaly

A rule can be considered a generalisation of another rule, if the second rule matches all the packets that the first rule matches, as well as packets from other addresses in the same subnet. Figure 3-6, illustrates an example of generalisation anomaly, rule 1 is generalised by rule 10, since rule 10 applies to the entire subnet, and uses an opposite action.

Generalisation anomaly defined by Al-Shaer et al, can be expressed using the following formal definition: Rule Rx is generalised by rule Ry, if rule Ry is a superset match of Rx and the actions of the rules are different (Al-Shaer et al. 2004, Yuan et al, 2006).

∀: \( Rx[a] \supseteq Ry[a] \) and \( Rx[b] \supseteq Ry[b] \) and \( Rx[c] \neq Ry[c] \)  
where \( a, b \in \{\text{Source IP, Source Port, Destination IP, Destination Port}\} \)  
and \( c \in \{\text{action}\} \)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Protocol</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>tcp</td>
<td>192.168.0.5</td>
<td>10.100.100.100</td>
<td>80</td>
<td>deny</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>tcp</td>
<td>192.168.0.0/24</td>
<td>10.100.100.100</td>
<td>80</td>
<td>accept</td>
</tr>
</tbody>
</table>

Figure 3-6. Generalisation Anomaly
Arguably, generalisation is not an anomaly, and it is not necessary for generalisations between rules to be identified. Figure 3-6 illustrated the definition of a generalisation anomaly; however, there is nothing fundamentally wrong with rules 1 and 10. Rule 1 denies a specific host, while rule 10 allows the rest of the subnet access to the web server.

### 3.4.4 Redundancy Anomaly

A rule can be considered redundant, if another rule matches the same packets, has the same filtering action and is not in relationships with any other rules in the policy. Figure 3-7, and Figure 3-8, illustrate examples of redundant rules.

Redundancy anomaly defined by Al-Shaer et al. can be expressed using the following formal definition: Rule Rx is redundant to rule Ry if, Rule Ry is a superset match of Rx and the actions of Ry and Rx are similar, and both rule Rx and Ry are not in relations with any other rules.

\[ \forall: \text{Rx}[a] \supseteq \text{Ry}[a] \text{ and } \text{Rx}[b] \supseteq \text{Ry}[b] \text{ and } \text{Rx}[c] = \text{Ry}[c] \]

where \( a, b \in \{\text{Source IP, Source Port, Destination IP, Destination Port}\} \) and \( c \in \{\text{action}\} \)

![Figure 3-7. Redundancy Anomaly 1](image1)

![Figure 3-8. Redundancy Anomaly 2](image2)

### 3.6 Computer Law

In order for a successful security policy implementation, the implications of the law need consideration. Computer crime refers to criminal activity involving computer networks, being used in, or being the victim of an attack. As such, there are laws in place to protect organisations, and aid in successful prosecution of perpetrators of cyber crime. At present, the Computer Misuse Act (CMA) 1990 allows for the prosecution of perpetrators of Distributed Denial of Service (DDoS) however, such cases are rare. The Computer Misuse act is almost two decades old, and during its
inception, the primary threats perceived were to be unauthorised access and manipulation to data (Worthy et al, 2007). However, the CMA fails to identify threats, which have become apparent in the past decade. Recently though, there was a successful prosecution under the CMA. It was in the case of David Lennon (G. Kon and P. Church, 2006). However, this case had very specific circumstances, and the judge in question took the approach of common sense rather than application of set legislation in the proceedings. Thus, it is unlikely that the same approach and conclusions can be drawn from other cases, and it necessary for amendments to be made to the CMA.

Almost every respondent from industry told us that the CMA [Computer Misuse Act 1990] is not adequate for dealing with DoS [Denial-of-Service] and DDoS [Distributed Denial-of-Service] attacks.1

All Party Parliamentary Internet Group, June 2004.

Intruders are constantly using sophisticated techniques to commit computer crime and DDoS attacks faced by organisations are increasing daily. A recent security report revealed that during the first 6 months of 2006 Semantic observed 6110 DDoS attacks per day and cited that the UK is the third most targeted country for DDoS attacks (Symantec Internet Security Threat Report, 2006). Furthermore, 96% of ISPs within the UK state that DoS attacks originating from subverted hosts give their business most concern (StreamShield Network Survey, July 2006)

Due to this inadequacy in current legislation, and phenomenal increase in threats, amendments to the Computer Misuse Act have been proposed. The key offence being considered is an “impairment offence” (Worthy et al. 2007).

The new offence requires the impairment of a computer. The CMA offence replaced by the Impairment Offence tested whether data had been modified. This was a reasonably objective requirement that could be assessed by reference to data records/usage logs, etc.

