Firewall Rule Set Analysis and Visualization

by

Pankaj Kumar Khatkar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2014 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Gail-Joon Ahn
Violet R. Syrotiuk

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

A firewall is a necessary component for network security and just like any regular equipment it requires maintenance. To keep up with changing cyber security trends and threats, firewall rules are modified frequently. Over time such modifications increase the complexity, size and verbosity of firewall rules. As the rule set grows in size, adding and modifying rule becomes a tedious task. This discourages network administrators to review the work done by previous administrators before and after applying any changes. As a result the quality and efficiency of the firewall goes down.

Modification and addition of rules without knowledge of previous rules creates anomalies like shadowing and rule redundancy. Anomalous rule sets not only limit the efficiency of the firewall but in some cases create a hole in the perimeter security. Detection of anomalies has been studied for a long time and some well established procedures have been implemented and tested. But they all have a common problem of visualizing the results. When it comes to visualization of firewall anomalies, the results do not fit in traditional matrix, tree or sunburst representations.

This research targets the anomaly detection and visualization problem. It analyzes and represents firewall rule anomalies in innovative ways such as hive plots and dynamic slices. Such graphical representations of rule anomalies are useful in understanding the state of a firewall. It also helps network administrators in finding and fixing the anomalous rules.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Dijiang Huang for his constant support and constructive feedback. Working with him at the SNAC Lab has been fun and a great learning experience. I would also like to thank my committee members for their insightful comments.

Behind this research is the guidance of my internship mentor Christian Romano. During my summer internship at CAaNES I have learned a lot and applied the knowledge in this research. This work is also supported by knowledge from my lab mates, especially Chun-Jen Chung. All members from the SNAC Lab have been very helpful in sharing and contributing ideas to this research and the work I have done in the past. I would also like to thank my roommate Harsh Vachhani for being a good friend and creating an encouraging atmosphere around me. Surviving the grad school was made easy by teaching assistantships from ASU and research assistantship from my advisor.

I am grateful to my parents and family who supported and stood by me all these years.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

A firewall is an essential component of network security infrastructure. A firewall sits on the boundary between a network that needs to be protected and the network that is considered to be unsafe. A simple firewall configuration is shown in Figure 1. Hosts 192.168.1.2 and 192.168.1.3 on the internal network are shielded by the firewall. We will use the term ‘internal network’ to represent network(s) that need to be protected.

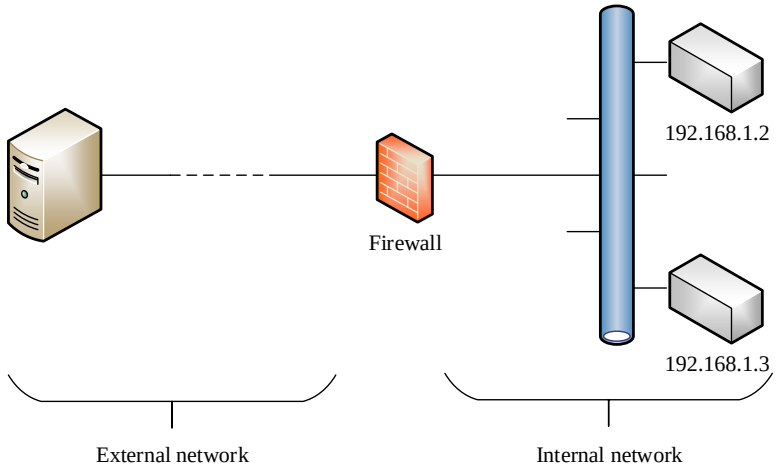

Figure 1. Simple firewall configuration

Depending on the type of data handled and inspected by the firewall, it can be classified in three generations. The first is a ‘packet based’ firewall. These firewalls do not consider if the packet being inspected is part of ongoing traffic. Every single packet is inspected against the firewall rule set. If the packet matches any rule, the respective action is taken, otherwise default action is applied to the packet. The second generation of firewall performs ‘state inspection’. These firewalls keep a history of ongoing connections.

Whenever a new packet comes in, the firewall examines if this is part of an ongoing connection or the start of a new one. Appropriate actions are taken if the packet matches a rule. The rules are similar to those of 'state' based firewalls but contain the connection state information. Little advanced from the first and second generation firewall is the 'application' based firewall. This third generation firewall not only performs functions of the previous two generations but can also distinguish between packets belonging to the same set of source and destination IP addresses or port numbers. These firewalls work at the application layer of the OSI model.

Firewalls are an active part of network security. There are several occasions when rule sets need to be changed. Whenever a new host is added to the internal network, a firewall rule needs to be added so that this new host can communicate with the external network. Sometimes internal services change which require modification to existing rules. Removal of a service/host requires removal of the corresponding rule from the firewall, although it is not necessary. Whenever the firewall is modified the network administrator must ensure that it is free from anomalies and does not contain redundant or overlapping rules. For a network of hundreds of hosts the firewalls can become very complex over time and this gives rise to anomalies. Over the years, firewall rules and the language in which they are specified have become very complex. Firewalls these days need a dedicated GUI for managing the rules. The complexity of the firewall specification language and vendor dependency has made it very difficult for network administrators to audit the firewall. This thesis targets the problem of firewall rule set analysis and presents visualizations that help in auditing of firewalls. We will describe rule anomalies in the next section.

