# POLICY ANOMALY REPORTING FOR DISTRIBUTED FIREWALLS

**A Thesis Submitted to**
**The Graduate School of Engineering and Sciences of**
**İzmir Institute of Technology**
**In Partial Fulfillment of the Requirements for the Degree of**

**MASTER OF SCIENCE**

**in Computer Software**

**by**
**Füsun ÇETİN**

**December 2007**
**İZMİR**

We approve the thesis of **Füsun ÇETİN**

_____

**Assist. Prof. Dr. Tuğkan TUĞLULAR**
Supervisor

_____

**Prof. Dr. Şaban EREN**
Committee Member

_____

**Assoc. Prof. Dr. Ahmet KOLTUKSUZ**
Committee Member

**18.12.2007**

**Date**

_____

**Prof. Dr. Sıtkı AYTAÇ**
Head of the Computer Engineering
of Department

_____

**Prof. Dr. Hasan BÖKE**
Dean of the Graduate School of
Engineering and Sciences

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Assist. Prof. Dr. Tuğkan TUĞLULAR, for his guidance, patience and encouragement. He was the one who supported me when I was in trouble with critical decisions. His valuable support and confidence have been the driving force of this thesis work.

I would also like to thank Sevgi Uslu, Ezgi Samanlı and Oğuz Yarımtepe who cooperated with me in many studies.

Finally, I should thank to my parents who always supported me throughout my education in my graduate study.

# ABSTRACT

POLICY ANOMALY REPORTING FOR DISTRIBUTED FIREWALLS

Firewall is a protective device which is installed between two networks. Firewall functionality depends on the filtering rules and their order. All rule relations must be considered in order to determine correct rule order. In this thesis, anomaly discovery algorithms are implemented for single and distributed firewall environments in a software tool called “Policy Anomaly Checker”. A number of tests are performed using different policies and network topologies in order to obtain operational values of these algorithms.

# ÖZET

## DAĞITIK GÜVENLİK DUVARLARI İÇİN POLİTİKA ANOMALİ RAPORLAMA

Güvenlik duvarı iki network arasına kurulan koruyucu bir cihazdır. Güvenlik duvarının işlevselliği filtreleme kurallarına ve bu kuralların sırasına bağlıdır. Doğru kural sırasını belirlemek için kurallar arasındaki bütün ilişkiler dikkate alınmalıdır. Bu tezde tek ve dağıtık güvenlik duvarı ortamları için anomali bulma algoritmaları "Politika Anomali Belirleyicisi" adı verilen bir yazılım aracında uygulanmıştır. Bu algoritmaların operasyon değerlerinin belirlenmesi için farklı kural setleri ve ağ yapıları kullanılarak testler yapılmıştır.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure** **Page**

# LIST OF TABLES

**Table** **Page**

# CHAPTER 1

# INTRODUCTION

Firewalls are fundamental elements in network security. Most of the dangers of the Internet may be blocked by installing a firewall. Firewall is installed at a point where the internal network connects to the Internet. It filters out unacceptable traffic coming from or going to internal network. Filtering decision is based on a set of ordered rules. Correct operation of the firewall is dependent on the filtering rules

Administrator must consider all rule relations in order to determine of correct rule order. Increase in the number of the filtering rules results in the increase the potential of anomaly occurrence in the firewall policy. In single firewall environment policy may include intra-firewall anomalies, where the same packet may match more than one filtering rule. In distributed firewall environments, firewalls might also have inter-firewall anomalies when individual firewalls in the same path perform different filtering actions on the same traffic.

The main concern of this study is to implement anomaly discovery algorithms (Al-Shaer and Hamed 2004) for reporting an anomaly that might exist in any firewall policy in single and distributed firewall environments. In order to implement these algorithms a tree-based filtering representation is used. These algorithms are implemented using Java programming language in a software tool called “Policy Anomaly Checker”.

This thesis is organized as follows. Chapter 2 describes the single and distributed firewall environments. Firewall policies and their tree-based representation are also identified in this chapter. Then in order to determine anomalies among filtering rules, relations are described as well. Chapter 3 explains anomaly discovery algorithms for single and distributed firewalls. Design and implementation of “Policy Anomaly Checker” are also explained in this chapter. Chapter 4 describes experiments and evaluation which are done using “Policy Anomaly Checker”. Finally, Chapter 5 gives the conclusion of this thesis work.

# CHAPTER 2

# BACKGROUND

In this chapter firewalls and firewall policies are explored. In addition, distributed firewalls and policy anomalies are explained.

## 2.1. Firewalls

Firewall is a protective device which is installed between two networks. A firewall usually separates an internal network from the Internet (Figure2.1). Firewall restricts people to entering or leaving at a controlled point and it prevents attackers from getting close to other defenses. All traffic between two networks must pass through the firewall and only authorized traffic, defined by the local security policy, can be allowed to pass.

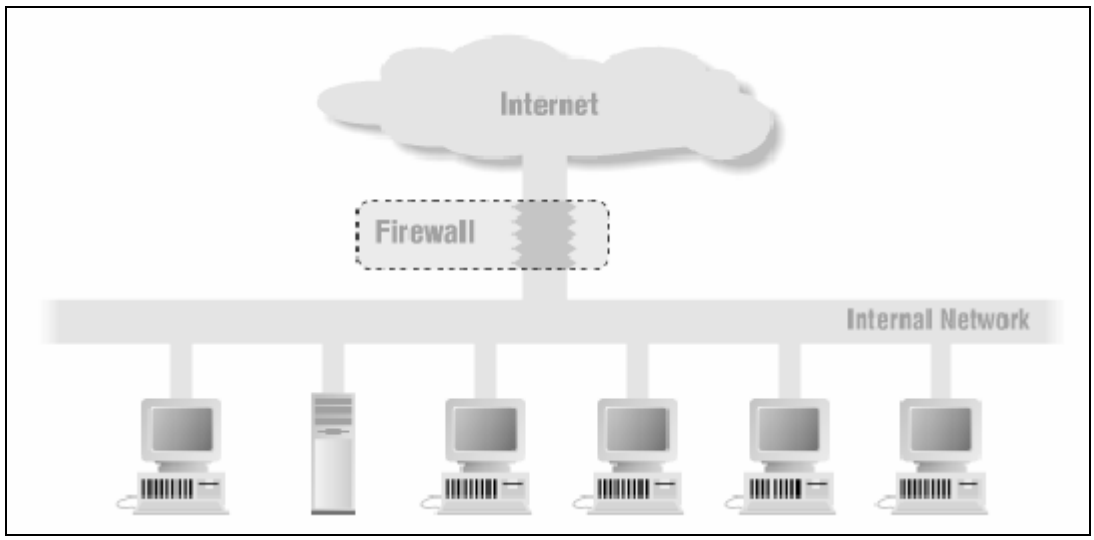



Figure 2.1. A Firewall Usually Separates an Internal Network from the Internet (Source: Chapman and Zwicky 1995)

The function of the firewall is to provide secure access to and from the internal network. Any packet that attempts to enter or leave at the entry point is first examined

by the firewall. Firewall decides either to accept or to discard the packet according to different fields of the packet.

Firewalls are included in network security approaches to protect internal networks. Internet firewall prevents the dangers of the Internet from spreading to internal network.

### 2.1.1. Types of Firewalls

There are several types of firewalls. Packet filter firewalls, stateful inspection firewalls, application-proxy gateway firewalls are fundamental types of firewalls.

#### 2.1.1.1. Packet Filter Firewalls

Packet filtering firewalls operate at Layer 3 of the OSI model. Packet filtering firewalls route IP packets selectively between internal and external hosts. IP packet's header contains four information; IP source address, IP destination address, IP protocol type and some characteristics of layer 4 such as the source and destination port. Firewall examines packets and allows or blocks them according to a rule set. Table 2.1 shows sample packet filter firewall rule set which blocks all tcp traffic coming from the network 192.168.1.* except http (Wack, et al. 2002).

Table 2.1. Sample Packet Filter Firewall Rule Set

| Order | Protocol | Source Address | Source Port | Destination Address | Destination Port | Action |
|---|---|---|---|---|---|---|
| 1 | tcp | 192.168.1.* | any | any | 80 | accept |
| 2 | tcp | 192.168.1.* | any | any | any | deny |

The rule set may be much larger and detailed for most firewalls. When a firewall accepts a packet, it examines source and destination addresses and ports of the packet. Then it starts at the top of the rule set and works down through the rules. When

a rule that permits or denies the packet is found, firewall passes the packet through the firewall or drops the packet without passing it through the firewall (Wack, et al. 2002).

#### 2.1.1.2. Stateful Inspection Firewalls

Stateful inspection firewalls are packet filters that incorporate added awareness of the OSI model data at Layer 4. A stateful firewall is able to hold in memory the state of the connection. It depends on the three-way handshake of the TCP protocol. When a client initiates a new connection, it sends a packet with the SYN bit set in the packet header. All packets with the SYN bit set are considered by the firewall as NEW connections. If the service which the client has requested is available on the server, the service will reply to the SYN packet with a packet in which both the SYN and the ACK bit are set. The client will then respond with a packet in which only the ACK bit is set, and the connection will enter the ESTABLISHED state. Such a firewall will pass all outgoing packets through but will only allow incoming packets if they are part of an ESTABLISHED connection (Wack, et al. 2002).

#### 2.1.1.3. Application Proxy Gateway Firewalls

Application-proxy gateway firewall is an application that runs on a firewall system. The user's client program talks to this proxy server instead of directly talk to the "real" server out on the Internet. The proxy server evaluates requests from the client and decides which to pass on and which to discard. If a request is approved, the proxy server talks to the real server on behalf of the client and proceeds to relay requests from the client to the real server and to relay the real server's answers back to the client (Chapman and Zwicky 1995).

### 2.2. Distributed Firewalls

Traditionally a firewall is installed as network gateway and same access policy is enforced for every computer on the network that is protected by the firewall. Same rule set is applied at the network boundary. This single point of interconnection between networks has some disadvantages. If the access control policy is incorrect, the protected network will be at risk. For example if malicious traffic is able to pass

through to firewall then all hosts on the network could be vulnerable to this malicious traffic. Also traditional firewall will not provide a protection for any attack from the same network. In addition a network firewall can be a bottleneck (Grant, et al. 2001).

Distributed firewall system is not a single system that acts as a gateway between two networks. By removing this single choke point, the network bottleneck is eliminated. In distributed firewall technology, an access control mechanism is distributed to each host which requires protection. Firewall functionality is removed from a single system. The unique access control for each of the network systems allows different levels of security to be implemented on computer system on the same network (Grant, et al. 2001).

A policy language, a policy distribution scheme and certificates are components of distributed firewall environment. Policy language defines what sort of connections are permitted or prohibited. Basically it is equivalent to packet filtering rules. Policy language should also support credentials, for delegation of rights and authentication purposes (Ioannidis, et al. 2000). While traditional firewalls usually use the IP address as an identifier, distributed firewalls use cryptographic certificates as identifier since distributed firewalls are independent of topology. Certificates enable the making decisions without knowledge of the physical location of the host. Public-key cryptography mechanisms are most often applied in contemporary implementations.

A sample policy is shown in Figure 2.2. In this sample if the certificate identifies the source as mailgw.example.com the packet will be accepted. If the certificate name is different, the packet will be dropped. (Bellovin 1999)

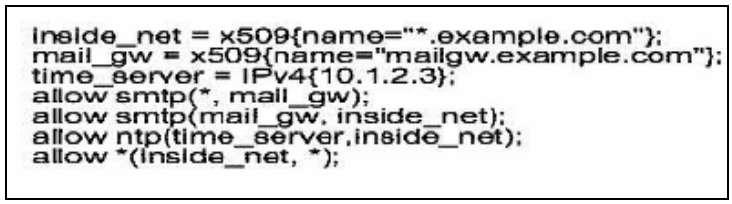



Figure 2.2. A Sample Policy Configuration File
(Source: Bellovin 1999)

Policy distribution scheme is used to enabling policy control from central point. Distribution scheme defines type of the security policy delegation to the members of

the network. Policies are distributed according to one of the following distribution scheme:

- Policies as well as credentials can be pushed to every single end point in the policy domain.
- Policies and credentials can be pulled from a trusted repository during initialization.
- Policies are pulled during initialization of the policy verifier whereas credentials for authentication mechanisms remain on a trusted repository and are requested whenever communication traffic is reaching a node from a yet unknown host.

In implementation of distributed firewall technology, policy languages are translated into some internal format by a compiler. This policy file is distributed to all the protected hosts by the system management software. A mechanism applies the security policy to incoming packets or connections. And incoming packet accepted or rejected by each host according to policy and the cryptographically verified identity of each sender (Ioannidis, et al. 2000).

Different variations can exist in implementation of distributed firewall technology. These variations are called hybrid firewall that is the combination of traditional firewall and distributed firewall. Some hybrid firewalls do not make use of cryptographic credentials and still rely on topological properties of the underlying network through inspection of the connecting nodes network address.

## 2.3. Firewall Policies

A firewall policy should describe how the firewall is managed. Before a firewall policy created, some of the analysis must be performed. The result of the analysis must identify network applications traffic which the firewall system handles and the details of which applications can traverse a firewall.

The steps for firewall policy creation are as follows (Wack, et al. 2002):

- Identification of network necessary applications,

- Identification of vulnerabilities associated with applications,
- Cost-benefits analysis of methods for securing the applications,
- Creation of firewall rule set based on applications traffic.

Most firewalls use rule sets for implementing security controls. The functionality of a firewall is determined by these rule sets. Nearly all rule sets, contain the following fields, as a minimum (Wack, et al. 2002):

- The source address of the packet, address of the computer system or device the network packet originated from (an IP address such as 192.168.1.1).
- The destination address of the packet, address of the computer system or device the network packet is trying to reach (e.g., 192.168.1.2).
- The type of traffic, the specific network protocol being used to communicate between the source and destination systems or devices.
- Possibly some characteristics of the Layer 4, the source and destination ports of the sessions (e.g., TCP:80 for the destination port belonging to a web server, TCP:1320 for the source port belonging to a personal computer accessing the server).
- Sometimes, information pertaining to which interface of the router the packet came from and which interface of the router the packet is destined for. Useful for routers with three or more network interfaces.
- An action, such as Deny or Permit the packet (Wack, et al. 2002)

Firewall rule set should be kept as simple as possible and should be building to be as specific as possible. Unless the traffic type and connections have been specially permitted, inbound traffic should be blocked by default policy. Common format of a rule in a firewall policy is as follows:

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

The *order* field determines the position of rule in rule set; *protocol* field specifies transport protocol of the packet. The IP addresses of the source and destination are specified respectively by *src_ip* and *dst_ip* fields. The port addresses of the source and destination of the packet are specified by *src_port* and *dst_port* fields (Al-Shaer and Hamed 2002).