Suggested action for IT users:

* All businesses should consider how “impairment” might be measured internally. IT departments should consider what systems performance and usage logs are kept and whether these records can distinguish day-to-day performance fluctuations from disruption caused by external sources.

* Data retention policies should also be reviewed. Typically, systems performance records are transient or are only kept for short periods of time – companies should consider retaining logs relating to business-critical systems for longer periods.

John Worthy, Martin Fanning 2007
It is suggested, that businesses need to investigate the performance of their existing network. It will be necessary for an acceptable benchmark performance for the network is established. In order to successfully prosecute, it will be necessary for the performance of the network under normal conditions to be presented, and contrasted against the impaired performance noticed under DDoS conditions. Furthermore, it will no longer be acceptable for organisations to build a network and assume it works well. A poorly performing network may not be able to observe a change in performance under DDoS. In order for successful prosecution, it will be necessary to produce an accountable difference in performance under normal, and DDoS conditions. Therefore, organisations need to continually audit existing systems, and, where necessary, rectify performance issues. Finally, retention needs to be considered, as many organisations do not hold onto performance analysis results for long periods. With regard to critical systems, logs should be kept for longer periods, and their accuracy and accountability should be continually audited.

3.7 Performance and Accountability

As discussed in Section 3.3, rule sets can easily contain anomalies, and redundant rules. By highlighting these anomalies to the administrator, they can be addressed, and thus an optimised firewall set can be deployed across the network. However Section 3.4 investigates the changes being made to the Computer Misuse Act and how this will require IT systems to be accountable and regularly audited. In the filed of forensic computing much work has been done in the logging and accountability of networking devices and the impact of logging on the system (J. Graves 2006). Graves argues that there is a balance of needs in a an accountable system, Figure 3-9.

In any system, there needs to be a balance between security, Quality of Service (QoS) and Quality of Evidence. Security needs to be implemented in such a way that the system can still be operated by its users, too many security implementations can result in a system which reduces the users productivity, that infringing on quality of service (Graves et al. 2006). Likewise focusing quality of evidence needs to be to an acceptable standard, that is it accountable in the event of a forensic investigation or if logs need to be produced as evidence, without infringing on quality of service to the end users of the system (Graves et al. 2006). Finally, quality of service needs to be at a level acceptable for employees, without compromising quality of service and evidence. Figure 3-10 illustrates an example of an un-optimised rule set which may be used in a firewall policy.
In Figure 9, rules 1 and 10 are in question. Assuming that there are no rules between rules 1 and 10, which involve rule 1 or rule 10, rule 1, can be considered redundant to rule 10. Rule 10 indicates that the whole 192.168.0.0/24 subnet will be blocked from accessing the web server at 123.100.100.100. However, the 192.168.0.0/24 subnet includes rule 1. The rules in this policy can be said to be valid, however it is clear that by removing rule 1 from the firewall policy, the needs of the security policy are still met. Removing rule 1 from the rule-set, some accountability is lost; rule 1 is specific to the individual host, while rule 10 is general to an entire subnet.

### 3.8 Conclusion

Arguably, the role of the network administrator is becoming more complex with regard to successful and efficient firewall implementations. The creation of firewall policies is seen to be commonly error prone process and an area requiring attention (Wool 2004). By addressing firewall mis-configurations, performance and efficiency can be increased. The advent of computer misuse is ever increasing, and DDoS is a major threat to organisations in the UK. Legislative changes are underway to address the concerns organisations have. This will require proactive monitoring of existing systems performance and auditing to be undertaken by administrators. A fine boundary exists, between performance and accountability. By increasing the performance of a system some accountability can be lost. In the event of a DDoS, it is this accountability of a system, which allows for successful prosecution of DDoS perpetrators (Worthy, 2007). If the administrator implements optimisation techniques as discussed, placement of logging agents such as Snort before the firewall may be necessary to ensure accountability.

The work by Al-Shaer et al, has its shortcoming in that it does not use CIDR notation. Furthermore, both Yuan et al, and Al-Shaer et al, define the correlation and generalisation of rules as anomalies. As discussed the rules in question in correlation and generalisation can be considered perfectly valid, if they do implement the security policy requirements. Arguably the mathematics behind correlation and generalisation anomalies is valid, however this purely technical approach is “out of touch” with the actual rule implementation on a firewall. Furthermore the work by Guttman et al. 1997 was identified to be tied down to a specific hardware vendor.