Rule Set Anomalies

Several rule set anomalies have been discussed in [6], [7], [19] and [20]. Anomalies can be intra firewall and inter firewall. This work targets five intra firewall anomalies. Before we describe and define the anomalies let us define some notations. A firewall rule is denoted by R_i where i is a positive integer denoting the rule number. R_i precedes R_j if $i<j$. Each rule has an action denoted by *Action* (R_i). We assume that a firewall rule has only one of the two actions: permit or deny. Each rule consists of protocol, source port, source address, destination port and destination address. We use the term *PacketSpace* (R_i) to denote the packet space of ports, addresses and protocol. This five dimensional space is all that is needed to identify a packet uniquely at layer 2 of the OSI stack. The firewall is also assumed to have default action of denying all packets which do not match any rule. This work deals with five types of anomalies. We will first give a formal definition and then explain each of them in detail with examples.

1. Redundancy

   Rule R_j is redundant to R_i if *Action* (R_i) = *Action* (R_j) and *PacketSpace* $(R_j) \subseteq$ *PacketSpace* (R_i), where $i<j$. Rule R_j is redundant because it deals with the same packet space that has been dealt by its preceding rule R_i. In this case rule R_j is never executed. It is important to note that execution of R_j is not critical because it has the same action as that of R_i. In table 1, rule 2 is redundant to rule 1.

2. Shadowing

   Rule R_j is shadow of R_i if *Action* $(R_j) \neq$ *Action* (R_i) and *PacketSpace* $(R_j) \subseteq$ *PacketSpace* (R_i), where $i<j$. Rule R_j is a shadow of R_i because R_j is never executed. Shadowing is similar to redundancy except that it is more severe to

network security. A different action for R_j shows a security policy conflict inside the firewall. In such case it is difficult to determine which rule should be removed in order to remove shadowing.

3. Generalization

   Rule R_j is generalization of R_i if $PacketSpace\ (R_i) \subset PacketSpace\ (R_j)$, where $i<j$. Generalization is an anomaly because rule R_j is executed for some packets not captured by R_i. Some packets are shared between R_i and R_j but their actions may or may not be different. If $Action\ (R_j) \neq Action\ (R_i)$ it is a security conflict for the shared space. It can be observed that the first rule takes care of some of the packets which were intended to be taken care of by the second rule.

4. Correlation

   Rule R_j is correlated to R_i if $Action\ (R_j) \neq Action\ (R_i)$, and R_j overlaps R_i for $i<j$, i.e. $PacketSpace\ (R_j) \nsubseteq PacketSpace\ (R_i)$, $PacketSpace\ (R_j) \not\supseteq PacketSpace\ (R_i)$ and $PacketSpace\ (R_j) \cap PacketSpace(R_i) \neq \emptyset$. Rules R_i and R_j overlap and have a different action.

5. Overlapping

   Rule R_j overlaps R_i if $Action\ (R_j) = Action\ (R_i)$, and R_j overlaps R_i for $i<j$, i.e. $PacketSpace\ (R_j) \nsubseteq PacketSpace\ (R_i)$, $PacketSpace\ (R_j) \not\supseteq PacketSpace\ (R_i)$ and $PacketSpace\ (R_j) \cap PacketSpace\ (R_i) \neq \emptyset$. Rules R_i and R_j overlap in their packet space but their actions are the same.

Having seen the formal definitions of anomalies, let us see example of each. Table 1 shows an example of the redundancy anomaly. $Rule_2$ is redundant to $Rule_1$ because the source address of R_2 is subset of R_1. The same can be said about the destination port.

Because $Rule_2$ is never executed, it is safe to remove it in order to get rid of the redundancy.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.0/24	any	10.1.2.23	any
2	permit	tcp	192.168.1.2	any	10.1.2.23	80

Table 1: Redundancy anomaly example.

Table 2 gives an example of a shadowing anomaly. $Rule_2$ is a subset of $Rule_1$ and is therefore never executed. Because $Rule_2$ and $Rule_3$ have the same packet space as of $Rule_1$, they are never executed. We can say that $Rule_1$ shadows $Rule_2$ and $Rule_3$.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.0/24	any	10.1.2.23	any
2	deny	tcp	192.168.1.2	any	10.1.2.23	80
3	deny	tcp	192.168.0/24	any	10.1.2.23	any

Table 2: Shadowing anomaly example.

Table 3 gives an example of a generalization anomaly. Unlike other anomalies, generalization arises because a preceding rule is found to be a subset of its descendants. Looking at $Rule_2$, we see that it is a superset of $Rule_1$ and has the same action. A situation like this is not a security issue; it is more about efficiency.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80	10.1.2.23	any
2	permit	tcp	192.168.1.2	any	10.1.2.0/24	any
3	deny	tcp	192.168.0/24	any	10.1.2.23	any

Table 3: Generalization anomaly example.