### 2.3.1. Firewall Policy Representation

Policy representation modeling is important for implementing management techniques and visualizing the firewall policy structure. Al-Shaer and Hamed (2002) described a policy representation model in their research “Design and Implementation of Firewall Policy Advisor Tools”. They represented the firewall policy by a single rooted tree which is called the policy tree. Fields of the rules are represented as nodes in policy tree. And a possible value of the associated field is represented as a branch of this node. The protocol field is represented as a root node of a policy tree, and action field is represented as a leaf node. Every tree path starting at the root and ending at a leaf represents a rule in the policy. Rules that have the same field value at a specific node, will share the same branch representing that value. A firewall policy example is illustrated in Figure 2.3 and policy tree model for this policy example is illustrated in Figure 2.4.

| | | Source | | Destination | | |
|---|---|---|---|---|---|---|
| | Protocol | Address | Port | Address | Port | Action |
| 1: | tcp, | 140.192.37.20, | any, | *.*.*.*, | 80, | deny |
| 2: | tcp, | 140.192.37.*, | any, | *.*.*.*, | 80, | accept |
| 3: | tcp, | *.*.*.*, | any, | 161.120.33.40, | 80, | accept |
| 4: | tcp, | 140.192.37.*, | any, | 161.120.33.40, | 80, | deny |
| 5: | tcp, | 140.192.37.30, | any, | *.*.*.*, | 21, | deny |
| 6: | tcp, | 140.192.37.*, | any, | *.*.*.*, | 21, | accept |
| 7: | tcp, | 140.192.37.*, | any, | 161.120.33.40, | 21, | accept |
| 8: | tcp, | *.*.*.*, | any, | *.*.*.*, | any, | deny |
| 9: | udp, | 140.192.37.*, | any, | 161.120.33.40, | 53, | accept |
| 10: | udp, | *.*.*.*, | any, | 161.120.33.40, | 53, | accept |
| 11: | udp, | 140.192.38.*, | any, | 161.120.35.*, | any, | accept |
| 12: | udp, | *.*.*.*, | any, | *.*.*.*, | any, | deny |

Figure 2.3. A Firewall Policy Example
(Source: Al-Shaer and Hamed 2002)

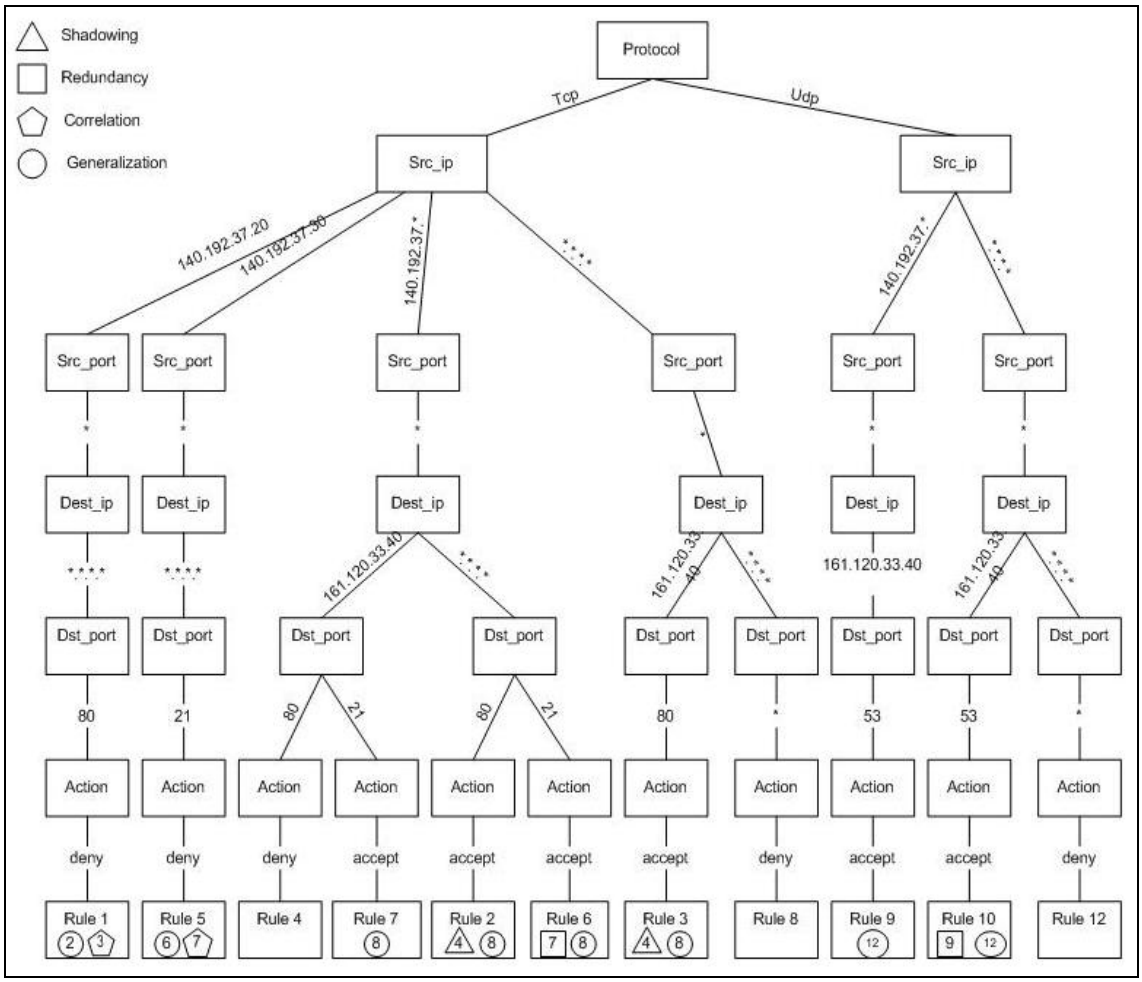



Figure 2.4. Policy Tree for the Firewall Policy in Figure 2.3
(Source: Al-Shaer and Hamed 2002)

Rule 1 to 9 all have the same protocol field value “tcp” and they share the same branch at protocol node. Rules 1 and 5 have different src_ip field value while rules 2, 4, 6 and 7 share the same src_ip field value. Similarly, rules 3 and 8 share the same src_ip branch. Rules 9, 10 and 12 share the same branch at protocol node with field value “udp”. Notice that rules 8 and 4 appear on rules 2 and 3 while they have separate branches also, because they have superset relations with other rules. Relations between filtering rules will be introduced later in this chapter.

## 2.4. Policy Anomalies

Correct operation of the firewall is dependent on the filtering rules. Therefore, administrator must give special attention to proper rule placement in filtering rule set.

In order to analyze the firewall policy modeling of firewall rule relations is necessary. The model which is created by Al-Shaer and Hamed (2002) and introduced in their research "Design and Implementation of Firewall Policy Advisor Tools" is used to analyze firewall policies. They defined all possible relations between filtering rules and they proved that there is no other relation exists. Definitions of relations are in Appendix A.

**Exactly matched:** Rules $R_x$ and $R_y$ are exactly matched if every field in $R_x$ is equal to the corresponding field in $R_y$ (Al-Shaer and Hamed 2002).

For the following rules, they are exactly matched because of all corresponding fields in both rules are equal.

1: tcp, 192.168.1.3, any, 193.140.248.*, 21, accept
2: tcp, 192.168.1.3, any, 193.140.248.*, 21, deny

**Inclusively matched:** Rules $R_x$ and $R_y$ are inclusively matched if they do not exactly match and if every field in $R_x$ is a subset or equal to the corresponding field in $R_y$ (Al-Shaer and Hamed 2002).

For the following rules, every field in rule 1 is subset or equal to corresponding field in rule 2 and every field in rule 2 is superset or equal to corresponding field in rule 1. In this relation rule 1 is called subset match, rule 2 is called superset match.

1: tcp, 192.168.1.3, any, 193.140.248.*, 21, accept
2: tcp, 192.168.1.*, any, 193.140.248.*, any, deny

**Completely disjoint:** Rules $R_x$ and $R_y$ are completely disjoint if every field in $R_x$ is not a subset and not a superset and not equal to the corresponding field in $R_y$ (Al-Shaer and Hamed 2002).

For the following rules, they are completely disjoint. Every corresponding fields in both rules are not subset and not superset and not equal.

1: tcp, 192.168.1.3, 2000, 10.10.1.4, 21, accept
2: tcp, 192.168.1.5, 3000, 10.10.1.11, 80, deny

**Partially disjoint:** Rules $R_x$ and $R_y$ are partially disjoint (or partially matched) if there is at least one field in $R_x$ that is a subset or a superset or equal to the corresponding field in $R_y$, and there is at least one field in $R_x$ that is not a subset and not a superset and not equal to the corresponding field in $R_y$ (Al-Shaer and Hamed 2002).

For the following rules, all fields except destination port field in rule 1 are subset or equal to corresponding fields rule 2.

1: tcp, 192.168.1.3, any, *.*.*.*, 21, accept
2: tcp, 192.168.1.*, any, *.*.*.*, 80, deny

**Correlated:** Rules $R_x$ and $R_y$ are correlated if some fields in $R_x$ that are subset or equal to the corresponding field in $R_y$, and the rest of fields in Rx are supersets of the corresponding fields in $R_y$ (Al-Shaer and Hamed 2002).

For the following rules, below protocol, source port and destination port fields in rule 1 are equals to corresponding fields in rule 2 src_ip field in rule1 is subset of src_ip field in rule 2 and dst_ip field in rule 1 is superset of dst_ip field in rule 2.

1: tcp, 192.168.1.3, any, *.*.*.*, 80, accept
2: tcp, *.*.*.*, any, 192.168.1.*, 80, deny

### 2.4.1. Intra-Firewall Policy Anomalies

"An intra-firewall policy anomaly is defined as the existence of two or more filtering rules that may match the same packet" (Al-Shaer, et al. 2005). Anomaly types which are classified in Appendix B as follows:

**Shadowing anomaly:** if all the fields in a rule match all the fields in a previous rule, this rule will never be worked through and shadowing anomaly will be occurred. If the shadowing rule is removed the security policy will not be affected. (Al-Shaer, et al. 2005)

Example rules for cases (1) (2) are shown below:

Case (1)

1: tcp, 140.192.37.*, any, 192.168.1.12, 80, accept

2: tcp, 140.192.37.*, any, 192.168.1.12, 80, deny

Case (2)

1: tcp, *.*.*.*, any, 192.168.1.12, 80, accept

2: tcp, 140.192.37.*, any, 192.168.1.12, 80, deny

Rule 2 is shadowed by rule 1 in the examples above.

**Correlation anomaly:** if some fields in first rule match corresponding fields in second rule and some fields in second rule match corresponding rules in first rule and if these rules have different action fields these rules have correlation anomaly. (Al-Shaer, et al. 2005).

Example rule for case (3) is shown below:

Case (3)

1: tcp, 192.168.1.20, any, *.*.*.*, 80, deny

2: tcp, *.*.*.*, any, 192.168.33.40, 80, accept

Rule 1 is in correlation with rule 2 in example above.

**Generalization anomaly:** if fields in second rule match all corresponding fields in first rule and they have different action, second rule is generalization of first rule (Al-Shaer, et al. 2005).

Example rule for case (4) is shown below:

Case (4)

1: tcp, 192.168.1.20, any, *.*.*.*, 80, deny

2: tcp, 192.168.1.*, any, *.*.*.*, 80, accept

Rule 2 is a generalization of rule 1 in example above.

**Redundancy anomaly:** a redundant rule performs same action on the same fields as another rule. If the redundant rule is removed policy will not be affected (Al-Shaer, et al. 2005).

Example rules for cases (5), (6) are shown below:

Case (5)
1: tcp, 192.168.37.*, any, 140.192.33.40, 21, accept
2: tcp, 192.168.37.*, any, 140.192.33.40, 21, accept

Case (6)
1: tcp, 192.168.37.*, any, *.*.*.*, 21, accept
2: tcp, 192.168.37.*, any, 140.192.33.40, 21, accept

Rule 2 is redundant to rule 1 in examples above.

## 2.4.2. Inter-Firewall Policy Anomalies

In distributed firewall environment, if any two firewalls on a network path take different filtering actions, an inter-firewall anomaly may exist. For example a firewall might block a traffic that is permitted by another firewall on the same network path.

Al-Shaer, Hamed, Boutaba and Hasan classified anomalies in multi-firewall environments (Al-Shaer, et al. 2005). And they used network model that is illustrated in Figure 2.5.

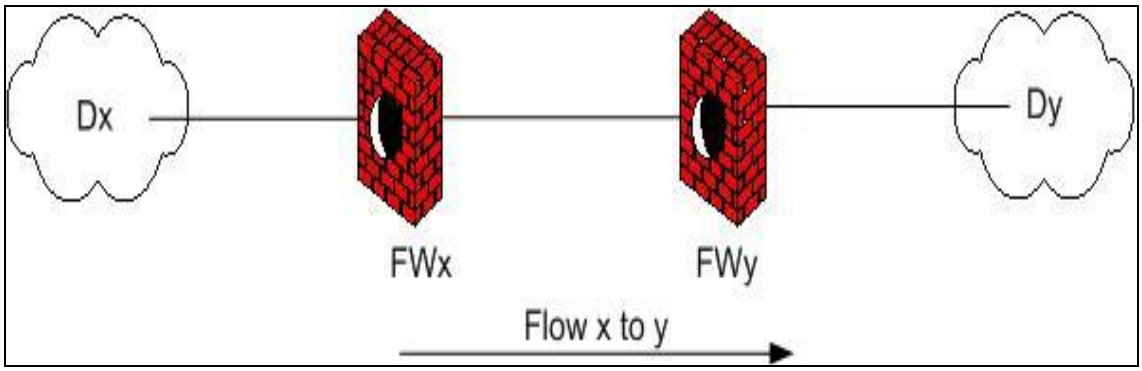



Figure 2.5. Cascaded Firewalls Isolating Domains $D_x$ and $D_y$.

As it is shown Figure2.5 traffic stream is flowing from subdomain $D_x$ to subdomain $D_y$ . According to direction of flow, the closest firewall to the flow source subdomain is called upstream firewall ($FW_x$) while the closest firewall to the flow destination subdomain is called downstream firewall ($FW_y$).

**Shadowing anomaly:** if an upstream firewall blocks the network traffic accepted by a downstream shadowing anomaly occurs. $R_d$ belongs to downstream firewall's policy while $R_u$ belongs to upstream firewall's policy (Al-Shaer, et al. 2005).

Example rules for cases (1), (2), (3), (4) are shown below:

Case (1)
$R_u$: tcp, 192.168.*.*, any, 140.192.22.5, 21, deny
$R_d$: tcp, 192.168.*.*, any, 140.192.22.5, 21, accept

Case (2)
$R_u$: tcp, *.*.*.*, any, *.*.*.*, any, deny
$R_d$: tcp, 192.168.*.*, any, 140.192.22.5, 21, accept

Case (3)
$R_u$: tcp, 192.168.*.*, any, 140.192.22.5, 25, deny
$R_d$: tcp, 192.168.*.*, any, 140.192.*.*, 21, accept

Case (4)
$R_u$: tcp, 192.168.1.*, any, 140.192.*.*, 23, accept
$R_d$: tcp, 192.168.*.*, any, 140.192.*.*, 23, accept

**Spuriousness anomaly:** if an upstream firewall permits the network traffic denied by a downstream, a spuriousness anomaly occurs (Al-Shaer, et al. 2005).

Example rules for cases (5), (6), (7), (8), (9) are shown below:

Case (5)
$R_u$: tcp, 140.192.*.*, any, 192.168.*.*, 80, accept
$R_d$: tcp, 140.192.*.*, any, 192.168.*.*, 80, deny

Case (6)

$R_u$: tcp, 140.192.*.*, any, 192.168.*.*, 80, accept

$R_d$: tcp, *.*.*.*, any, *.*.*.*, any, deny

Case (7)

$R_u$: tcp, 192.168.1.*, any, 140.192.*.*, 23, accept

$R_d$: tcp, 192.168.1.*, any, 140.192.37.*, 23, accept

Case (8)

$R_u$: tcp, 192.168.*.*, any, 140.192.*.*, 21, accept

$R_d$: tcp, 192.168.*.*, any, 140.192.22.5, 21, accept

Case (9)

$R_u$: tcp, 192.168.1.*, any, 140.192.37.1, 23, deny

$R_d$: tcp, 192.168.1.*, any, 140.192.37.*, 23, deny

**Redundancy anomaly:** if a down stream firewall denies the network traffic already blocked by upstream firewall a redundancy anomaly occurs (Al-Shaer, et al. 2005).

Example rules for cases (10), (11) are shown below:

Case (10)

$R_u$: tcp, 192.168.1.*, any, 140.192.37.1, 80, deny

$R_d$: tcp, 192.168.1.*, any, 140.192.37.1, 80, deny

Case (11)

$R_u$: tcp, *.*.*.*, any, *.*.*.*, any, deny

$R_d$: tcp, 192.168.1.*, any, 140.192.37.1, 80, deny

**Correlation anomaly:** as a result of having two correlated rules in upstream and downstream firewall a correlation anomaly occurs (Al-Shaer, et al. 2005).

Case (12)

$R_u$: tcp, 140.192.*.*, any, 192.168.1.*, 80, accept

$R_d$: tcp, 140.192.37.1, any, 192.168.*.*, 80, accept

In this example, only the traffic coming from 140.192.37.1 and destined to 192.168.1.* will be accepted and other traffic destined to 192.168.*.* will be shadowed at the upstream firewall

Case (13)

$R_u$: tcp, 140.192.*.*, any, 192.168.1.*, 80, deny

$R_d$: tcp, 140.192.37.1, any, 192.168.*.*, 80, deny

In this case, the traffic coming from 140.192.37.1 and destined to 192.168.1.* will deny at the downstream firewall.