The design of the prototype system will use these findings in order to overcome the shortcomings of existing work, while building on the generic syntax and anomaly definitions by Al-Shaer.
4. Design

4.1 Introduction

This chapter outlines the various design considerations to be taken. The evaluation of previous work influences the design of the system and experiments to be performed. This chapter provides the foundations for the implementation of the system, in a focussed manner.

4.2 Generic Firewall Compiler and Modeller Design

The literature review has identified various functionality to be considered in the development of the prototype. Furthermore, extended functionality has been identified for implementation, as well as a novel approach to rule set optimisation is presented – Rule Crunching.

- Generic Syntax to Specific Syntax Compiler
- Anomaly Discovery
- Rule Crunching

In order to achieve the functionality identified, the Generic Firewall Compiler and Modeller will implement the following classes/objects:

- MainMenu.cs – The GUI and core of the system
- Rule_set.cs – The arraylist containing all the rules
- Rule.cs – The rule object

4.2.1 Generic Syntax

The presented work incorporates and extends on the generic syntax proposed in the Firewall Policy Advisor (Figure 4-1). The range of the parameters that can be used in the compiler needs to be completed. Research has identified that the syntax used in the Firewall Policy advisor by Al-Shaer did not incorporate CIDR notation. Thus, it is important to implement such functionality, as the majority of networks today use CIDR notation. The literature review also showed that the work by Guttman was specific to Cisco routers running IOS release 10, which is outdated. Thus, it is appropriate that the compiler is capable of producing syntax, which can be converted into platform specific syntax.

```
<order><protocol><s_ip><s_port><d_ip><d_port><action>
```

Figure 4-1. Generic Syntax (Al-Shaer et al, 2004)
4.2.2 Anomaly Discovery

During the literature review, the concept of firewall anomalies was identified. The anomaly discovery process is based on the definitions of anomalies by Al-Shaer et al and Yuan et al. However, the literature review revealed that the definitions of a Generalisation and Correlation anomalies are not necessarily errors, but rather warnings. Thus, the presented work will not identify these errors, but will present them as notices to the user.

The facility for the user to remove redundancy and shadowing anomalies will be presented, as these anomalies are considered safe to remove and do not affect the firewall policy.

4.2.3 Rule Crunching

Rule crunching provides a novel approach to policy optimisation, which reduces firewall rule-set size, while keeping consistency. The concept of rule crunching involves taking a number of rules from within the policy that is replaceable by one rule, which will produce the same result, thus keeping the security policy consistent. Consider Figure 4-3.

1. tcp 10.0.0.1 80 200.150.150.150 any deny
2. tcp 10.0.0.1/16 any 200.150.150.150 any accept
3. tcp 192.168.0.5 any 123.100.100.100 80 deny
4. tcp 180.100.100.5/24 any any any accept
5. tcp 192.168.0.6 any 123.100.100.100 80 deny
6. tcp 140.192.37.0/24 161.120.33.40 53 accept
7. tcp 192.168.0.7 any 123.100.100.100 80 deny

Figure 4-3. Example Firewall Policy

The firewall policy described in Figure 4-3 does not contain any inconsistencies or anomalies. However, consider rules 3, 5 and 7. Each of these rules refers to specific hosts, and each IP Address is contiguous. Figure 4-4 shows the source addresses of rules 3, 5, and 7 once they are converted to binary.
Rule 1. 11000000.10101000.00000000.00000101
Rule 2. 11000000.10101000.00000000.00000110
Rule 3. 11000000.10101000.00000000.00000111

Figure 4-4. Source Addresses Converted to Binary

By converting the addresses to binary it can be seen that it’s the final bits of the host address which differs. It is these final bits of each address which are considered for crunching. In order to determine where the addresses differentiate, the binary XOR operator is used. The XOR bit operator returns true where there is a difference in bit values and false where bit values are the same. The XOR operator is applied to the numerically first and last addresses, in this example, rules 1 and 3 from figure 4-5:

Rule 1. 11000000.10101000.00000000.00000101
⊕
Rule 3. 11000000.10101000.00000000.00000111
→ 00000000.00000000.00000000.00000010

Figure 4-5. Result of AND operation on binary addresses

The result of the XOR operation, as can be seen in Figure 4-5, and it can be noted that the last octet is where the bit value differences in the addresses is seen. The result from this calculation is used to create the crunch rule which will replace the host specific rules. All octets in the result which are made up of 0’s are considered the same, it is the octets which contain 1’s which we are concerned with, in this can octet 4. The position of the last 0 before a 1 which defines the network address for the new rule, in the example the position of the last 0 is position 29. The new address with use the first three octets of the original rules, and the fourth octet will contain all zero’s except in the position of the last 0, which is replaced with a 1. This is shown in Figure 4-6.