With respect to $Rule_2$, $Rule_1$ is a duplicate. Considering $Rule_3$, we notice that $Rule_1$ is subset of $Rule_3$ and they have conflicting actions. It is a common practice for network administrators to add such rules if they want a smaller portion of the traffic to be cleared/blocked from a larger set of traffic.

Table 4 gives firewall rules involved in correlation. Both rules are similar except that they act on different source port ranges. $Rule_1$ permits a source port ranging from 80 to 90 (inclusive) and $Rule_2$ denies source ports 85 to 100. Intuitively, source ports 85-90 are common in both rules but the action is conflicting. To resolve such conflicts we will describe a '*split-n-merge*' in the Anomaly Resolution chapter.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80-90	10.1.2.23	any
2	deny	tcp	192.168.1.2	85-100	10.1.2.23	any

Table 4: Correlation anomaly example.

Table 5 lists overlapping firewall rules. Rules involved in an overlapping anomaly have the same action. Such conflicts can be solved by merging the overlapping fields. For example, merging source ports for $Rule_1$ and $Rule_2$ will yield the following rule:

permit tcp 192.168.1.2 80-100 10.1.2.23 any

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80-90	10.1.2.23	any
2	permit	tcp	192.168.1.2	85-100	10.1.2.23	any

Table 5: Overlapping anomaly example.

This is all the background knowledge needed. The 'split-n-merge' strategy for conflict detection is covered in the 'Anomaly Resolution' chapter. The next section covers the related work.

Related Work

One of the early solutions targeting the firewall auditing problem was developed by Mayer et al. [1]. Their system 'Fang' was designed to collect and parse the firewall configuration files and create an internal representation of the network topology. Using this internal representation Fang can take queries from the user and reports back the portion of the query that manages to pass through the network, from source to destination. This allowed network administrators to audit the firewall for any security loopholes and against spoofing attacks. This is a useful feature but it does not help with rule set anomaly detection. Anomalies can be detected with certain queries but such an approach is not discussed in their work. Another drawback of this work is that it requires a user to the specify network topology using the Firmato MDL language [2].

Wool [3] improved Fang by adding new features and named the product Lumeta Firewall Analyzer (LFA). It supported automatic generation of queries rather than inherently depending upon the user. LFA does not rely on the manual entry of network topology. Instead it reads the routing table and constructs the connectivity file. Other improvements over Fang included batch processing of practically every possible packet and changing the output format to HTML pages. LFA is flexible with different vendors because it uses an intermediate firewall configuration language and converts different vendor specific formats to this intermediate representation.

Lupu et al. [4] describes conflict analysis for management policies. The authors developed a tool to determine policy conflicts in a large-scale distributed system. It performs offline static analysis of policies and detects conflict. Fu et al. [5] present a

policy management system for IPSec in both the intra-domain and inter-domain environments. Eppstein et al. [6] deal with the packet classification and filter conflict detection problem. They use *kD-tree* [8] to check if any two filters acting on the same packet specify different actions. Hari et al. [7] describe a k-tuple filter for conflict detection. The conflict detection time of their algorithm in a 2-tuple mode is independent of the number of rules and conflict resolution is linear with the number of conflicts.

Al-Shaer et al. in [18] and [19] present a very comprehensive study of firewall rule set anomalies and describe a tool named the Firewall Policy Advisor (FPA). It comprises algorithms for automatically detecting anomalies and also suggests possible good points for insertion of new rules. Apart from anomaly detection, FPA also performs translation of rules into a high-level language for easy understanding of complex firewall rules. FPA classifies four anomalies namely, correlation, generalization, redundancy and shadowing. Their policy translation to a textual description is simple but the way this information is presented to the user is not intuitive. The description is textual in nature and this makes it harder to comprehend a set of hundreds of rules. They target inter and intra firewall anomaly discovery.

Yuan at al. [20] introduced a tool called FIREMAN that is capable of evaluating firewall misconfigurations as well as policy violations and inefficiencies. Their solution examines all paths between firewall interfaces but does not consider NAT or internal routing. An extension to FIREMAN was introduced as Prometheus [22]. This included support for NAT and internal routing but it does not handle change-impact across firewalls.

Govaerts et al. [21] explained a formal logic approach to firewall filter analysis. They use formal logic to detect anomalies such as non-existent hosts, action conflict among filters, and rules that may never receive traffic. Nelson et al. [23] describe a tool called Margrave that performs conflict detection and security compliance. It uses a query language and allows users to issue queries to check the effectiveness of the firewall. Hu et al. [24] and [25] described a Firewall Anomaly Management Environment (FAME) that uses a rule-based segmentation approach to detect conflict among firewall rules. FAME visualizes the result through a grid diagram showing rules and segments as rows and columns respectively. However, their representation is not suitable for a large number of rules and segments.