Case (14)

$R_u$: tcp, 140.192.*.*, any, 192.168.1.*, 80, accept

$R_d$: tcp, 140.192.37.1, any, 192.168.*.*, 80, deny

In this example upstream firewall permits the traffic which is coming from 140.192.37.1 while same traffic is denied at the downstream firewall

Case (15)

$R_u$: tcp, 140.192.*.*, any, 192.168.1.*, 80, deny

$R_d$: tcp, 140.192.37.1, any, 192.168.*.*, 80, accept

In this example, upstream firewall blocks the traffic which is coming from 140.192.37.1 and destined to 192.168.1.* while same traffic is accepted at the downstream firewall.

# CHAPTER 3

# DESIGN AND IMPLEMENTATION OF POLICY ANOMALY CHECKER

Firewall functionality depends on the filtering rules and their order. If there is an anomaly among rules, correct and efficient operation can not be achieved. In this chapter, design and implementation of Policy Anomaly Checker is explained.

## 3.1. Requirements of Policy Anomaly Checker

Policy Anomaly Checker application is developed to discover firewall policy anomalies. The main requirement is to define anomalies among policy rules by using firewall rules relations and policy representation model which is introduced by Al-Shaer and Hamed (2002).

### 3.1.1. Functional Requirements of Policy Anomaly Checker

The functional requirements of Policy Anomaly Checker are based around three main use cases which are Intra-firewall anomaly checking, inter-firewall anomaly checking and rule generation.

#### 3.1.1.1. Intra-Firewall Anomaly Checking

In this requirement rules in the selected policy are checked in order to discover anomaly. Policy can be selected from either network topology that is already defined or file system.

Input: Selected policy.
Process: Check intra-firewall anomaly.
Output: Results of the intra-firewall anomaly checking process.

### 3.1.1.2. Inter-Firewall Anomaly Checking

Rules in all policies are checked in order to discover anomaly. In order to achieve this requirement all network paths between subdomains must be already defined in the system.

Input: Network paths between subdomains, firewall policies that are deployed on these paths.
Process: Check inter-firewall anomaly.
Output: Results of the inter-firewall anomaly checking process.

### 3.1.1.3. Rule Generation

Beside anomaly checking property, rule generation functionality is added to Policy Anomaly Checker. Rules are generated with or without anomaly according to user selection.

Input: Number of rules that will be generated, with anomaly or without anomaly selection
Process: Generate firewall rules
Output: Generated rules with policy tree representation.

## 3.2. Design Principles

Object oriented development is a development strategy where analysis, design and programming process are executed to create and apply an object model. The concepts of application domain are implemented as objects and their interactions in the model. It is aimed to provide software reusability, extensibility, reliability and portability using such a development strategy.

Design patterns strategies are applied during object oriented development process. Design patterns can be considered as pre-defined, repeatable and non-finished solutions. Similar problems can occur during different software development process. Instead of generating new solutions each time it is easier to use these pre-defined and reliable solutions which also generate a common language between developers.

Design patterns can be classified as creational, behavioral and structural patterns depending on the types of problems they are used to solve. During software development process of this thesis Singleton Pattern is applied. It is a creational pattern and used to restrict instantiation of a class to one object in the system (Cooper 2000).

## 3.3. Architectural Layers of Policy Anomaly Checker

Architectural layers of application are illustrated in Figure 3.1. The architecture consists of three layers which separates application functionality into three distinct abstractions.

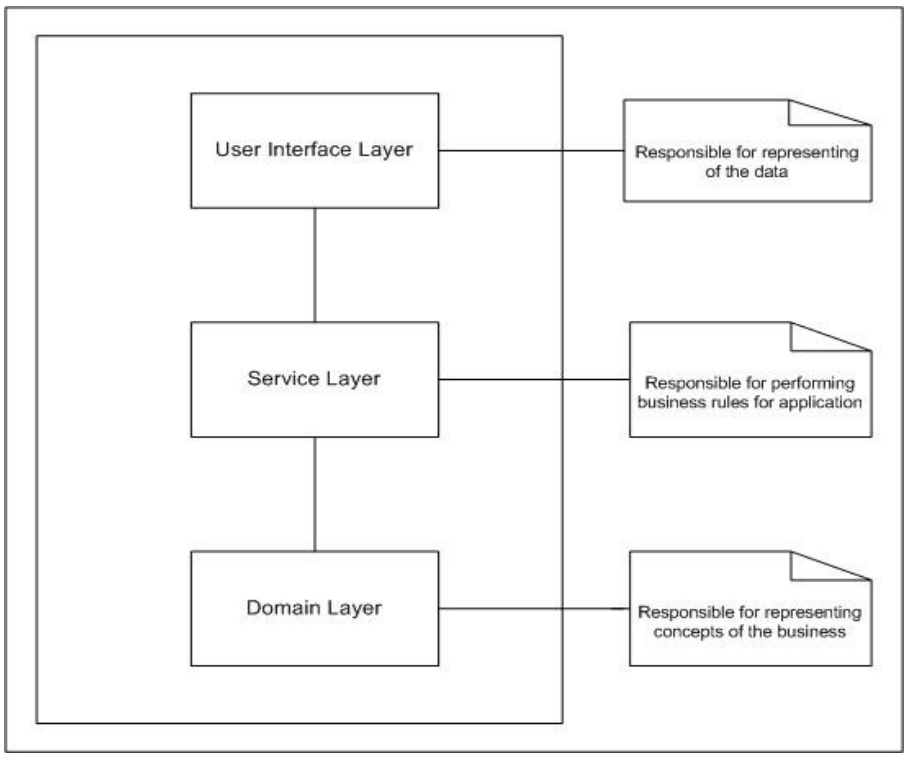



Figure 3.1. Architectural Layers of Policy Anomaly Checker

### 3.3.1. Domain Layer of Policy Anomaly Checker

Domain layer of the application is designed for the representation of business concepts (Figure 3.2). A policy consists of rules and their tree representation in model. Policy also contains its anomaly checker. Each rule has order, protocol, src_ip, dest_ip, src_port, dest_port and action fields. An interface class which is called *Icategorization* provides methods for building policy tree by using fields of the rules in a policy. An

abstract class which is called *AbstractField* provides methods for definition of relation between rules.

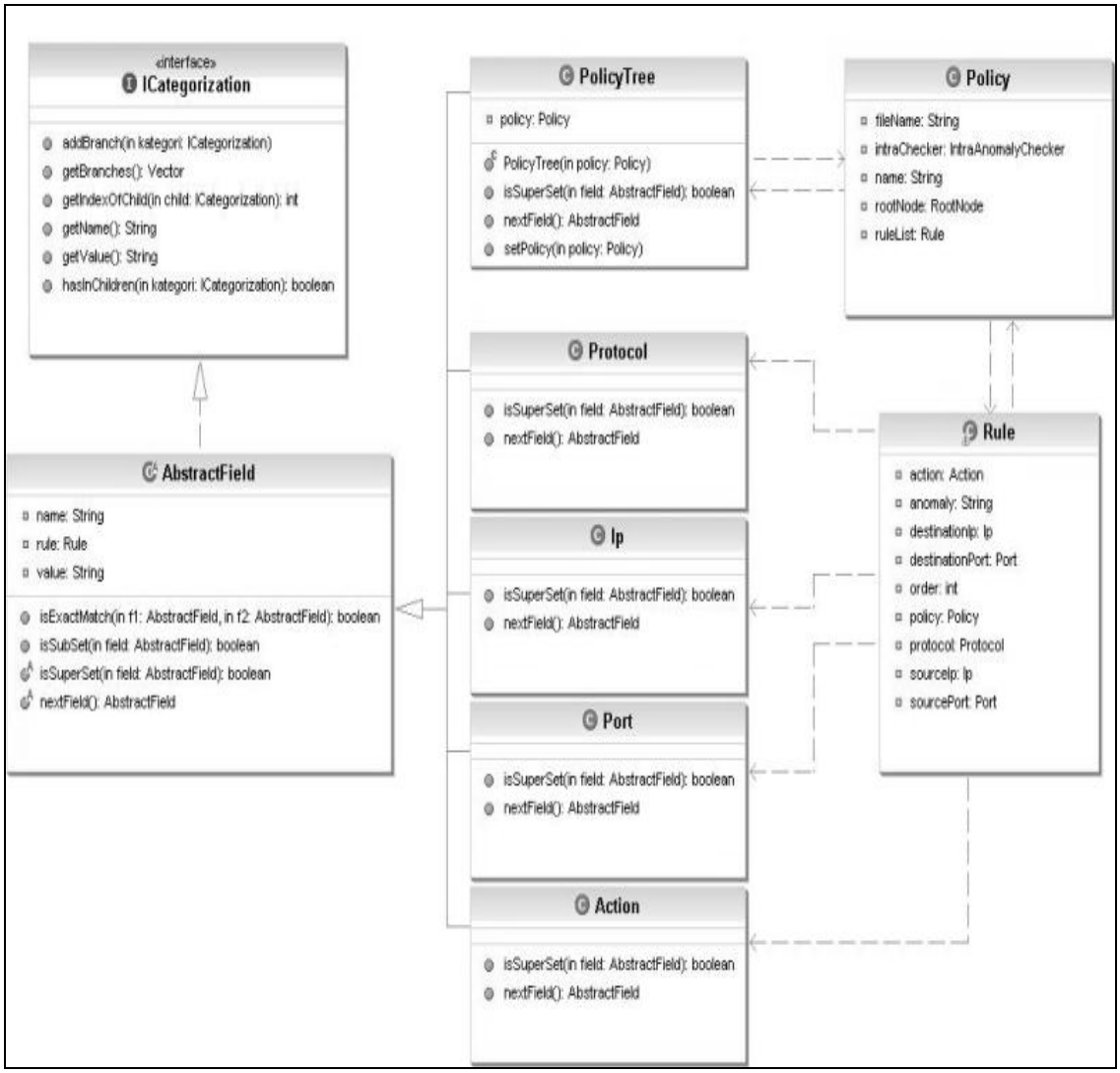



Figure 3.2. Domain Layer of Policy Anomaly Checker

## 3.3.2. Service Layer of Policy Anomaly Checker

Service layer of the application is designed for the performing business rules. (Figure 3.3) An abstract class which is called *AnomalyChecker* provides methods for inter and intra-firewall anomaly checking. *IntraAnomalyChecker* and *InterAnomalyChecker* inherit *anomalyTransition* method from abstract class.

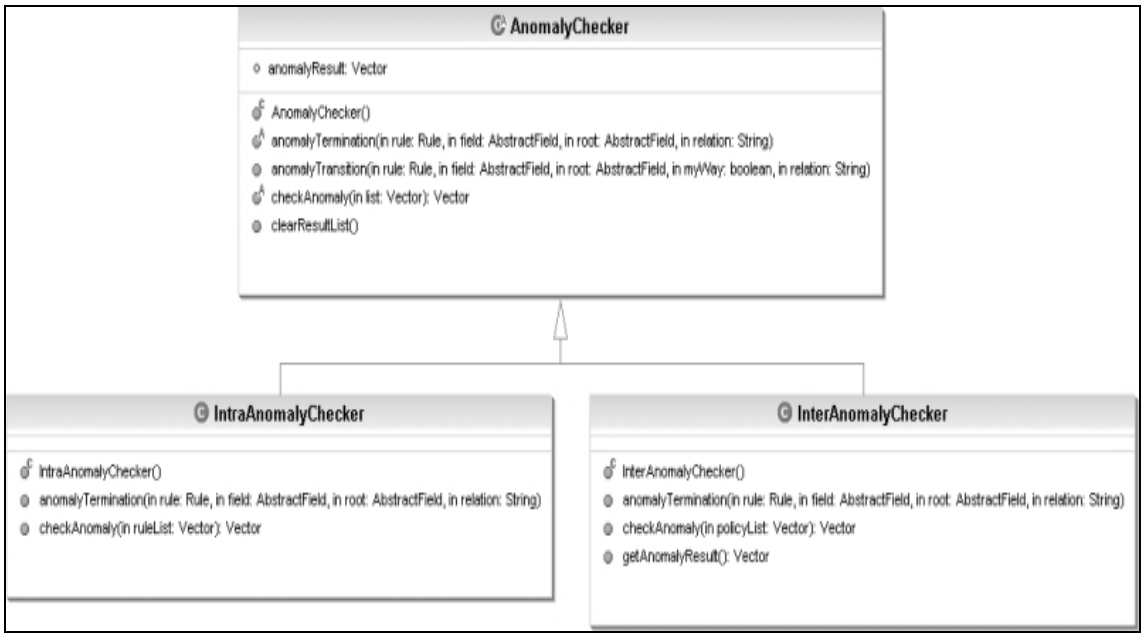



Figure 3.3. Service Layer of Policy Anomaly Checker

### 3.3.3. User Interface Layer of Policy Anomaly Checker

User interface layer is designed for the presentation of policy and network topology trees (Figure 3.4). There are two singleton classes which are called *PolicyTreeFormController* and *TopologyController*. *TopologyController* class provides method for creation of the policies using network paths which are defined from the network topology. In order to discover inter-firewall anomalies, policies must be defined in system. *PolicyTreeFormController* class provides methods for presentation of policy and topology trees in the user interface layer. *ReportForm* class is used to show the results of the anomaly checking process to user.

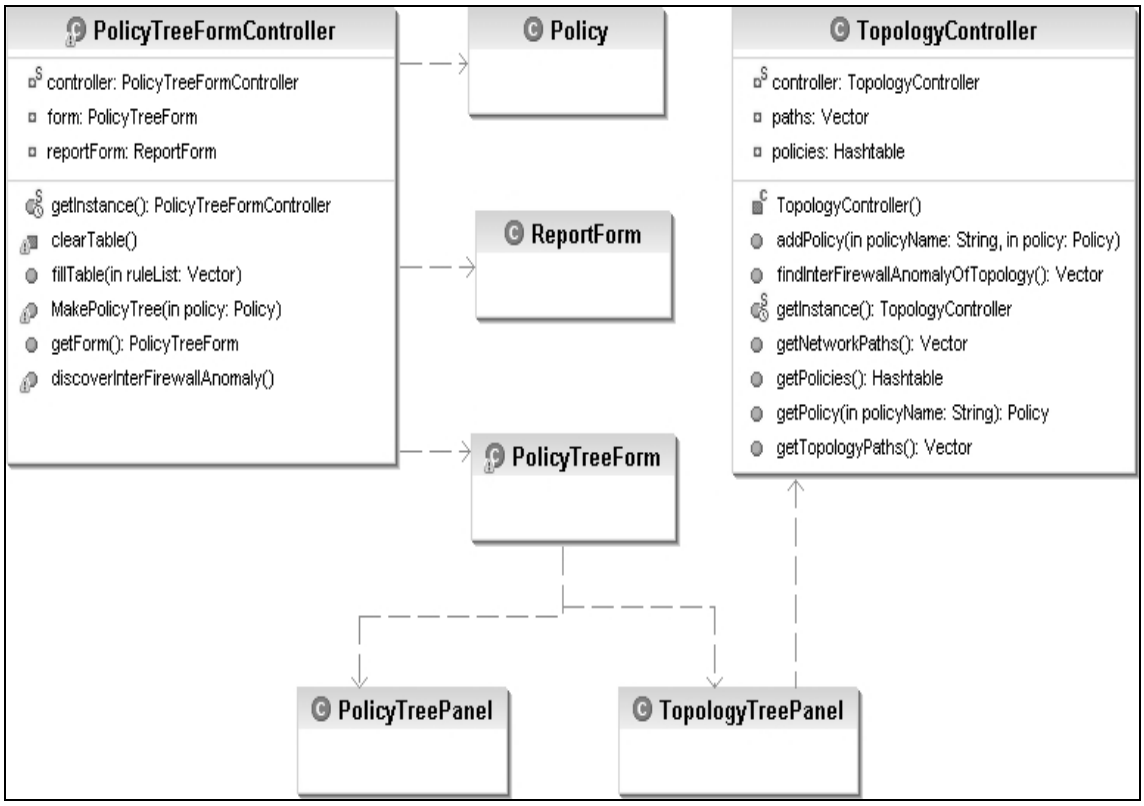



Figure 3.4. User Interface Layer of Policy Anomaly Checker

## 3.5. Algorithms

Policy anomalies can be determined based on anomaly definitions in chapter 2. In order to discover policy anomalies for intra and inter-firewall environment, two algorithms that are introduced by Al-Shaer and Hamed (2004) are implemented.