New rule – binary: 11000000.10101000.00000000.00000100
New rule – dotted decimal: 192.168.0.4

Figure 4-6. Creating the new crunch rule

The rule created is the network address, it is now necessary to create the network mask. The network mask is derived by calculating the position of the last “1” in the whole address when in binary. It can be noted from figure 10 above, that the position of the last “1” is 29. Thus the subnet mask is /29. And the final crunched rule can be written in CIDR notation as show in Figure 4-7.

192.168.0.4/29

Figure 4-7. Crunch Rule

It can be seen that the initial 3 rules can be crunched into a network address with subnet mask containing those addresses. However, it needs to be noted that the 192.168.0.4/29 subnet also contains the host address 192.168.0.4, which was not in the original firewall policy. The example shown is convenient in that each host
address increases by one. However this is not a typical case, and can be considered a best case scenario. Consider the example below in Figure 4-8.

<table>
<thead>
<tr>
<th>Rule Number</th>
<th>Protocol</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tcp</td>
<td>192.168.0.1</td>
<td>123.100.100.100</td>
<td>80</td>
<td>deny</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>tcp</td>
<td>192.168.0.255</td>
<td>123.100.100.100</td>
<td>80</td>
<td>deny</td>
</tr>
</tbody>
</table>

Figure 4-8. Crunching worst case scenario

In Figure 4-8 it can be noted that the difference in host addresses is vast, with over 250 hosts addresses between the two addresses. If crunching were to be applied using the techniques described the resultant rule would be produced, as described in Figure 4-9.

![](tcp_192.168.0.0/24_any_123.100.100.100_80_deny.png)

Figure 4-9. Worst case scenario rule crunch result

The result of this crunching will save 1 rule in the firewall rule policy, however 253 hosts have been denied access to the web server located at 123.100.100.100 which may not have been intended.

### 4.3 Experiment Design

#### 4.3.1 Experiment 1 – Rule Verification Timing

Once rules have been entered into the compiler, in the generic syntax, it is necessary to verify that the syntax is error free. In order to check that rules comply with the defined syntax, it is necessary to work through each rule and check each of the parameters. If it is found that a parameter is incorrect, this needs to be noted for the user’s attention. If no errors are found, this should also be shown to the user. If a rule is considered error free, the rule class is used to create a rule object and add it to the array list of rules.

The task of checking each parameter within the rules, is a time consuming task. The purpose of this experiment is to investigate the time taken to verify rule sets on various sizes, and graph the performance of the verification algorithm.

Rule set sizes used:

- 100 Rules
- 200 Rules
- 500 Rules
- 1000 Rules
- 2000 Rules
- 5000 Rules
- 10000 Rules
In order to measure the rule verification time additional software is required. The *PrecisionStopWatch* is used as opposed to the default timer methods available within .NET. The *PrecisionStopWatch* is a class available within the Centre for Mobile Computing and Security useful Networking Libraries - CMCS. Utilities Namespace, (Lionel Saliou & Kevin Chalmers, Centre for Mobile Computing and Security, Napier School of Computing, 2006)

### 4.3.2 Experiment 2 – Rule Anomaly Discovery Timing

Anomaly discovery is a task involving searching through the rules multiple time in order to model the firewall policy rules. Once rules have been verified and stored in memory, the anomaly discovery algorithm performs a nested for loop, during this looping each rule is checked for anomalies as defined in the literary review. This task can be expected to take considerably longer to complete than the verification stage, as rules are cast multiple times during the nested for loops. As in experiment 1, policies of various sizes are tested and performance graphed.

Rule set sizes used:

- 100 Rules
- 200 Rules
- 500 Rules
- 1000 Rules
- 2000 Rules
- 5000 Rules
- 10000 Rules
- 20000 Rules

Additional software required for this experiment includes the *PrecisionStopWatch* (CMCS. Utilities Namespace - Lionel Saliou & Kevin Chalmers, Centre for Mobile Computing and Security, Napier School of Computing, 2006)

### 4.3.3 Experiment 3 – Rule Crunching Timing

During this experiment, the rule-crunching algorithm is tested to see how long it takes to discover crunchable rules.

Rule set sizes used:

- 100 Rules
- 200 Rules
- 500 Rules
- 1000 Rules
• 2000 Rules
• 5000 Rules
• 10000 Rules
• 20000 Rules

Additional software required for this experiment includes the *PrecisionStopWatch* (CMCS.Utilities Namespace - Lionel Saliou & Kevin Chalmers, Centre for Mobile Computing and Security, Napier School of Computing, 2006)

### 4.3.4 Experiment 4 – Platform Specific Rule Generation Timing

This is the final timing experiment, and involves the compilation of firewall specific syntax. A for loop works through the array list of rules, casts the rule object and then calls methods for the compilation of Linux syntax, and Cisco syntax.