This thesis is about visualizing the rule set anomaly results, so it is important to see past research related to visualization of large datasets and firewall configurations in particular. Becker et al. [27] introduced a system called *SeeNet* that uses a static display with dynamic parameters to visualize the data associated with a network. They reason that spatial and matrix layouts become confusing for large networks. To solve this problem of cluttered information, they incorporate dynamic parameters when it comes to information visualization. Some of these parameters include the size of symbol, level of aggregation, time-interval and display threshold. Bertini et al. [28] present *SpiralView*, a visualization tool for assessing network security policies. It provides a comprehensive view of alarms generated in the network over a period of time and ties this information with network resources. Foresti et al. [30] present a system *VisAlert* that facilitates situational awareness through information visualization. Time based Network traffic Visualizer (TNV) was presented by Goodall et al. [31]. It uses a matrix representation of hosts and

network packet timestamps. Radial hierarchical frequent pattern visualization was presented in [32]. Similar to Sunburst [42] and Interring technique [29], their representation focuses on showing frequent item sets. Stasko et al. [42] explains a knowledge representation based approach to visualize network traffic. It allows users to store analysis results as a logical model, which can then be used later in future analysis.

A survey of visualizations for network security is presented in [38]. *SnortView* [33] is a visualization system for NIDS logs. Every log entry from Snort is visualized with sorted host IP addresses along the y-axis and time along x-axis. Visual Firewall [34] employs separate views for showing network security related information such as packet flow, throughput and suspicious network activity. Mansmann et al. [35] proposed hierarchical network maps where each child node is placed inside a parent node and number if children determine the size of the parent node. Another tool *PortVis* [36], visualizes network activity at three different levels: timeline, hour, port. Morrissey et al. [37] presented a concept of 'created void'. They visualize overlapping rule sets that prevent a packet from reaching an accept rule because it has been denied previously by another rule. Their representation can be best understood in three dimensional space, but the image itself doesn't convey much information. Also, they limit their analysis to two protocols TCP and UDP, which is far from comprehensive set of protocols used in industrial firewalls. *PolicyVis* [39] visualizes firewall rules to facilitate conflict detection and policy semantic discovery.

Research Outline

We have provided an introduction to firewall and rule set anomalies. In the following sections we will describe the concepts and algorithms used in my research. Chapter 2 starts with introduction to data structures we have used. It explains how the firewall rule entities like port and addresses can be converted to BDD and stitched together to create a binary expression representing the filter.

Chapter 3 on Anomaly Detection describes the algorithms and concepts behind anomaly detection. It explains segmentation and rule set analysis and provides background for understanding figures and visualizations. In Chapter 4 we will explain anomaly resolution strategies.

Chapter 5 presents a set of visualizations such as segment table, along with newer representations such as rule table and hive plot. We will see why some of the visual methods are not appropriate for the results generated. System's performance is evaluated in Chapter 6. There is always room for improvement with every research project. Some areas where my work can be extended is provided in Chapter 7.

Chapter 2

DATA STRUCTURES

Firewall rule set anomaly detection requires a data structure capable of handling set operations at high speeds. In [9] researchers have used Ordered Binary Decision Diagrams for packet classification. Srinivasan et al. [10] introduced Tuple Space Search algorithm that maintains each tuple as a hash table but is designed for 2-tuple filters. Woo [11] combines heuristic tree search with filter buckets to solve packet classification problem. Several other algorithms and data structures have been proposed in [13][14][15][16] but they are complex for firewall rule set anomaly detection. This work uses Binary Decision Diagram (BDD) to detect conflicts. The next section gives a brief idea of BDDs and explains how firewall rules are converted to a BDD.

Binary Decision Diagram

A Binary Decision Diagram (BDD) is a graph like data structure for representing a Boolean expression. A BDD is a rooted directed acyclic graph with two terminal nodes representing values 1 and 0 (true and false respectively). Every non-terminal node represents a Boolean variable. A path from the root to a terminal node is therefore a Boolean expression. Figure 2 shows some common Boolean expressions and their BDD representations. Every two non-terminal nodes are connected by two different arrows. A solid arrow represents a ‘true/1’ value and a dotted arrow represents a ‘false/0’. BDDs can be very large and complex, and they can also contain redundancy. Minimization of

BDDs is discussed in [40]. We will use the term BDD for Reduced Ordered BDD (ROBDD) throughout this report.

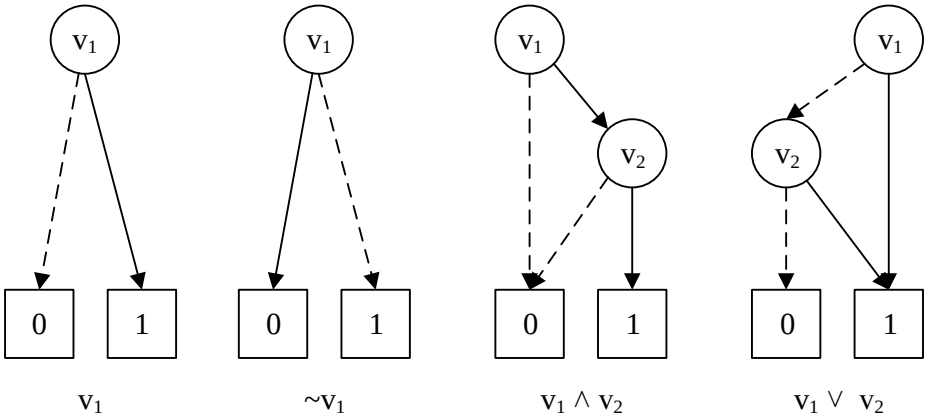

Figure 2. Common BDD expressions

Converting a Firewall Rule to a BDD

We have seen how Boolean expressions are converted to BDDs. To convert a firewall rule into a BDD, we first need to convert it to a Boolean expression. Let us see how to convert port numbers and IP addresses to Boolean expressions.