### 3.5.1. Intra-Firewall Anomaly Discovery Algorithm

In tree representation of policy, if any two rules share the same tree path explained in 2.3.1, there is a potential anomaly among these rules. The basic idea of intra-firewall anomaly discovery algorithm is to determine anomalies while building the policy tree. This algorithm is shown in Figure 3.5.

```
function IntraAnomalyDiscovery(rule_list)
  root = new node
  for each rule in rule_list do
    AnomalyTransition(rule, PROTOCOL, root, UNDETERMINED)
  return root
end function
```

Figure 3.5. Intra-Firewall Anomaly Discovery Algorithm
(Source: Al-Shaer and Hamed 2004)

Intra-firewall anomaly discovery algorithm invokes main function: an anomaly transition function which represents the transition states of fields in the state diagram (Figure 3.6).

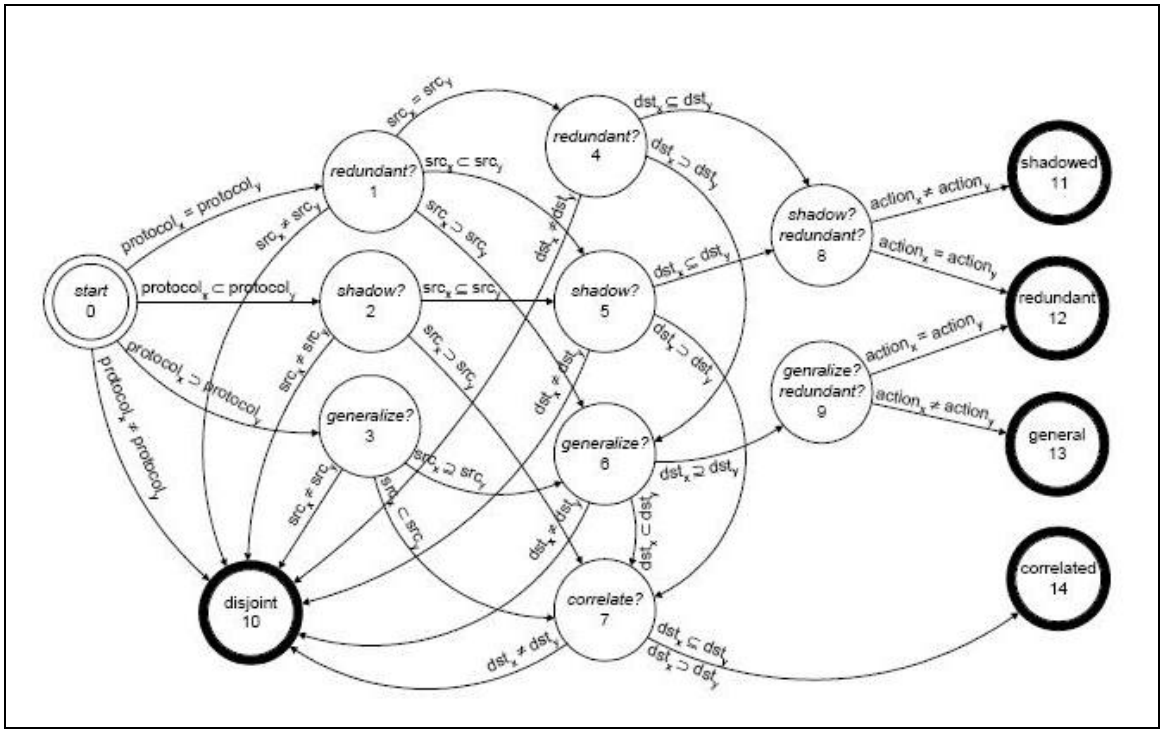



Figure 3.6. State Diagram for Detecting Intra-firewall Anomalies for Rules $R_x$ and $R_y$
(Source: Al-Shaer and Hamed 2004)

Anomaly discovery states for any two rules $R_x$ and $R_y$ are illustrated in Figure 3.6. $R_x$ and $R_y$ are in the same firewall and $R_x$ comes after $R_y$ in policy rule set. Source and destination fields for ips and ports are represented as a one field in diagram for simplification. State transition starts with protocol field. Each field in $R_x$ is compared to

the corresponding field in $R_y$. Result of subsequent comparison determines the relation between rules. If every field of $R_x$ is a subset or equal to the corresponding field in $R_y$ and both rules have the same action, $R_x$ is redundant to $R_y$, while if the actions are different, $R_x$ is shadowed by $R_y$. If every field of $R_x$ is a superset or equal to the corresponding field in $R_y$ and both rules have the same action, $R_x$ is potentially redundant to $R_y$, while if the actions are different, $R_y$ is a generalization of $R_x$. If some fields of $R_x$ are subsets or equal to the corresponding fields in $R_y$, and some fields of $R_x$ are supersets to the corresponding fields in $R_y$, and their actions are different, then $R_x$ is in correlation with $R_y$ (Al-Shaer and Hamed 2004).

AnomalyTransition function is invoked for every rule with an UNDETERMINED relation. This algorithm is modified in order to create separate path for rule which has anomaly. Since the algorithm does not provide separate rule placement for rules which also appears on other rule branches. Modified algorithm is illustrated in Figure 3.7.

If the field of the rule that is added to policy tree matches an already existing field in the tree then next relation state is determined base on the shown state diagram in Figure 3.5.

The algorithm is executed recursively to check the remaining fields. Relation state is updated until the final field is reached. If there is no determined relation, new branch is created at the current node.

```
function AnomalyTransition(rule, field,root, myway,relation)
        if field = ACTION then
            value_found = FALSE
            for each branch in node.branch_list do
                if branch.value = rule.field.value then
                    value_found = TRUE
                    if relation = UNDETERMINED then relation = EXACT
                        AnomalyTransition(rule, field.next, branch.node, myway,relation)
                else if rule.field.value is superset of branch.value then
                    if relation is in {SUBSET, CORRELATED} then
                        AnomalyTransition(rule, field.next, branch.node,false, CORRELATED)
                    else
                        AnomalyTransition(rule, field.next, branch.node, false,SUPERSET)
                else if rule.field.value is subset of branch.value then
                    if relation is in {SUPERSET, CORRELATED} then
                        AnomalyTransition(rule, field.next, branch.node, false,CORRELATED)
                    else
                        AnomalyTransition(rule, field.next, branch.node, false,SUBSET)
                    end if
            end for
        end if
        if value_found = FALSE
            if field = ACTION then
                AnomalyTermination(rule,field,root,relation);
            if myway=true then
                root.addBranch(field);
                anomalyTransition(rule, field.nextField(), field, true,DISJOINT);
            end if
        end if
end function
```

Figure 3.7. Modified Anomaly Transition Algorithm

When all the fields of rule have been inserted in the tree, IntraAnomalyTermination function is executed. This algorithm is modified in order to add action field to the tree. Since the algorithm does not provide addition of action field. Modified algorithm is illustrated in Figure 3.8.

```
function IntraAnomalyTermination(rule, field, root, relation)
    anomaly = NOANOMALY
    if root has branch_list then
    branch = root.branch_list.first()
        if relation = CORRELATED then
            if not rule.action = branch.value then
              anomaly = CORRELATION
              branch.rule.anomaly = CORRELATION
              report rule rule.id is in correlation with rule branch.rule.id
              root.addbranch(field)
            end if
        else if relation = SUPERSET and not rule.action = branch.value then
            anomaly = GENERALIZATION
            branch.rule.anomaly = SPECIALIZATION
            report rule rule.id is a generalization of rule branch.rule.id
            root.addbranch(field)
        else if relation = SUPERSET and rule.action = branch.value then
            if branch.rule.anomaly = NOANOMALY then
              branch.rule.anomaly = REDUNDANCY
              report rule branch.rule.id is redundant to rule rule.id
                root.addbranch(field)
            end if
        else if rule.action = branch.value then
            anomaly = REDUNDANCY
            report rule rule.id is redundant to rule branch.rule.id
            root.addbranch(field)
      else if not rule.action = branch.value then
          anomaly = SHADOWING
          report rule rule.id is shadowed by rule branch.rule.id
          root.addbranch(field)
      end if
    end if
    rule.anomaly = anomaly
end function
```

Figure 3.8. Modified Anomaly Termination Algorithm

In IntraAnomalyTermination function, if the rule action coincides with the action of another rule on the tree, an anomaly is discovered and rule is inserted in the rule's node. At that point the final anomaly state is determined and any anomalies are reported.

### 3.5.2. Inter-Firewall Anomaly Discovery Algorithm

Inter-firewall anomaly discovery algorithm (Figure 3.9) is implemented in order to find relations and discover anomalies between rules in two or more connected firewalls.

```
function InterAnomalyDiscovery(path_list)
    for each path in path_list do
      IntraAnomalyDiscovery(path.firewalls[1].rules)
      root = BuildPolicyTree(path.firewalls[1].rules)
      for each firewall in path.firewalls except path.firewalls[1] do
        IntraAnomalyDiscovery(firewall.rules)
        AnomalyTransition(rule, PROTOCOL, root, UNDETERMINED)
      end for
    end for
end function
```

Figure 3.9. Inter-Firewall Anomaly Discovery Algorithm
(Source: Al-Shaer and Hamed 2004)

The inter-firewall anomaly discovery process should be performed on all firewalls in the path connecting any two sub-domains in the network. The all possible paths between subdomains in the network must be determined before the execution of the algorithm. For example in Figure 3.10 there are six distinct paths between subdomains. Traffic flow from D1.1 to Internet {FW1, FW0} is one path and {FW1, FW0, FW2} is another path for the traffic from D1.2 to D2.2.

Inter-firewall anomaly discovery algorithm takes a path list as parameter. For every firewall in the path firstly intra-firewall anomaly discovery algorithm (Figure 3.5) is executed to ensure that every individual firewall is free from intra-firewall anomalies. Next, for the most upstream firewall policy tree is built and the rules of all the consecutive firewalls in the path are added into this tree.

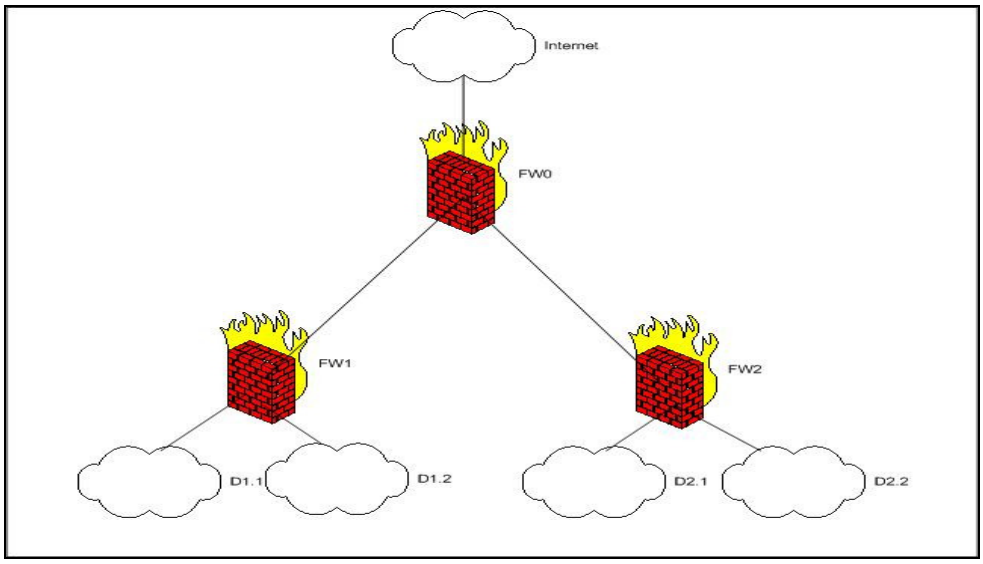



Figure 3.10. Example for a Distributed Firewall Environment

Same AnomalyTransiton function is used for both Inter-firewall anomaly discovery algorithm and intra-firewall anomaly discovery algorithm. However, different AnomalyTermination function is invoked for the inter-firewall anomaly discovery algorithm.

InterAnomalyTermination function determines the anomaly based on the discovered relation and the actions of the currently inserted rule with the existing rule in the upstream policy. This algorithm is also modified in order to add action field to the tree. Since the algorithm which is introduced by Al-Shaer and Hamed (2004) does not provide addition of action field. Modified algorithm is illustrated in Figure 3.11

```
function InterAnomalyTermination(rule, field, node, relation)
    anomaly = UNDETERMINED
    for each branch in node.branch_list do
        if rule.firewall = branch.rule.firewall then
            exit for
        else if relation = EXACT then
            if rule.action = accept and branch.rule.action = deny then
                anomaly = SHADOWING
                report rule rule.id is shadowed by rule branch.rule.id
                root.addbranch(field)
            else if rule.action = deny and branch.rule.action = accept then
                branch.rule.anomaly = SPURIOUSNESS
                report rule branch.rule.id is spurious to rule rule.id
                root.addbranch(field)
            else if rule.action = deny and branch.rule.action = deny then
                anomaly = REDUNDANCY
                report rule rule.id is potentially redundant to rule branch.rule.id
                root.addbranch(field)
            end if
        else if relation = SUBSET then
            if rule.action = accept and branch.rule.action = deny then
                anomaly = SHADOWING
                report rule rule.id is shadowed by rule branch.rule.id
                root.addbranch(field)
            else if branch.rule.action = accept
                branch.rule.anomaly = SPURIOUSNESS
                report rule branch.rule.id is partially spurious to rule rule.id
                root.addbranch(field)
            else if rule.action = deny and branch.rule.action = deny then
                anomaly = REDUNDANCY
                report rule rule.id is potentially redundant to rule branch.rule.id
                root.addbranch(field)
            end if
        else if relation = SUPERSET then
            if rule.action = deny then
                branch.rule.anomaly = SPURIOUSNESS
                report rule branch.rule.id is spurious to rule rule.id
                root.addbranch(field)
            else if rule.action = accept then
                anomaly = SHADOWING
                report rule rule.id is partially shadowed by rule branch.rule.id
                root.addbranch(field)
            end if
        else if relation = CORRELATED then
                anomaly = CORRELATION
                branch.rule.anomaly = CORRELATION
                report rule rule.id is in correlation with rule branch.rule.id
                root.addbranch(field)
        end if
    end for
    rule.anomaly = anomaly
end function
```

Figure 3.11. Modified InterAnomaly Termination Algorithm

## 3.6. Implementation of Policy Anomaly Checker

Application development environment and main scenario is explained in implementation of Policy Anomaly Checker. In addition, user interfaces of application are also described in this section.

### 3.6.1. Application Development Environment

Policy Anomaly Checker is developed on Java platform using Eclipse. Version of Java that is used in application development is JDK1.5.0.6

### 3.6.2. Scenarios and User Interfaces

Scenarios and user interface of Policy Anomaly Checker is described in this section.

#### 3.6.2.1. Scenarios

Policy Anomaly Checker has three main scenarios. First scenario is the intra-firewall anomaly checking that is illustrated in Figure 3.12.