Rule set sizes used:

• 100 Rules
• 200 Rules
• 500 Rules
• 1000 Rules
• 2000 Rules
• 5000 Rules
• 10000 Rules
• 20000 Rules

Additional software required for this experiment includes the *PrecisionStopWatch* (CMCS.Utilities Namespace - Lionel Saliou & Kevin Chalmers, Centre for Mobile Computing and Security, Napier School of Computing, 2006)

### 4.3.5 Experiment 5 – A Linux Un-Optimised vs. Optimised Test

One of the most critical outcomes of the system, is it’s ability to discover anomalies. The removal of Anomalies which are considered safe to remove, such as redundant and shadowed rules, reduces the rule set size. Once anomalies have been removed from the rule set, it can be considered to be optimised.

The purpose of this experiment is to test the performance of a Linux firewall using an un-optimised rule set of 1501 rules. The same performance test will be replicated using the rule set after it has been optimised.

This experiment is undertaken using the automated evaluation environment presented by Saliou et al. 2007.
4.4 Conclusion

This chapter has provided a high-level approach to the development of the system, and the evaluation methods. It can be seen, that rule crunching has a role to play in reducing the size and manageability of firewall policies, especially in best-case scenarios where few or no new rules are added to the policy. However, in worst-case scenarios entire subnets can be blocked, while saving management of a single rule in the policy. The administrator needs to be aware of the consequences of crunching rules, and possible implications for other hosts on the network.

By testing the timing of the application, code efficiency can be noted, and whether the code execution time increase, for large size rule sets is exponential. During the final experiment, a real life test of the effectiveness of an optimised rule set versus and un-optimised rule set is carries out on a Linux firewall.
5. Implementation

5.1 Introduction

The development of the Generic Firewall Compiler and Modeller was chosen to be carried out in the Microsoft .NET environment, using the C# programming language. The .NET platform designed by Microsoft facilitates rapid development of Windows based applications with the controls provided in the .NET framework. Furthermore, the programmer does not have to deal with low-level issues such as memory management associated with some other programming languages, such as C++.

5.2 Generic Syntax Compiler

The Generic Syntax Compiler is the main deliverable of the system. The proposed syntax is used for modelling the firewall, and thus can be considered the foundation to further work involving anomaly discovery and rule crunching. The first step in developing the syntax compiler is to establish a list of syntax for each parameter, after reviewing the requirements of the compiler. Table 5-1 illustrates the parameters used for the syntax definition.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Source IP</th>
<th>Source Port</th>
<th>Destination IP</th>
<th>Destination Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Dotted Decimal Format</td>
<td>0-65000</td>
<td>Dotted Decimal Format</td>
<td>0-65000</td>
<td>accept</td>
</tr>
<tr>
<td>UDP</td>
<td>CIDR</td>
<td>any</td>
<td>CIDR</td>
<td>any</td>
<td>deny</td>
</tr>
<tr>
<td>IP</td>
<td>Star Notation</td>
<td>Star Notation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5-1. Syntax Definition

The interface provides an area for the user to enter rules in the generic syntax (Figure 5-2). After the user has entered all the rules, they are verified to ensure correct use of syntax. During verification each component of the rule is checked that it is using the correct syntax, and that values are within valid ranges. Each rule is processed sequentially and if a rule is found to contain errors these are reported to the user for rectification and re-verification (Figure 5-3). A rule is considered to be correct, if it does not contain any syntax errors. During verification, error free rules are cast as rule objects and added to the rule-set array list.
Figure 5-2. User Interface

Figure 5-3. Verification Flow

Diagram showing the verification flow with steps:
1. Protocol
2. Source Address
3. Source Port
4. Destination Address
5. Destination Port
6. Action

Invalid Rule
Valid Rule
5.3 Anomaly Discovery

The anomaly discovery section of the application was given its own tab as it required a substantial amount of space for displaying information (Figure 5-4. Rule Anomalies Tabre 5-4). The rule anomaly discovery algorithm worked by iterating through the array list containing all the rule objects. This was done using two nested for loops in order to do a comparison of every single rule with each other. During the comparison process, the various elements of the rule objects were compared to each other. Rule duplication and, and rules where only the actions varied require simple logical comparisons. In order to establish subsets and superset relations, bitwise operations were required. One of the first obstacles reached was that the .NET framework does not incorporate any methods for converting data from decimal to binary, and vice-versa. Thus it was necessary to write methods to do the conversion.