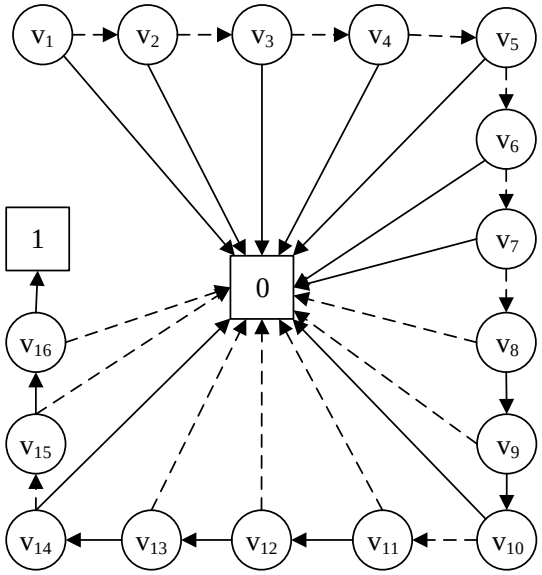

Figure 3. BDD for port number 443

Port Conversion

Port numbers can be specified as a fixed value or a range of values. If a port number is fixed, a simple conversion to binary number and then to a BDD can be done. The IP header allocates 16 bits for port numbers. Each of these bits can be thought of as a Boolean variable and the port number as a conjunction of these 16 variables. For example port number 80 can be first translated to binary number 0000 0000 0101 0000. Starting from the Most Significant Bit (MSB) the Boolean expression can be formed as follows:

$$\sim v_1 \wedge \sim v_2 \wedge \sim v_3 \wedge \ldots \wedge \sim v_9 \wedge v_{10} \wedge \sim v_{11} \wedge v_{12} \wedge \sim v_{13} \wedge \ldots \wedge \sim v_{16}$$

Figure 3 shows a BDD for port number 443. Port numbers can also be specified by inequality relations. For example 'less than 80', 'greater than 80'. Algorithm 1 describes a recursive method for generating BDDs. It generates BDDs from a binary string and an array of BDD variables. Creating a BDD for expressions such as 'greater than 80' is a matter of using the result of expression 'less than 80' and negating it.

```
Input: Port number, Array of BDD variables
Output: BDD expression that accepts all port numbers less than the input number.
s⟵binary translation of input port number
i⟵0
return RecurseBdd(s, i);

RecurseBdd (s, i)
  if (s.length<1)
    return true;
  if (s.substring(0,1)=0)
    return NOT(varArray[i]) AND RecurseBdd(s.substring(1), i+1)
  if (s.substring(0,1)=1)
     return varArray[i] OR (NOT(varArray[i]) AND RecurseBdd (s.substring(1), i+1))
```

Algorithm 1: Recursive function for generating BDD for port numbers

IP Address Conversion

CIDR notation of an IP address consists of four octets. Representing an IP address as a BDD is a simple operation of joining a BDD for each octet in the IP address. Figure 4 shows a BDD for IP address 192.168.1.2.

Network Address Conversion

To convert a network address with a given prefix into a BDD, first translate the network address to a binary representation and apply the network prefix to it. The resulting binary value can then be used to create a BDD. Figure 5 shows the conversion of network the address 192.168.5.64/26 to a BDD.