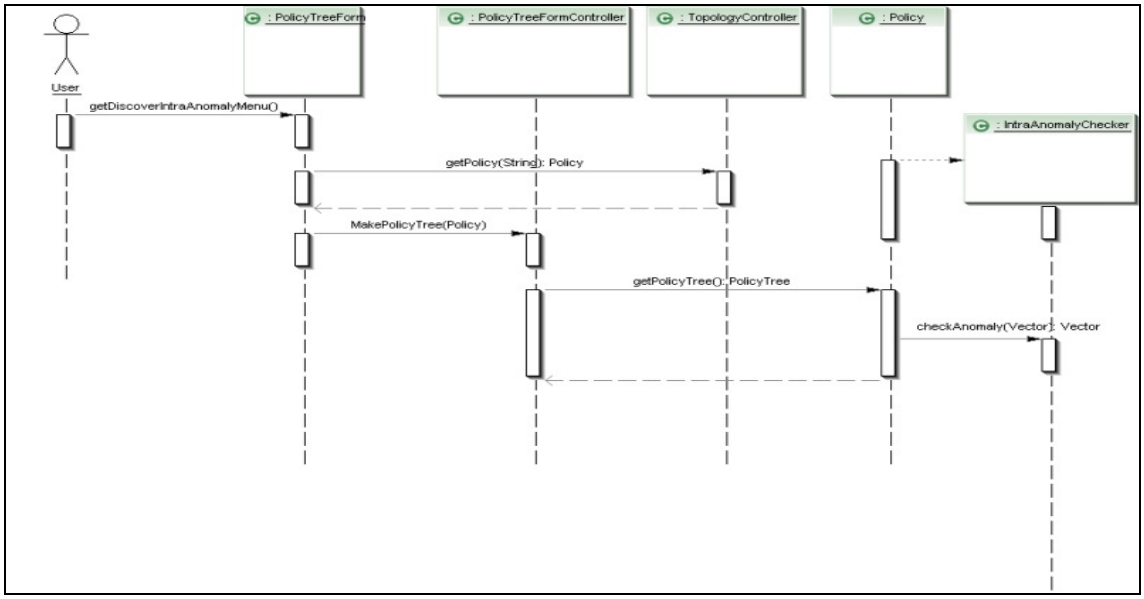



Figure 3.12. Sequence Diagram for Intra-firewall Anomaly Checking

When user selects a “Discover Intra-firewall Anomaly” menu item from user interface of the application, firstly related policy object is pulled from *TopologyController* class and its *PolicyTree* object is shown on the user interface’s tree. If *PolicyTree* of this object has not built yet, *IntraAnomalyChecker* object is used to built *PolicyTree* and get its anomaly results.

Second scenario is the inter-firewall anomaly checking which is illustrated in Figure 3.13. When user selects a “Discover Inter-firewall Anomaly” menu item from user interface of the application, firstly network paths which are defined in a properties file are pulled from *TopologyControl* class. For each path in the network paths, policies which are installed in this path are pulled from *TopologyContro*l class and *InterAnomalyChecker* checks the anomaly for these policies’s rule set. Anomaly results are shown on the report form.

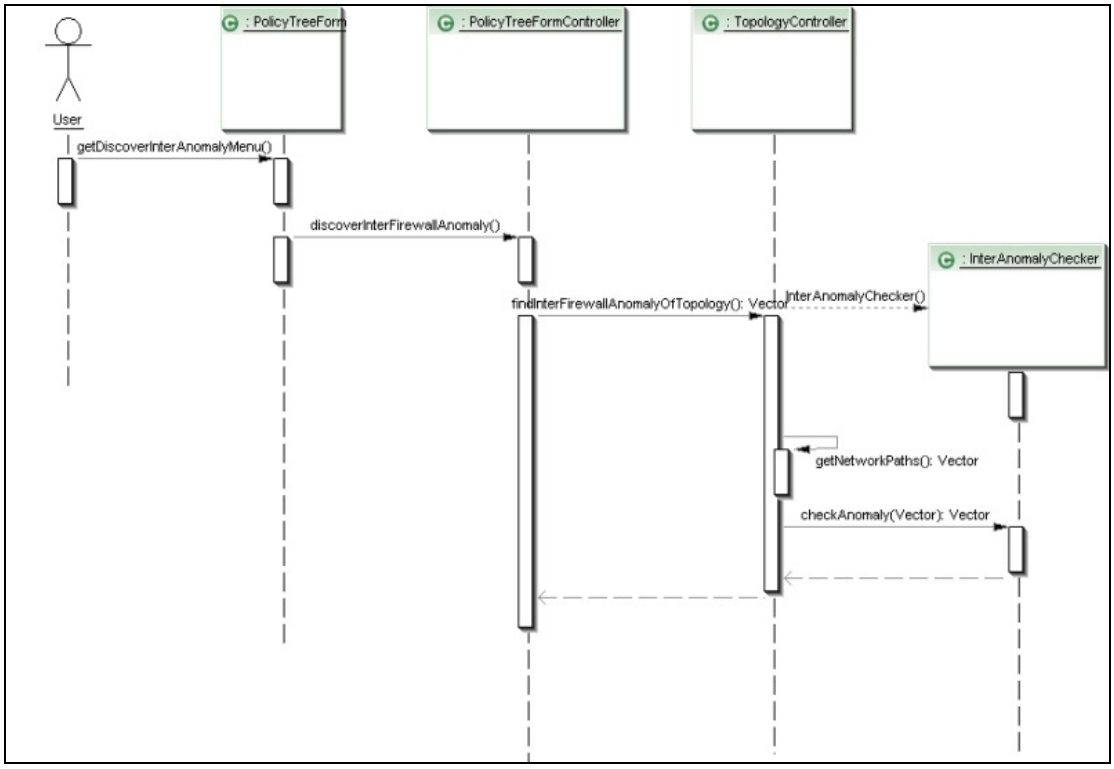



Figure 3.13. Sequence Diagram for Inter-firewall Anomaly Checking

Last scenario is the rule generation which is illustrated in Figure 3.14. In the rule generation scenario when user selects a “Generate Rules Without Anomaly” menu item from the user interface, firstly a service object which is called *RuleGenerator* is

created. Policy which is belonging to generated rules is created by *RuleGenerator* class. Generated rules are added to this Policy. In the rule addition process anomaly is checked between newly added rule and other rules in the policy. If there is an anomaly between rules, newly added rule is removed and new rule is generated. This process is repeated until an anomaly free rule is found. In this scenario, generated rules can be saved to selected files.

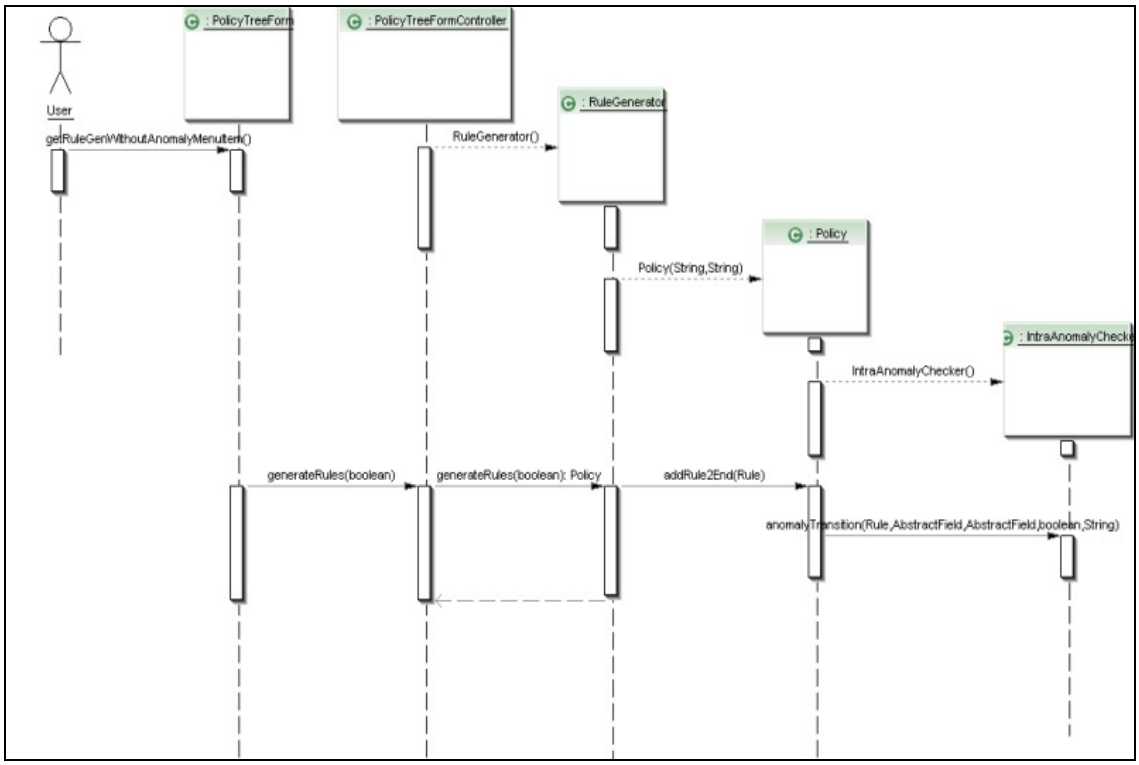



Figure 3.14. Sequence Diagram for Rule Generation without Anomaly

### 3.6.2.2. User Interfaces

User interface of Policy Anomaly Checker consists of three main panels which are called Topology Tree, Policy Tree and Rule List (Figure C.1). Topology tree is used to presentation of policies in the network topology. Policy tree is used to presentation of rules in the selected policy. On the Rule List Panel, rules are shown as a table for selected policy. Intra-firewall anomaly discovery is achieved by Anomaly Detection Menu (Figure C.2). It is achieved by Show Rules and Discover Anomaly Menu from TopologyTree (Figure C.3). As illustrated in Figure C.3 in order to execute Anomaly Discovery process, a policy must be selected from Topology Tree.

For intra-firewall anomaly checking and inter-firewall anomaly checking scenarios, result of the processes are shown on to Anomalies Form (Figure C.4). In order to generate random or anomaly free rules Rule Generator Menu is used (Figure C.5).

## 3.7. Test Cases

Execution of intra and inter-firewall anomaly discovery algorithm is tested for all the possible anomaly cases which are explained in Chapter 2. Necessary rules are generated and algorithms are executed with these rules for cases.

### 3.7.1. Intra-Anomaly Discovery Algorithm Test Cases

Intra-firewall anomaly algorithm is executed for six cases which are explained in section 2.4.1. Results for each case are shown in Figure 3.15.

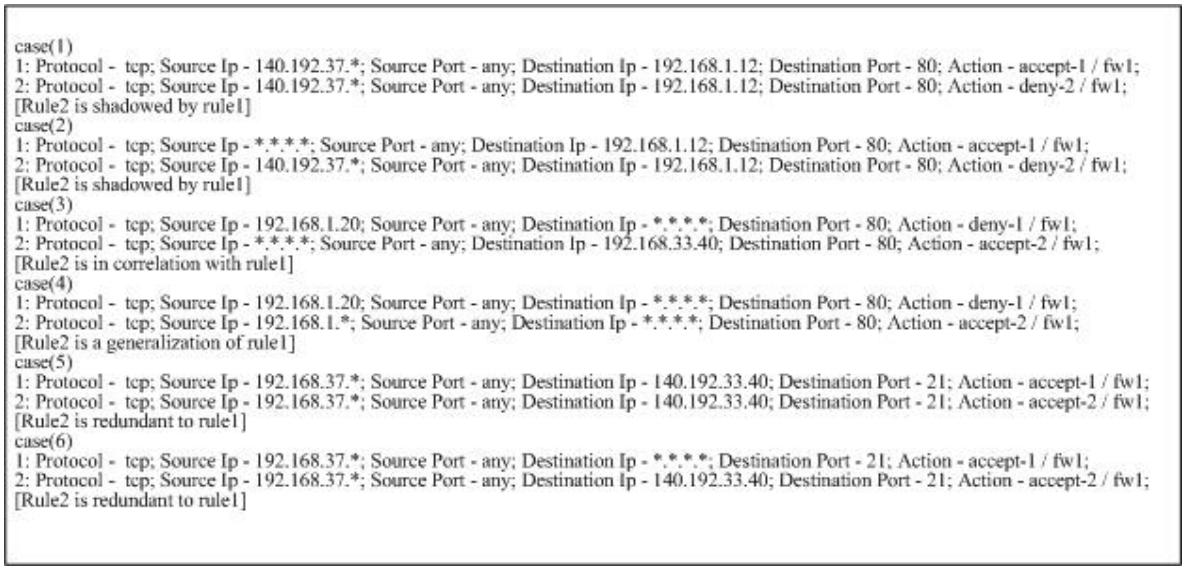



Figure 3.15. Intra-Anomaly Discovery Algorithm Results for Anomaly Cases

### 3.7.2. Inter-Anomaly Discovery Algorithm Test Cases

Inter-firewall anomaly algorithm is executed for fifteen cases which are explained in section 2.4.2. Network topology which is used in test has two firewalls; FW1 and FW2. FW1 is defined as upstream firewall; FW2 is defined as downstream firewall. Results for each case are shown in Figure 3.16.

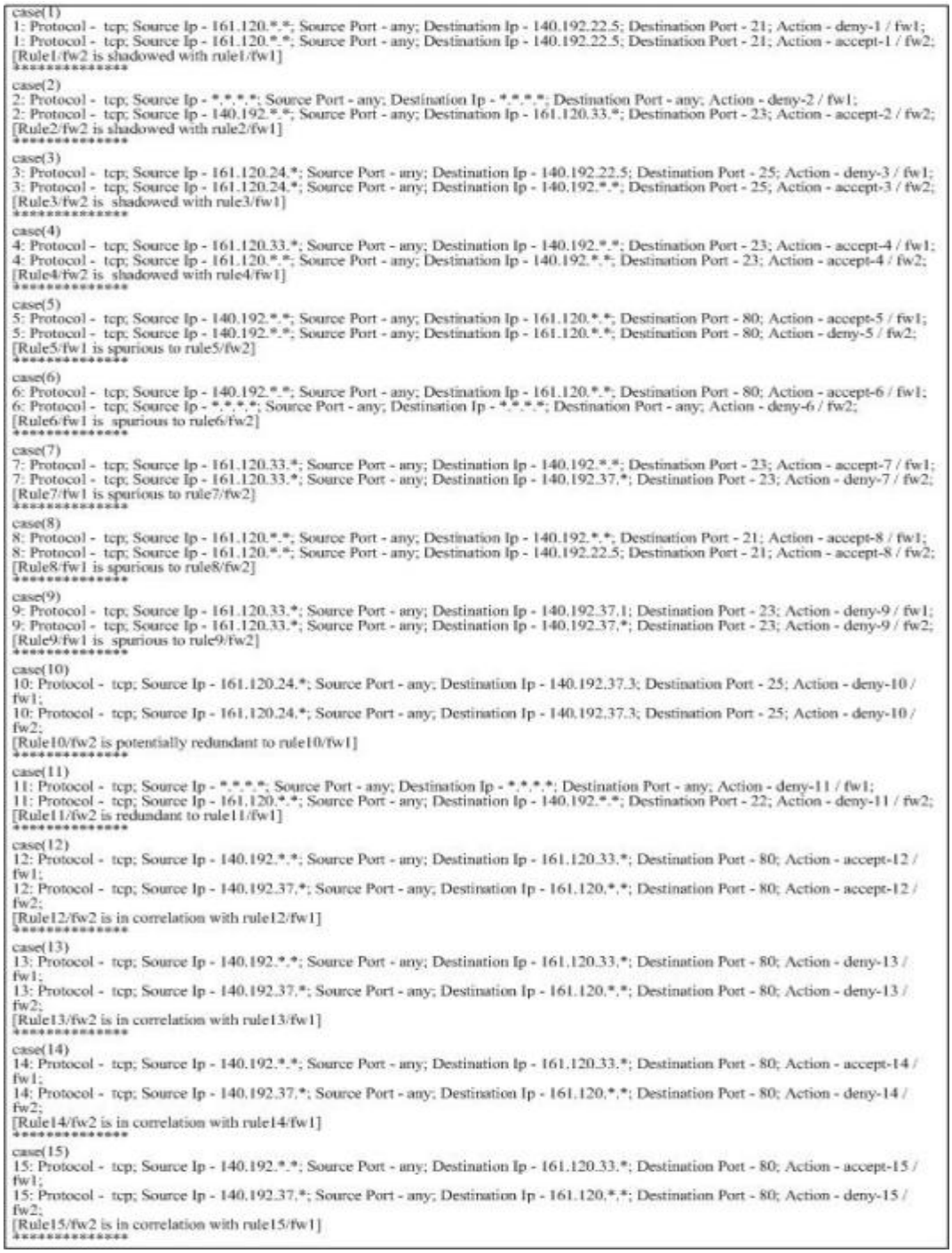



Figure 3.16. Inter-Anomaly Discovery Algorithm Results for Anomaly Cases

# CHAPTER 4

# EXPERIMENTS AND EVALUATION

In order to obtain operational values of intra and inter-firewall anomaly discovery algorithms, a number of tests are performed using different policies and network topologies. These tests are performed on a Pentium IV-M 1.73 GHz. processor with 1.49 GByte of RAM.