The ConvertToBinary method was written to convert decimal numbers into binary, and uses a novel recursive approach to the method.

```csharp
private string ConvertToBinary(int number)
{
    if (number == 0)
        return "";
    if ((number % 2) == 0)
    {
        return ConvertToBinary(number / 2) + "0";
    }
    else
    {
        return ConvertToBinary((number - 1) / 2) + "1";
    }
}
```

The binaryToDecimal method was written to convert number from binary into decimal, shown below.

```csharp
private int binaryToDecimal(string binary)
{
    int ii = 0;
    double result = 0;
    //for loop run to length of input binary string
    for (int i = binary.Length - 1; i >= 0; i--)
    {
        if (binary[i] == '1')
        {
            result = result + (Math.Pow(2, ii));
        }
        ii++;
    }
    return Convert.ToInt32(result);
}
```

Furthermore, rules were displayed to the user for deletion. If the user opts to delete the redundancy anomalies, which have discovered it is necessary to run the anomaly discovery algorithm again. This is due to the algorithm implemented, only identifying
redundant rules, which are in relation to each other, and no other rules. If a rule shadows two redundant rules is discovered, the shadowing of the two rules is identified. The redundancy is not considered as the rules are in multiple relations with each other.

**Figure 5-4. Rule Anomalies Tab**

### 5.4 Rule Crunching

The novel rule crunching element of the application got its own tab as it produced a similar amount of output as the anomaly discovery algorithm, and kept with the consistency of the application (Figure 5-5). The rule crunching element of the application is only able to identify crunchable rules, based on source address information. The process of discovering crunchable rules required an understanding of all the rules in the compiler and the relations they have to one another. If a rule is only in relation to other rules which are in turn related to the original rule and associated rules, based on source address, and all other parameters are the same, the rules were considered for crunching. In order to implement such relationship building, the anomaly discovery algorithm was modified, as well as the rule object. The “rule.cs” class file was modified to include an array list, which includes all relations each rule is in as discovered during the anomaly discovery algorithm. This added to the execution time for the anomaly discovery algorithm, but saved time while discovering crunchable rules, as relations would not need to be calculated again. Furthermore the rule crunching algorithm only detects single hosts, and does not work in the identification of crunchable subnets.
Figure 5-5. Rule Crunching Tab
6. Experiment Data Analysis

6.1 Introduction

This chapter will discuss the results of the experiments outlined in Section 4. The purpose of the experiments is to evaluate the various functionalities of the system. The experiments undertaken are considered unique as existing work in the field presented in Chapter 3 did not indicate any similar tests.

6.2 Experiments

6.2.1 Experiment 1 – Rule Verification Timing

During this experiment, the time it takes the compiler to verify rules is investigated. Rule-sets of varying sizes, from 100 rules to 20000 rules, are tested and timed. The rules are generated using a random generator method, which creates TCP rules, with completely random source address, destination address, ports and actions.

![Rule Verification Timing Results](image)

Figure 6-1. Rule Verification Timing Results

Figure 6-1 illustrates the results of the experiments. The increase in time taken to verify rules is exponential. In the case of the rule set consisting of 20000 rules, the rule verification time took over 53 minutes. The results here are interesting in that the graph did not have an even gradient, as there are no nested loops within the verification algorithm. The results suggest that as array list size increases, so performance decreases at an exponential rate.
6.2.2 Experiment 2 – Rule Anomaly Discovery Timing

The purpose of this experiment is to test the execution time of the anomaly discovery algorithm. This test is similar to the Rule Verification Test, in that the rule set used is once again completely random.

![Anomalies Discovery](image)

Figure 6-2. Anomaly Discovery Timing Results

Figure 6-2 illustrates the results of the Anomaly Discovery experiment. The results indicate an exponential increase in the time taken to calculate anomalies as the rule set size increases. This is inline with predictions that there would be an exponential degradation in performance as the use of nested loops to develop relationships between rules would produce exponential output.

6.2.3 Experiment 3 – Rule Crunching Timing

During the Rule Crunching Timing experiment, as in the previous two tests, rule sets generated randomly in varying sizes were tested to find crunchable rules. This test needs to be taken into perspective, as the rule crunching algorithm is not complete and only finds crunchable rules based on source address information. However, the results are once again exponential. This is due to the use of nested loops. The calculations, which are executed during the loops, are simple logical comparison test, which are not CPU intensive. This is reflected in the results, in that even when a rule set of 20 000 rules was tested, execution time was just over a minute (Figure 6-3)
6.2.4 Experiment 4 – Platform Specific Rule Generation Timing

During this experiment, the final timing experiment, the rule generation algorithms are tested. As in the previous tests an entirely random set of firewall rule sets is used. Once again rule sets of 100, 200, 500, 1000, 2000, 5000, 10000, and 20000 are used.