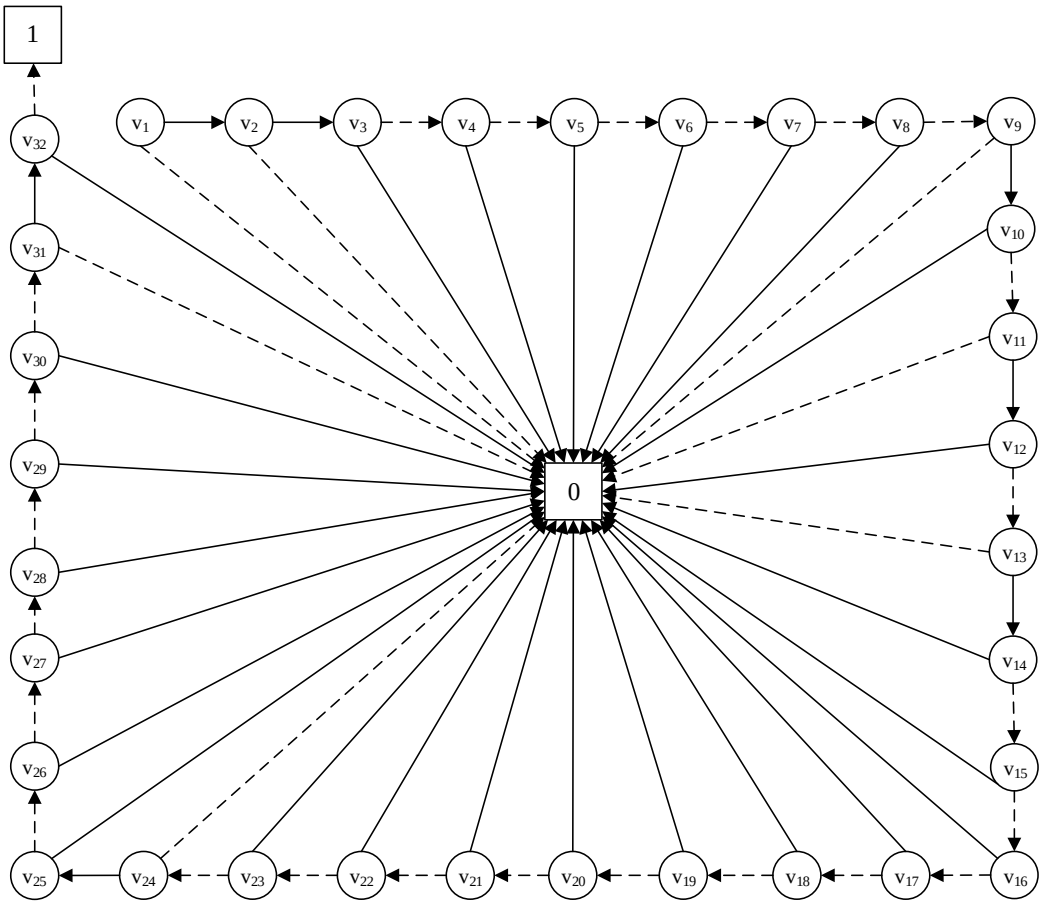

Figure 4. BDD for accepting IP address 192.168.1.2

We have seen conversion of port numbers and IP/network addresses to BDDs. Conversion of a BDD from a firewall rule simply involves conversion of a BDD for each of the source and destination port numbers and also the IP addresses. These separate BDDs can then be combined as conjunctive statements to get a BDD that represents the firewall rule.