In order to obtain operational values of intra-firewall anomaly discovery algorithm, four sets of firewall rules are generated. The first set includes rules that have different destination address only, and the second set includes rules that have distinct source addresses. These two represent the best case scenario because they require the minimum policy-tree navigation for analyzing each rule. In the third set, each rule is a superset match of the preceding rule. This set represents the worst case scenario because each rule requires complete policy-tree navigation in order to analyze the entire rule set. The fourth set includes rules that are randomly selected from the three previous sets in order to represent the average case scenario. Policy Anomaly Checker is used to execute intra-firewall anomaly discovery algorithm on each set using various sizes of rule sets (10–100 rules). In each case, processing time is measured. The results are shown in Figure 4.1.

In order to obtain operational values of the inter-firewall anomaly discovery algorithm, two different experiments are performed. In the first experiment, the discovery algorithm is executed on a set of firewalls that exist on one network path. The rules that are used in each firewall are similar to set 2 rules in the previous test case. Number of rules in each firewall and the number of firewalls on the path is increased. The results are shown in Figure 4.2.

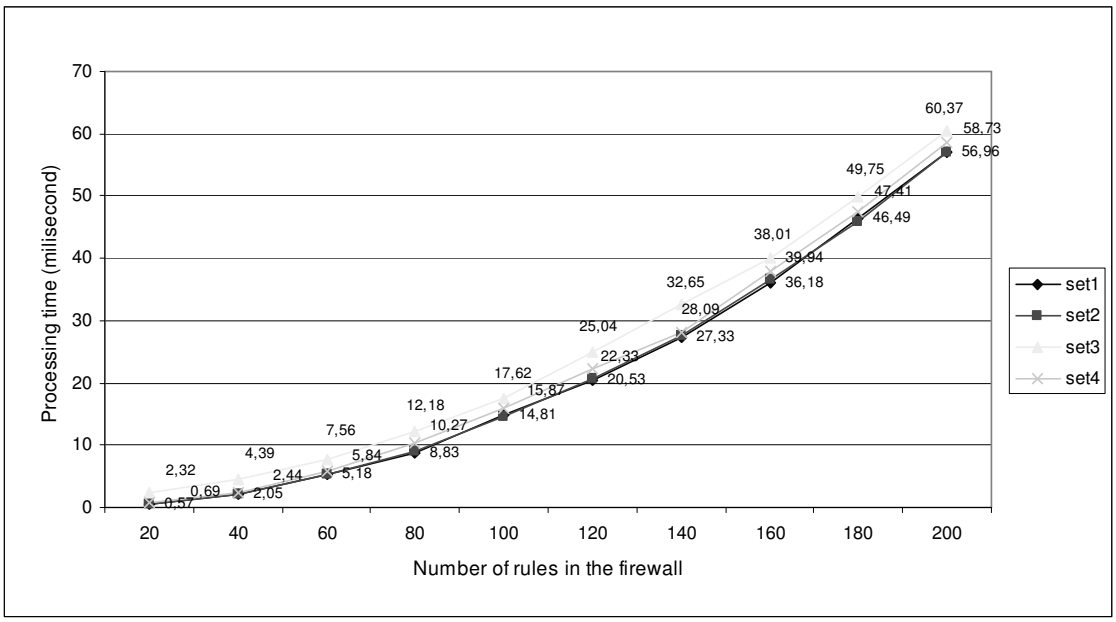



Figure 4.1. Processing Time for Intra-firewall Anomaly Discovery Algorithm

As illustrated in Figure 4.2 the processing time of the inter-firewall anomaly discovery algorithm is very close to processing time of intra-firewall algorithm for number of rules. For example, it takes 10 ms to analyze four firewalls each containing 20 rules. This is almost equal to the time required to perform intra-firewall algorithm on a single firewall having 80 rules.

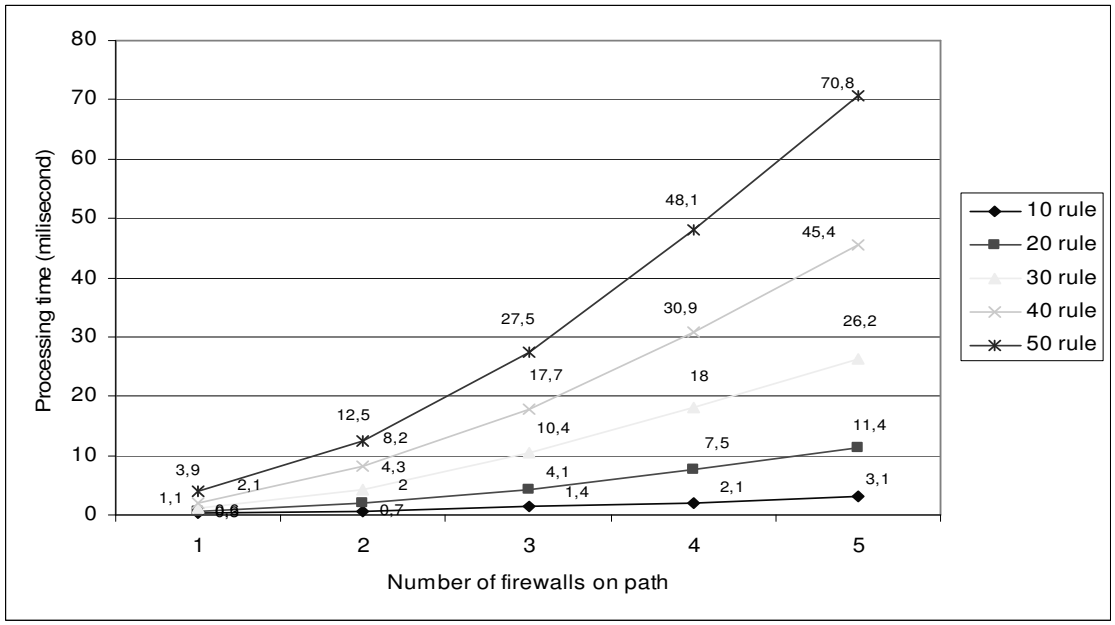



Figure 4.2. Inter-firewall Anomaly Discovery Algorithm for One Path

In the second experiment, inter-firewall anomaly discovery algorithm is executed for the different network topologies. Network paths which are used in the execution of anomaly discovery algorithm are created for three different networks. These networks are in the form of (1) 2-2-2, (2) 3-2-2 and (3) 3-3-2. For example, the root node in network 2 has 3 branches, whereas every node on levels two and three has two branches. For each network, a random set of filtering rules in each firewall are installed. In each network, the processing time required to produce the final policy anomaly report is measured. The results are shown in Figure 4.3.

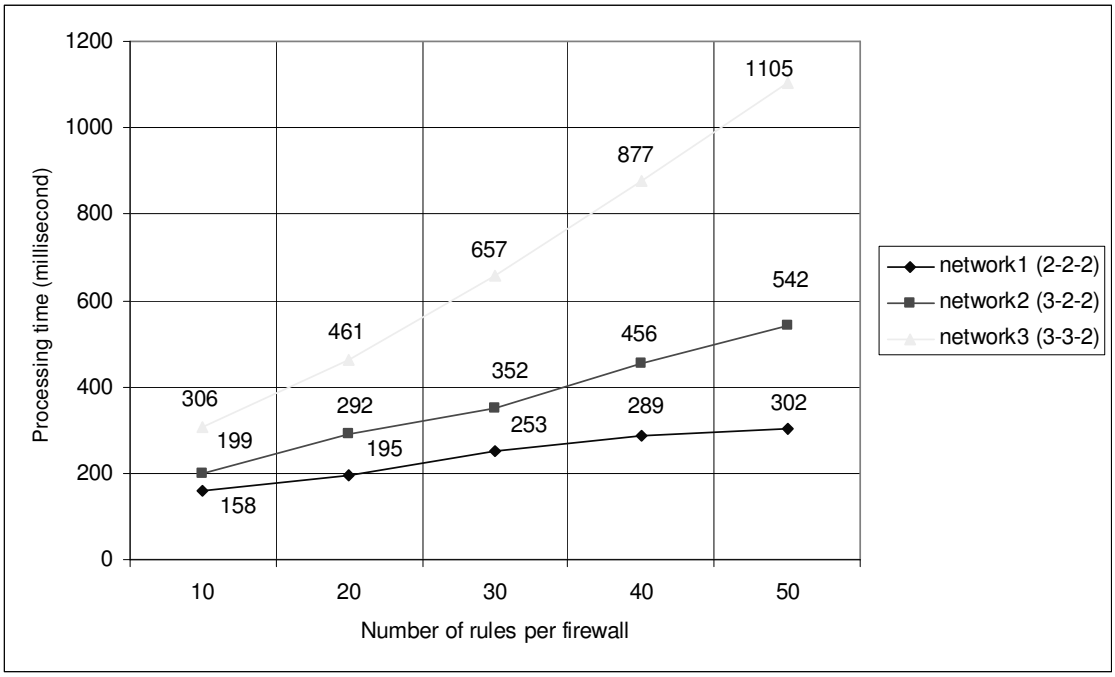



Figure 4.3. Inter-firewall Processing Time

These results indicate that inter-firewall anomaly algorithm is dependent on the total number of paths between sub-domains in the network. Since number of paths in network 3 is more than in other networks and network 3 has much processing time than others.

Finally, in order to obtain operational values of the inter-firewall anomaly discovery algorithm in real distributed environment a number of experiments are performed. Experiment environment is illustrated in Figure 4.4.

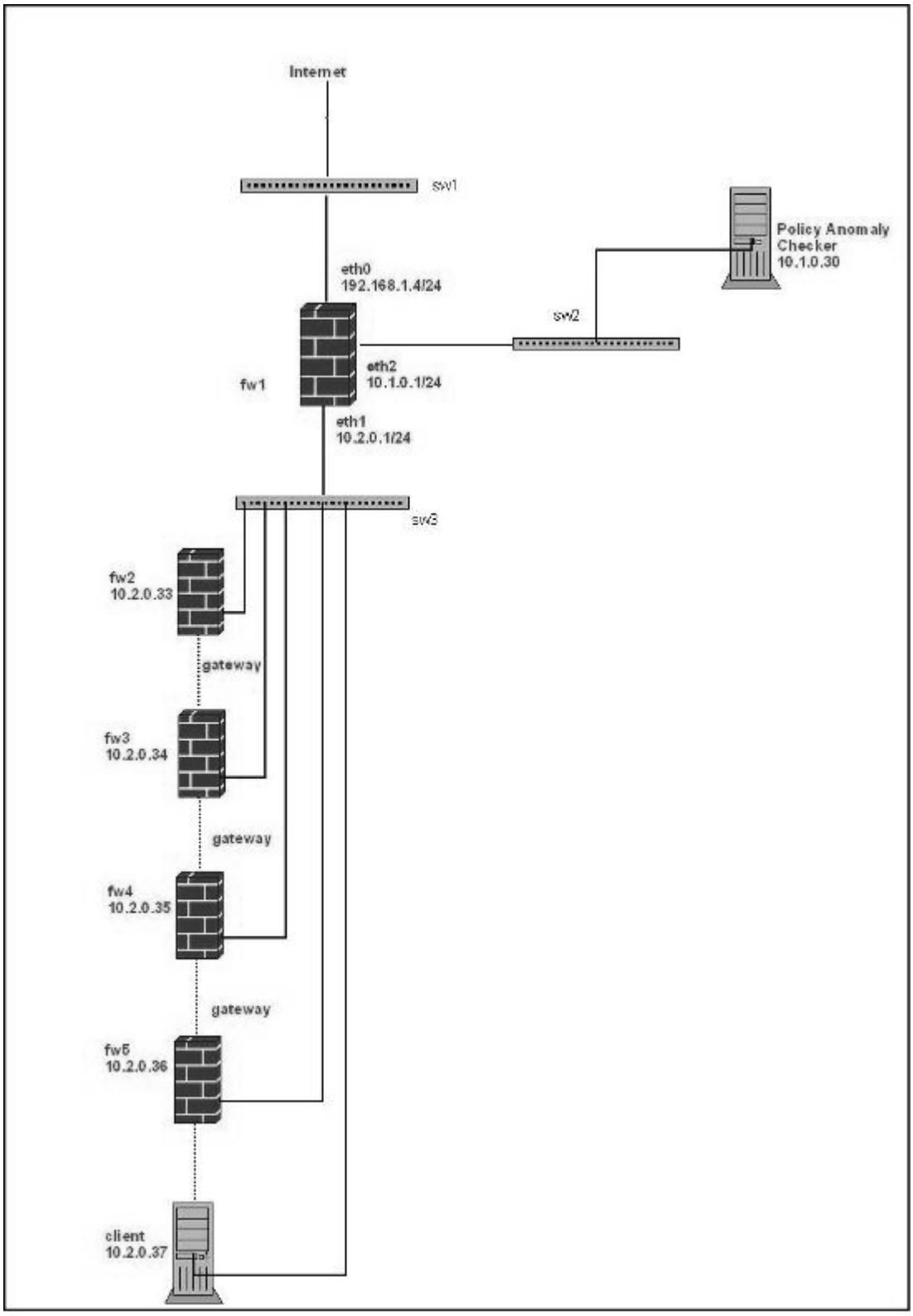



Figure 4.4. Environment used for experiment

Traffic flow from Internet to client machine is through fw1, fw2, fw3, fw4, fw5 and traffic flow from client to Internet is through fw5, fw4, fw3, fw2, fw1. All firewalls are dual home hosts and connected with Ethernet cable from their standard 10/100Mbs PCI Ethernet cards to 48 port Cisco switches. Firewall configurations are illustrated in Appendix C.

Policy Anomaly Checker is deployed on Pentium IV-M 1.73 GHz. processor with 1.49 GByte of RAM. In order to communicate with Firewall Agent, socket communication functionality is added to Policy Anomaly Checker. Firewall Agent is deployed on each firewall on the path. It is developed by using Python. The main functionality of Firewall Agent is to communicate firewall devices with Policy Anomaly Checker. Communication between Policy Anomaly Checker and Firewall Agent has four steps:

Step1: Firewall Agent sends a new rule to Policy Anomaly Checker which will be added to its policy.

Step2: Policy Anomaly Checker receives the rule, checks the relation with other rules and sends the anomaly result to Firewall Agent.

Step3: If there is no anomaly determined, Firewall Agent adds the new rule to its policy and sends the modified policy to the Policy Anomaly Checker.

Step4: Firewall Agent displays a warning message.

Communication protocol between Policy Anomaly Checker and Firewall Agent as follows;

Step1: Firewall Agent sends *bor (begin of rule)* message.

Step2: Firewall Agent sends the new Rule.

Step3: Firewall Agent sends *eor (end of rule)* message.

Step4: Policy Anomaly Checker sends a string compiled of anomaly result. If there is no anomaly sends *ok* message.

Step5: Firewall Agent sends *bof (begin of file)* message.

Step6: Firewall Agent sends policy file.

Step7: Firewall Agent sends *eof (end of file)* message.

The test scenario is same with the number of firewalls in one path scenario which is illustrated in Figure 4.2.

In order to reach Policy Anomaly Checker
from fw2; fw1 must be passed through,
from fw3; fw2 and fw1 must be passed through,
from fw4; fw3, fw2 and fw1 must be passed through,
from fw5; all firewalls on path must be passed through.