Figure 6-4 illustrates the impact bigger rule sets has on the performance of the rule generation algorithm. During this algorithm, there are no nested loops, thus it would
be expected that the graph would have a consistent gradient. This test backs up the conclusion from the results from experiment 1, which did not use any nested loops in its verification algorithm. This backs up the conclusion from experiment 1, that as the array list size increases, so the performance decreases exponentially.

6.2.5 Experiment 5 – A Linux Un-Optimised vs. Optimised Test

This experiment presented a unique opportunity to evaluate the performance of an optimises rule set. As the system did not fully implement rule crunching, an optimised rule set was one that was considered not to contain anomalies.

Unlike previous tests, the randomness in the random rule generation routine was manipulated to produce a certain number of anomalies. A completely random rule set wouldn’t produce any anomalies, as the constraints are too big. Thus, only 15 ports were included in this test, and the source and destination addresses only had variants in their final octets.

During this test 1500 rules were generated, and tested for anomalies. After optimisation was applied, 253 anomalies were identified and removed. The compiler produced the corresponding Linux firewall rule sets, which were loaded onto a Linux firewall. Then the performance of the firewall with either rule-sets was measured. The security overhead in terms of bandwidth consumption is then measured. The environment is also capable of controlling the level of traffic usage crossing the firewall. Hence the measurements are repeated for varying usage from null to 50 percent in 10 percent increments. Figure 6-5 illustrates the security overhead variation noticed between the optimised and the un-optimised rule-sets while a 100Mbps network is used.
6.3 Conclusions

The results gained from the experiments gave an indication on various aspects of the applications performance. During experiments 1, 2, 3, 4 and performance was based on how long the system took to complete tasks. For these experiments as the rule-set size increases, it is expected that the execution time will increase. In the case of experiments 2 and 3, the system’s methods, which are being evaluated, contain nested loops. Due to the nature of nested loops producing exponential iterations, the results would be expected to reflect this, which they do. However experiments 1, and 4 evaluate methods, which do not contain nested loops, and a single iteration through the list of rules is completed. It is interesting to note, that in both these experiments the results produced were exponential. This leads to the conclusion that as an array size increases, so the performance (in this case execution time) depredation is exponential.

Experiment 5 was based on a randomly generated rule set, through the use of a random algorithm which was manipulated to include anomalies. It must be stressed though that the test does not reflect the policy rule-set which would be encountered in a real-world scenario. Due to the security implications, no organisation would be willing to release its firewall policy to the public, thus this test provides a results which may not reflect real-world results.
7. Conclusions

The main aim of this project was to create a Generic Firewall Compiler and Modeller. The key objective was defining a complete range of syntax, which can be compiled into vendor specific syntax. The presented work has extended on the work done by AL-Shaer et al. through the identification of shortcoming in their work. Furthermore Work has been done in firewall modelling, and an implementation of anomaly discovery has been incorporated. By investigating the real world implementation of firewall policies and is what mathematically considered to be an anomaly, the presented work is more reflective of what the administrator would naturally implement. The novel concept of rule crunching has been implemented, and is capable of discovering crunchable rules.

It needs to be highlighted however, that during experiment 5 - A Linux Un-Optimised vs. Optimised Test, the original rule set had been generated by an algorithm which had been manipulated to produce a certain number of anomalies. The randomness of an entirely random rule set would otherwise not contain few, if any anomalies. Unfortunately organisational firewall policies are not available for modelling and anomaly discovery as organisations security is at stake. Thus an approach to force a certain number of anomalies was used, as research suggests all firewalls contain anomalies (Wool, 2005)

The presented work is capable of discovering anomalies, and provides a proof of concept implementation of rule crunching discovery. The complete process of crunching rules was not implemented due to complexity and time constraints. Thus, the suggestion should be made that a fuller implementation of the rule-crunching concept presented is undertaken in the future. Furthermore, due to the rapid prototyping software lifecycle used in developing the system, it is suggested that the core engine of the system could be decoupled from the graphical user interface (GUI). This would allow for code mobility and the development of an entirely platform independent application.
8. References