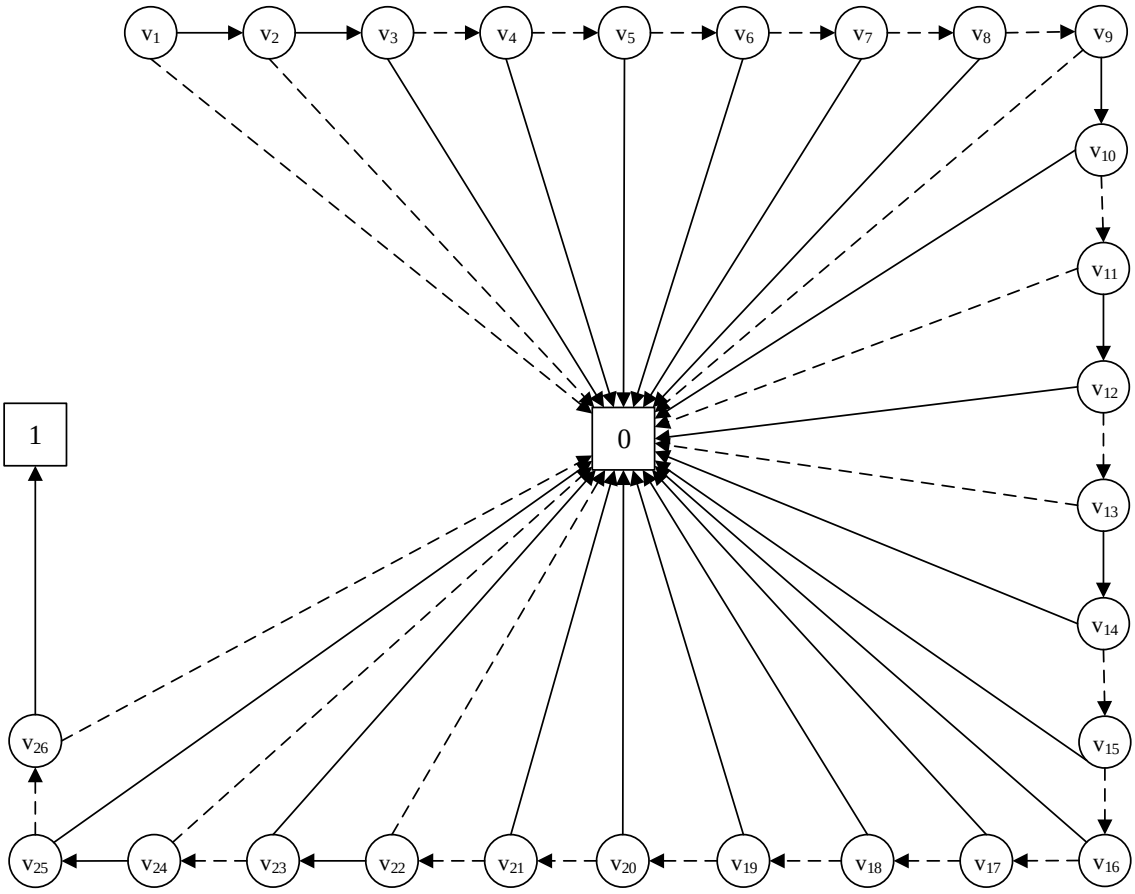

Figure 5. BDD for network address 192.168.5.64/26

Chapter 3

ANOMALY DETECTION

To detect firewall rule set anomalies, we used a segmentation based approach as described in [24]. Algorithm 2 is the segment generation algorithm. It is very similar to [24, Algorithm 1] but we have added annotations which make it easier to understand.

```
Input: A set of rules R.
Output: A set of segments S.
S ⟵ empty set;
for Rule r ∈ R do
    A ⟵ PacketSpace (r);
    skip ⟵ false;
    for Segment s ∈ S
        B ⟵ PacketSpace (s);
        if (A=B)
            skip ⟵ true;
            break;
        AnegateB ⟵ A ^ ~ B;
        BnegateA ⟵ B ^ ~ A;
        if (AnegateB = 0)
            S.add (BnegateA);
            s.packetSpace ⟵ A;
            skip ⟵ true;
            break;
        AandB ⟵ A ^ B;
        if (BnegateA=0)
            A ⟵ AnegateB;
        else if (AandB != 0)
            S.add (BnegateA);
            A ⟵ AnegateB;
    if (skip = false)
        S.add (A);
return S;
```

Algorithm 2: Segment generation algorithm

Segment Based Approach

Algorithm 2 begins with two sets, an empty set of segments and a set of rules belonging to a firewall. For each rule in the input rule set, it is compared with segments from the input segment set. If the segment set is empty a segment is created from the current rule in process and is stored in the segment set. A segment is a five dimensional entity: source address and port, destination address and port, and a protocol. Segments are different from the packet space in that they cannot overlap. Figure 6 gives a graphical explanation of how segments are created from packet space. A segment for rule R_i is denoted by S_i. In Figure 6(a), a packet space of two distinct rules R_1 and R_2 generates two segments, one for each. Figure 6(b) presents an example of rules having the same packet space. Keep in mind that segments do not consider rule action. That's why there is one segment representing both the rules.

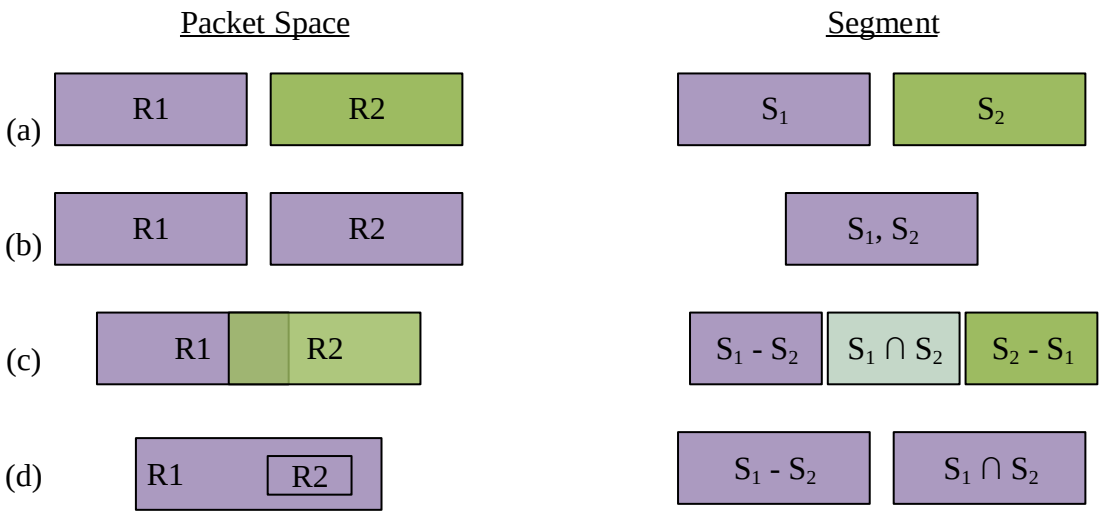

Figure 6. Segment generation example.

Figure 6(c) is a case of overlapping rules. Creating segments from a set of overlapped rules requires some set computation. In this particular case, three segments need to be

generated. The first segment is created by set subtraction of segments of R_1 and R_2. This segment is represented by S_1-S_2. Second segment denotes the packet space common between R_1 and R_2; therefore this segment can be generated by set intersection of S_1 and S_2. Finally, the third segment is obtained by set subtraction of S_1 from S_2. Figure 6(d) is an example of subset rules. Two segments are created in this scenario, one from removing packet space of R_2 from R_1 and another from performing set intersection of S_1 and S_2.