First, a new rule is sent by Firewall Agent from the fw1 to Policy Anomaly Checker. When new rule is received, it is added to its policy which is defined in Policy Anomaly Checker and inter-firewall anomaly discovery algorithm is executed with this new policy. At the end of the algorithm anomaly results is sent back to Firewall Agent. This test is repeated for each firewall on the path. Total response time and inter-firewall anomaly discovery algorithm process time is measured for each firewall. Total response time is measured by the Firewall Agent, inter-firewall anomaly discovery algorithm process time is measured by the Policy Anomaly Checker and latency time is measured using (total response time - discovery algorithm process time) formula. The results are shown in Figure 4.5 and Figure 4.6.

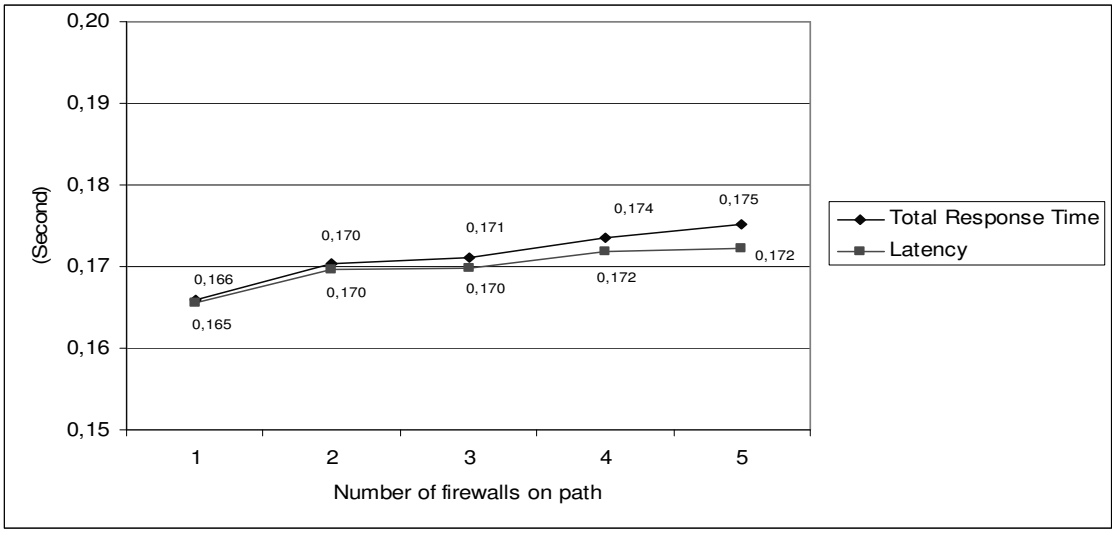



Figure 4.5. Experiment for Inter-firewall Anomaly Discovery Algorithm Using One Path Only (Total Response Time and Latency Values)

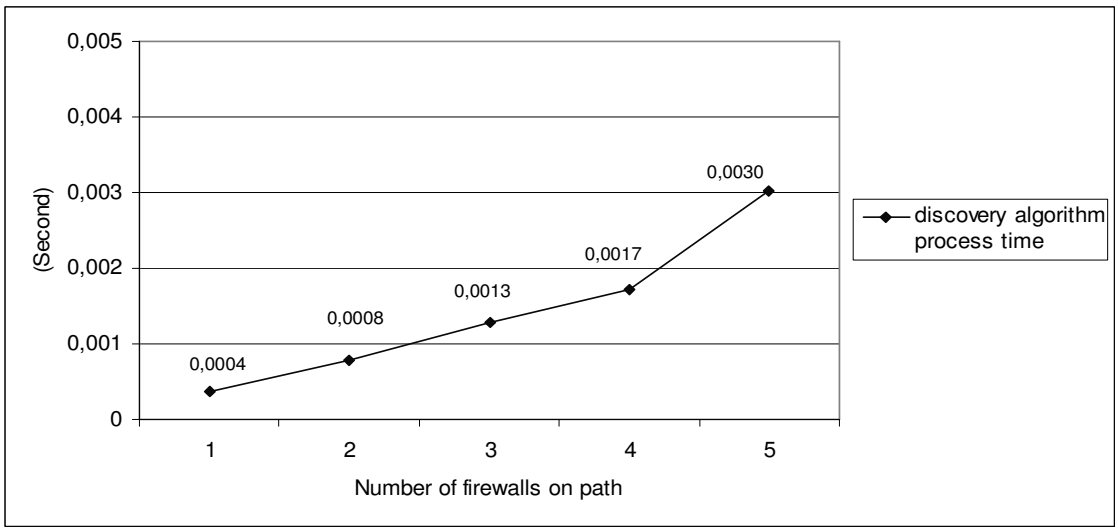



Figure 4.6 Experiment for Inter-firewall Anomaly Discovery Algorithm Using One Path Only (Discovery Algorithm Process Time)

These results indicate that inter-firewall anomaly discovery algorithm process time is very close to process time which is aforementioned in Chapter 3 for inter-firewall anomaly discovery algorithm process and it depends on the total number of rules in all firewalls.

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

In this thesis anomaly discovery algorithms for single and distributed environments are implemented in a software tool called “Policy Anomaly Checker”. Besides its anomaly discovery functionality, anomaly free rule set generation functionality is added to this tool.

Correct operation of the firewall is dependent on the filtering rules and their order. If the same packet matches more than one filtering rule in a policy, an intra-firewall anomaly may exist. If any two firewalls on a network path take different filtering actions, an inter-firewall anomaly may exist. All rule relation must be considered in order to determine correct rule order.

Mathematical background and related algorithms are available in literature. Although it is implied in (Al-Shaer and Hamed 2004) that an implementation exists and some experimental results are given, implementation details are not explained. So Policy Anomaly Checker is implemented using object oriented techniques in Java. Test cases are created.

Policy Anomaly Checker now only checks anomalies for intra and inter-firewall environments. Rule addition and rule removal functionality can be added for policy management. Other algorithms (Cuppens, et al. 2005) can be implemented and Intra and inter-firewall anomaly algorithms can be compared with other algorithms. Policy Anomaly Checker can be used in campus environment in order to determine its usability.

# REFERENCES


Al-Shaer, E.S. and H.H. Hamed. 2002. Design and Implementation of Firewall Policy Advisor Tools. *DePaul CTI Technical Report* CTI-TR-02-006

Al-Shaer, E.S. and H.H. Hamed. 2004. Discovery of Policy Anomalies in Distributed Firewalls. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China.*

Al-Shaer, E.S., Hamed, H.H., Boutaba, R. and H. Masum. 2005. Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications* 23(10): 2069-2084.

Bellovin, S. M. 1999. Distributed Firewalls. *Login*; *special issue on security.*

Chapman, Brent and Elizabeth Zwicky, eds. 1995. *Building Internet Firewalls*. Cambridge: Orielly & Associates Inc.

Cooper, James W. 2000. *Java™ Design Patterns: A Tutorial*. Boston: Addison Wesley.

Cuppens, F., Cuppens-Boulahia, N. and J. Garc´ıa-Alfaro. 2005. Detection and Removal of Firewall Misconfiguration. In *Proceedings of the International Conference on Communication, Network and Information Security, Phoenix, USA*

Grant, J., Attfield, P. and K. Armstrong. 2001. Distributed Firewall Technology. *EWA,CANADA*

Ioannidis, S., Keromytis, A.D., Bellovin, S.M. and J.M. Smith. 2000. Implementing a Distributed Firewall. In *Proceedings of the ACM Conference on Computer and Communication Security, Athens, Greece.*

Wack, J., Cutler, K. and J. Pole. 2002. Guidelines on Firewalls and Firewall Policy. *NIST Special Publication* 800-41

# APPENDIX A

# DEFINITIONS OF RELATIONS BETWEEN RULES

$\Re$ is the universal set of rule relation and $\Re = \{R_{CD}, R_{PD}, R_{EM}, R_{IM}, R_C\}$

**Definition 1:** Rules $R_x$ and $R_y$ are exactly matched if every field in $R_x$ is equal to the corresponding field in $R_y$. Formally:

$Rx \; \Re_{EM} \; Ry$ iff

$\forall$ i: $R_x[i] = R_y[i]$

where i $\in$ {protocol, src_ip, src_port, dst_ip, dst_port} (Al-Shaer and Hamed 2002).

**Definition 2:** Rules $R_x$ and $R_y$ are inclusively matched if they do not exactly match and if every field in $R_x$ is a subset or equal to the corresponding field in $R_y$. Formally:

$R_x \; \Re_{IM} \; R_y$ iff

$\forall$ i : $R_x[i] \subseteq R_y[i]$ and $\exists$ j such that: $R_x[j] \neq R_y[j]$

where i, j $\in$ {protocol, src_ip, src_port, dst_ip, dst_port}

In this relation, $R_x$ is called the subset match while $R_y$ is called the superset match (Al-Shaer and Hamed 2002).

**Definition 3:** Rules $R_x$ and $R_y$ are completely disjoint if every field in $R_x$ is not a subset and not a superset and not equal to the corresponding field in $R_y$. Formally:

$R_x \; \Re_{CD} \; R_y$ iff

$\forall$ i : $R_x[i] \rhd / \lhd R_y[i]$

where $\rhd \lhd \in \{\supset, \subset, =\}$, i $\in$ {protocol, src_ip, src_port, dst_ip, dst_port} (Al-Shaer and Hamed 2002).

**Definition 4:** Rules $R_x$ and $R_y$ are partially disjoint (or partially matched) if there is at least one field in $R_x$ that is a subset or a superset or equal to the corresponding field in $R_y$, and there is at least one field in $R_x$ that is not a subset and not a superset and not equal to the corresponding field in $R_y$. Formally:

$R_x \, \Re_{PD} \, R_y$ iff

$\exists$ i,j such that : $R_x[i] \rhd\lhd R_y[i]$ and $R_x[j] \rhd/\lhd R_y[j]$

where $\rhd\lhd \in \{\supset, \subset, =\}$and i,j $\in$ {protocol, src_ip, src_port, dst_ip, dst_port} (Al-Shaer and Hamed 2002).

**Definition 5:** Rules $R_x$ and $R_y$ are correlated if some fields in $R_x$ that are subset or equal to the corresponding field in $R_y$, and the rest of fields in Rx are supersets of the corresponding fields in $R_y$. Formally:

$R_x \, \Re_C \, R_y$ iff

$\forall$ i : $R_x[i] \rhd\lhd R_y[i]$ and

$\exists$ i,j such that : $R_x[i] \subset R_y[i]$ and : $R_x[j] \supset R_y[j]$

where $\rhd\lhd \in \{\supset, \subset, =\}$and i,j $\in$ {protocol, src_ip, src_port, dst_ip, dst_port} (Al-Shaer and Hamed 2002).

# APPENDIX B

# ANOMALY TYPES

**Intra-Firewall Anomaly Types:**

**Shadowing anomaly:** Formally rule $R_y$ is shadowed by rule $R_x$ if one of the following conditions holds:

$R_x$ [order]<$R_y$ [order], $R_x$ $\Re_{EM}$ $R_y$, $R_x$ [action] ≠ $R_y$ [action] (1)

$R_x$[order]<$R_y$ [order], $R_y$ $\Re_{IM}$ $R_x$, $R_x$ [action] ≠ $R_y$ [action] (2) (Al-Shaer, et al. 2005)

**Correlation anomaly:** Formally rule $R_x$ and rule $R_y$ have correlation anomaly the following condition holds:

$R_x$ $\Re_C$ $R_y$, $R_x$ [action] ≠ $R_y$ [action] (3) (Al-Shaer, et al. 2005)

**Generalization anomaly:** Formally rule $R_y$ is generalization of rule Rx if the following condition holds:

$R_x$ [order] < $R_y$ [order], $R_x$ $\Re_{IM}$ $R_y$, $R_x$ [action] ≠ $R_y$ [action] (4) (Al-Shaer, et al. 2005)

**Redundancy anomaly:** Formally rule $R_y$ is redundant to rule $R_x$ if one of the following conditions holds:

$R_x$ [order] < $R_y$ [order], $R_x$ $\Re_{EM}$ $R_y$, $R_x$ [action] = $R_y$ [action] (5)

$R_x$ [order] < $R_y$ [order], $R_y$ $\Re_{IM}$ $R_x$, $R_x$ [action] = $R_y$ [action] (6) (Al-Shaer, et al. 2005)

**Inter-Firewall Anomaly Types:**

**Shadowing anomaly:** Formally rule $R_d$ is shadowed by rule $R_u$ if one of the following conditions holds:

$R_d \Re_{EM} R_u$, $R_u$ [action] = deny, $R_d$ [action] = accept (1)

$R_d \Re_{IM} R_u$, $R_u$ [action] = deny, $R_d$ [action] = accept (2)

$R_u \Re_{IM} R_d$, $R_u$ [action] = deny, $R_d$ [action] = accept (3)

$R_u \Re_{IM} R_d$, $R_u$ [action] = accept, $R_d$ [action] = accept (4) (Al-Shaer, et al. 2005)

**Spuriousness anomaly:** Formally rule $R_u$ allows spurious traffic to rule $R_d$ if one of the following conditions holds:

$R_u \Re_{EM} R_d$, $R_u$ [action] = accept, $R_d$ [action] = deny (5)

$R_u \Re_{IM}$ Rd, $R_u$ [action] = accept, $R_d$ [action] = deny (6)

$R_d \Re_{IM} R_u$, $R_u$ [action] = accept, $R_d$ [action] = deny (7)

$R_d \Re_{IM} R_u$, $R_u$ [action] = accept, $R_d$ [action] = accept (8)

$R_u \Re_{IM} R_d$, $R_u$ [action] = deny, $R_d$ [action] deny (9) (Al-Shaer, et al. 2005)

**Redundancy anomaly:** Formally rule $R_d$ is redundant to rule $R_u$ if one of the following conditions holds:

$R_d \Re_{EM} R_u$, $R_u$ [action] = deny, $R_d$ [action] = deny (10)

$R_d \Re_{IM} R_u$, $R_u$ [action] = deny, $R_d$ [action] = deny (11) (Al-Shaer, et al. 2005)

**Correlation anomaly:** as a result of having two correlated rules in upstream and downstream firewall a correlation anomaly occurs. Formally, the correlation anomaly for rules $R_u$ and $R_d$ occurs if one of the following conditions holds:

$R_u \Re_C R_d$, $R_u$ [action] = accept, $R_d$ [action] = accept (12)

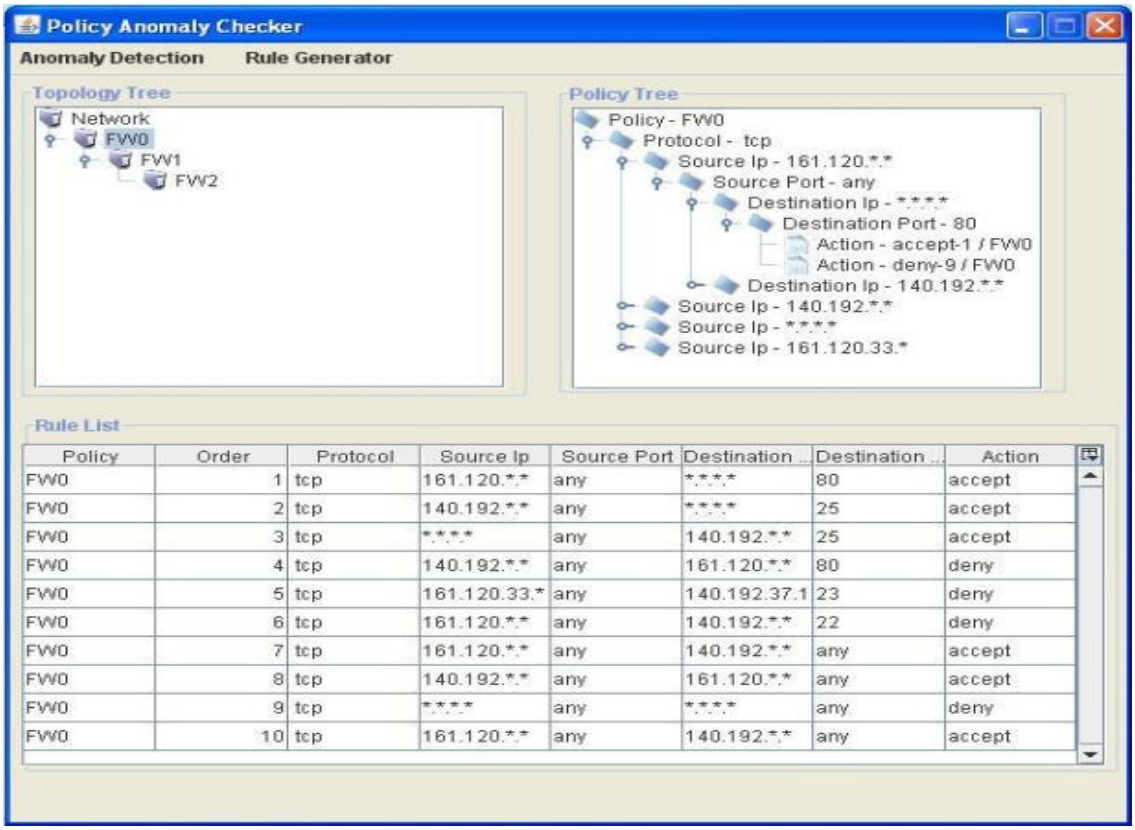
$R_u \Re_C R_d$, $R_u$ [action] = deny, $R_d$ [action] = deny (13)