9. Appendices

Appendix 1 – Table of Subnet Masks

<table>
<thead>
<tr>
<th>CIDR</th>
<th>Hosts*</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>/32</td>
<td>1</td>
<td>255.255.255.255</td>
</tr>
<tr>
<td>/31</td>
<td>2</td>
<td>255.255.255.254</td>
</tr>
<tr>
<td>/30</td>
<td>4</td>
<td>255.255.255.252</td>
</tr>
<tr>
<td>/29</td>
<td>8</td>
<td>255.255.255.248</td>
</tr>
<tr>
<td>/28</td>
<td>16</td>
<td>255.255.255.240</td>
</tr>
<tr>
<td>/27</td>
<td>32</td>
<td>255.255.255.224</td>
</tr>
<tr>
<td>/26</td>
<td>64</td>
<td>255.255.255.192</td>
</tr>
<tr>
<td>/25</td>
<td>128</td>
<td>255.255.255.128</td>
</tr>
<tr>
<td>/24</td>
<td>256</td>
<td>255.255.255.000</td>
</tr>
<tr>
<td>/23</td>
<td>512</td>
<td>255.255.254.000</td>
</tr>
<tr>
<td>/22</td>
<td>1024</td>
<td>255.255.252.000</td>
</tr>
<tr>
<td>/21</td>
<td>2048</td>
<td>255.255.248.000</td>
</tr>
<tr>
<td>/20</td>
<td>4096</td>
<td>255.255.240.000</td>
</tr>
<tr>
<td>/19</td>
<td>8192</td>
<td>255.255.224.000</td>
</tr>
<tr>
<td>/18</td>
<td>16384</td>
<td>255.255.192.000</td>
</tr>
<tr>
<td>/17</td>
<td>32768</td>
<td>255.255.128.000</td>
</tr>
<tr>
<td>/16</td>
<td>65536</td>
<td>255.255.000.000</td>
</tr>
<tr>
<td>/15</td>
<td>131072</td>
<td>255.254.000.000</td>
</tr>
<tr>
<td>/14</td>
<td>262144</td>
<td>255.252.000.000</td>
</tr>
<tr>
<td>/13</td>
<td>524288</td>
<td>255.248.000.000</td>
</tr>
<tr>
<td>/12</td>
<td>1048576</td>
<td>255.240.000.000</td>
</tr>
<tr>
<td>/11</td>
<td>2097152</td>
<td>255.224.000.000</td>
</tr>
<tr>
<td>/10</td>
<td>4194304</td>
<td>255.192.000.000</td>
</tr>
<tr>
<td>/9</td>
<td>8388608</td>
<td>255.128.000.000</td>
</tr>
<tr>
<td>/8</td>
<td>16777216</td>
<td>255.000.000.000</td>
</tr>
<tr>
<td>/7</td>
<td>33554432</td>
<td>254.000.000.000</td>
</tr>
<tr>
<td>/6</td>
<td>67108864</td>
<td>252.000.000.000</td>
</tr>
<tr>
<td>/5</td>
<td>134217728</td>
<td>248.000.000.000</td>
</tr>
<tr>
<td>/4</td>
<td>268435456</td>
<td>240.000.000.000</td>
</tr>
<tr>
<td>/3</td>
<td>536870912</td>
<td>224.000.000.000</td>
</tr>
<tr>
<td>/2</td>
<td>1073741824</td>
<td>192.000.000.000</td>
</tr>
<tr>
<td>/1</td>
<td>2147483648</td>
<td>128.000.000.000</td>
</tr>
<tr>
<td>/0</td>
<td>4294967296</td>
<td>000.000.000.000</td>
</tr>
</tbody>
</table>
Appendix 2 – Experiment Results

Timing Results.

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify</td>
<td>0.1397</td>
<td>0.4393</td>
<td>2.1935</td>
<td>8.4116</td>
</tr>
<tr>
<td>Anomalies</td>
<td>0.005</td>
<td>0.1366</td>
<td>0.8481</td>
<td>3.5455</td>
</tr>
<tr>
<td>Crunch</td>
<td>0.0303</td>
<td>0.002</td>
<td>0.0411</td>
<td>0.0679</td>
</tr>
<tr>
<td>Output</td>
<td>0.009</td>
<td>0.0146</td>
<td>0.0737</td>
<td>0.3047</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
<th>20000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify</td>
<td>33.0507</td>
<td>202.7487</td>
<td>837.5192</td>
<td>3232.563</td>
</tr>
<tr>
<td>Anomalies</td>
<td>14.71</td>
<td>94.4358</td>
<td>339.1049</td>
<td>1489.444</td>
</tr>
<tr>
<td>Crunch</td>
<td>0.25514</td>
<td>3.4058</td>
<td>13.955</td>
<td>60.8967</td>
</tr>
<tr>
<td>Output</td>
<td>1.4981</td>
<td>10.203</td>
<td>34.9425</td>
<td>166.5291</td>
</tr>
</tbody>
</table>