$R_u \Re_C R_d$, $R_u$ [action] = accept, $R_d$ [action] = deny (14)

$R_u \Re_C R_d$, $R_u$ [action] = deny, $R_d$ [action] = accept (15) (Al-Shaer, et al. 2005)

# APPENDIX C

# USER INTERFACES OF POLICY ANOMALY CHECKER




| Policy | Order | Protocol | Source Ip | Source Port | Destination ... | Destination ... | Action |
|---|---|---|---|---|---|---|---|
| FW0 | 1 | tcp | 161.120.*.* | any | *.*.*.* | 80 | accept |
| FW0 | 2 | tcp | 140.192.*.* | any | *.*.*.* | 25 | accept |
| FW0 | 3 | tcp | *.*.*.* | any | 140.192.*.* | 25 | accept |
| FW0 | 4 | tcp | 140.192.*.* | any | 161.120.*.* | 80 | deny |
| FW0 | 5 | tcp | 161.120.33.* | any | 140.192.37.1 | 23 | deny |
| FW0 | 6 | tcp | 161.120.*.* | any | 140.192.*.* | 22 | deny |
| FW0 | 7 | tcp | 161.120.*.* | any | 140.192.*.* | any | accept |
| FW0 | 8 | tcp | 140.192.*.* | any | 161.120.*.* | any | accept |
| FW0 | 9 | tcp | *.*.*.* | any | *.*.*.* | any | deny |
| FW0 | 10 | tcp | 161.120.*.* | any | 140.192.*.* | any | accept |

Figure C.1. User Interface of the Policy Anomaly Checker

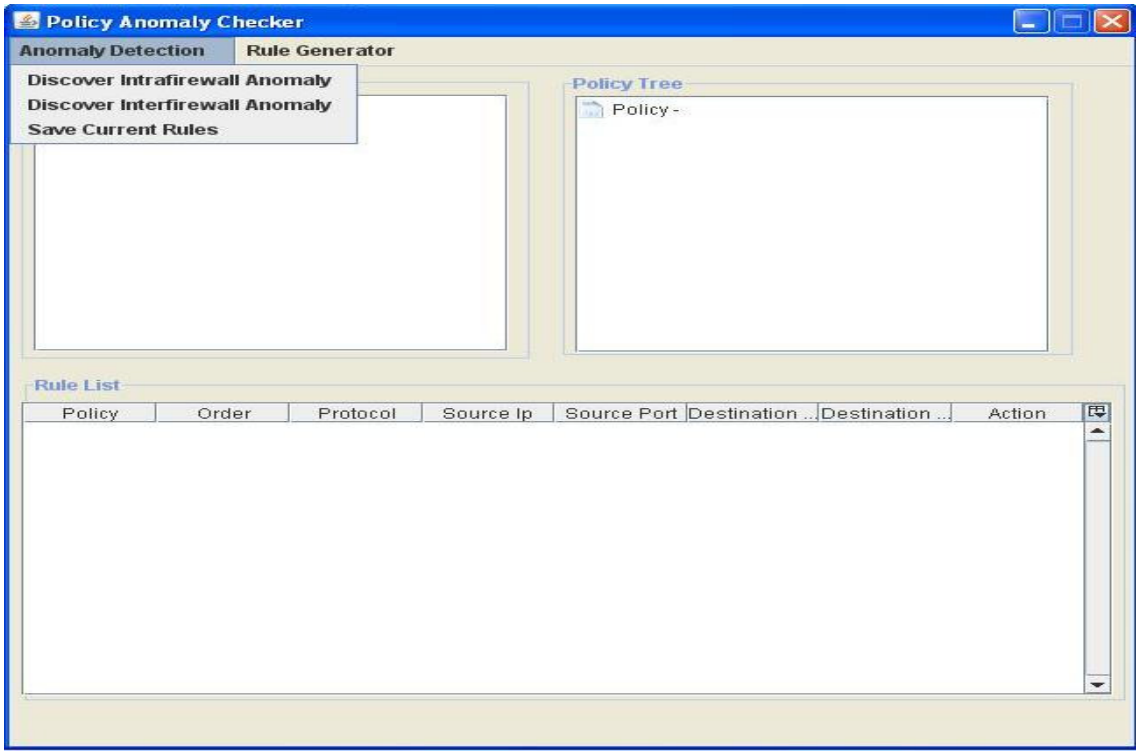



Figure C.2. User Interface of the Policy Anomaly Checker with Anomaly Detection Menu

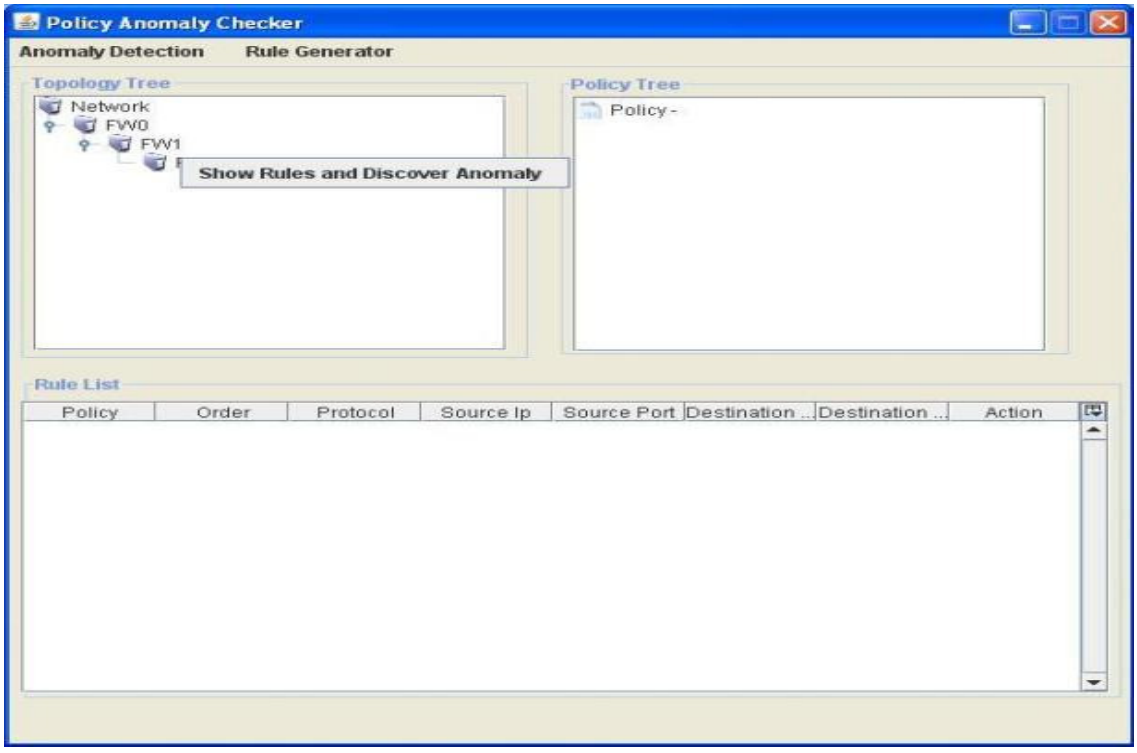



Figure C.3. User Interface of the Policy Anomaly Checker with Topology Menu

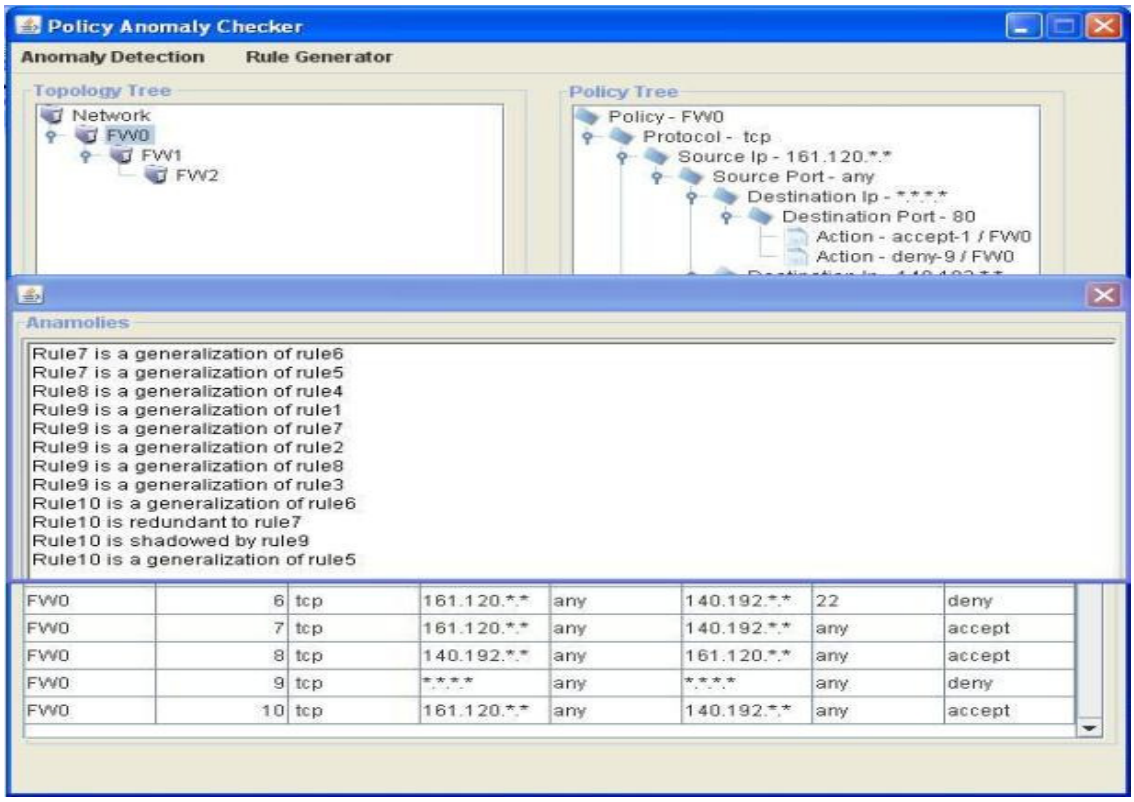



Figure C.4. Anomaly Result Form

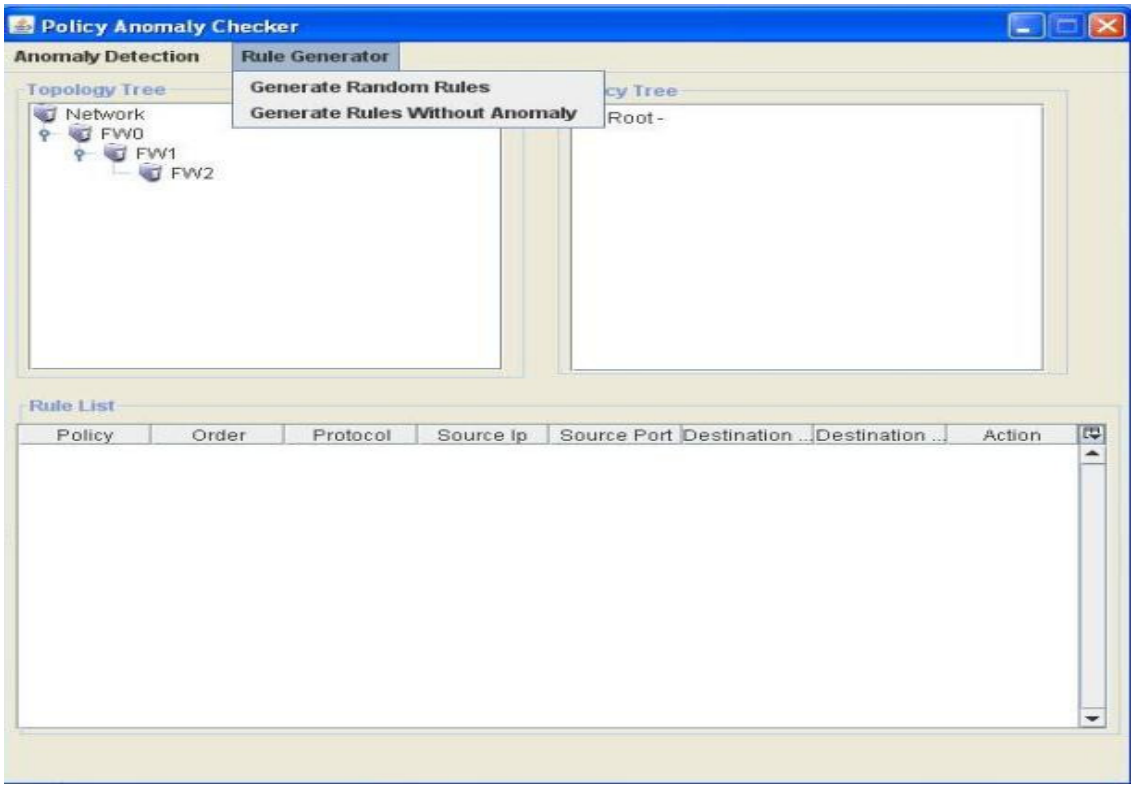



Figure C.5. Rule Generation Menu

# APPENDIX D

# FIREWALL CONFIGURATIONS

The firewalls are configured via iptables. Route definitions of fw1 to fw5 as follows.

fw1 route tables:

# route –n

Kernel IP routing table

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.2.0.34 | 10.2.0.33 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth1 |
| 10.2.0.35 | 10.2.0.33 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth1 |
| 10.2.0.36 | 10.2.0.33 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth1 |
| 10.2.0.37 | 10.2.0.33 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth1 |
| 10.2.0.0 | 10.2.0.1 | 255.255.255.0 | UG | 0 | 0 | 0 | eth1 |
| 10.2.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth1 |
| 192.168.1.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth0 |
| 10.1.0.0 | 10.1.0.1 | 255.255.255.0 | UG | 0 | 0 | 0 | eth2 |
| 10.1.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth2 |
| 0.0.0.0 | 192.168.1.3 | 0.0.0.0 | UG | 0 | 0 | 0 | eth0 |

Fw2 route tables:

# route -n

Kernel IP routing table

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.2.0.35 | 10.2.0.34 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.36 | 10.2.0.34 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.37 | 10.2.0.34 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | eth0 |
| 0.0.0.0 | 10.2.0.33 | 0.0.0.0 | UG | 0 | 0 | 0 | eth0 |

fw3 route tables:
# route -n

Kernel IP routing table

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.2.0.36 | 10.2.0.35 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.37 | 10.2.0.35 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | 0 eth0 |
| 0.0.0.0 | 10.2.0.33 | 0.0.0.0 | UG | 0 | 0 | 0 | eth0 |

By these route definitions, any package that is sent from client to the anomaly checker will follow the firewalls from bottom to the fw1.

Fw4 route tables:

# route -n

Kernel IP routing table

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.2.0.37 | 10.2.0.36 | 255.255.255.0 | UGH | 0 | 0 | 0 | eth0 |
| 10.2.0.0 | 0.0.0.0 | 255.255.255.0 | U | 0 | 0 | 0 | 0 eth0 |
| 0.0.0.0 | 10.2.0.34 | 0.0.0.0 | UG | 0 | 0 | 0 | eth0 |