Following from [25], once segments are generated, we classify them into two categories: overlapping and non-overlapping. Non-overlapping segments are not our concern because they represent distinct set of rules. However, overlapping segments denote the existence of an anomaly. Overlapping segments are further divided into conflicting and non-conflicting segments as defined in [25]. If any two overlapping segments have a different action, they are deemed conflicting. If all overlapping segments have the same action (accept or deny), then they are called non-conflicting segments. Overlapping segments pose a risk to the firewall in terms of efficiency, but conflicting overlapping segments are more risky. This is because such segments denote conflicting actions for the same packet, which then raises questions about the network administrator's intent. Results from segment classification are used to generate a visualization called the 'Segment Table'. We will see them in the coming Chapter 5 on Data Visualization.

Rule Based Approach

Anomaly detection can also be performed without segmentation. In order to do this, perform set operations between every rule pair in the rule set, checking if the rules are distinct, overlap or contained.

```
Input: A set of rules R.
Output: A set of rule anomalies annotated as follows:
        A1: redundant rules
        A2: shadow rules
        A3: overlapping rules
        A4: generalized rules
        A5: correlated rules
for Rule x ∈ R do
    for Rule y ∈ R do
        if (x.bdd = y.bdd)
            if (x.action = y.action)
                A1.add (x, y);
                continue;
            else
                A2.add (x, y);
                continue;
        xNegatey ⟵ x.bdd ^ ~ y.bdd;
        yNegatex ⟵ y.bdd ^ ~ x.bdd;
        xAndy⟵ x ^ y;
        if (yNegatex = 0)
            if (x.action = y.action)
                A1.add (x, y);
            else
                A2.add (x, y);
        else if (xNegatey = 0)
            if (x.action = y.action)
                A1.add (x, y);
            else
                A4.add (x, y);
        else if (xAndy != 0)
            if (x.action = y.action)
                A3.add (x, y)
            else
                A5.add (x, y);
```

Algorithm 3: Anomaly detection without segmentation.

Algorithm 3 describes how to detect anomalies without using segmentation. The worst case performance of this algorithm is $O(n^2)$, where n is the number of rules, because it performs comparison between every pair of rule in the rule set. For each rule, algorithm 3 performs a set operation checking if this rule is a subset, superset or overlaps another rule. Then depending on whether the rule actions match or not, rules are added to five sets, one for each anomaly discussed in the first chapter. Similar approaches have been mentioned in [18] and [19] but this algorithm is different from those in that it considers five anomalies and uses a BDD for efficiency. Results from algorithm 3 are visualized as a 'Rule Table'.

Chapter 4

ANOMALY RESOLUTION

There has been much work around anomaly detection in firewall rule sets but not much has been done in resolving the anomalies. In the past [18] and [24] have made good attempts to help resolve anomalies. Al-Shaer et al. [18] present an idea to assist network administrators in injecting new rules in the firewall and placing them appropriately so as to avoid redundancy and shadowing. Hu et al. [24] describe a resolution approach that considers network security assessment data from Nessus [43] and using manual support from the user decides upon ideal placement of rules. Their method [24, Figure 5] describes 'resolution strategy selection' as a process that makes decision based on risk level of conflicts, strategy repository and rule conflict information. Their solution is about changing the order of rules so that certain action constraints can be specified.

Here we present a novel approach to conflict resolution. The idea is to modify the rule itself to resolve the anomaly. To understand how this works, let's go back a little and try to understand how a rule is created. To create a rule we need specific data which includes action, ports and addresses. These are the minimal requirements for creating but some extra information may be required depending on the firewall vendor. For the analysis here, this information is not directly available. When performing the BDD set operations, some of the information is lost because the BDDs are reduced. This information loss is a result of fast BDD operations. Let us see how we can convert a BDD expression back to its Boolean expression.

Reverse Engineering BDD

Reversing a BDD to get the Boolean expression is about tracing all paths from the root to the decision nodes. What is obtained after such a process is a series of binary expressions that can then be converted to appropriate information depending on the requirements. For example, consider the BDD in figure 7. Tracing a path from the root to leaf node '1' generates $\sim v_1 \wedge v_2 \wedge v_3 \wedge \sim v_4 \wedge v_5$. Substituting 0 for false and 1 for true, we get 01101. If v_1 is the MSB then binary 01101 translates to 13 in decimal.

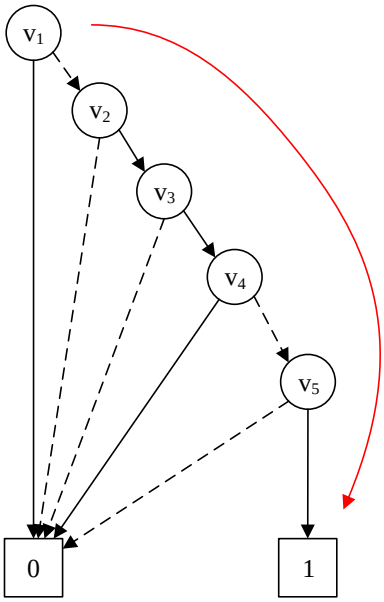

Figure 7. Reversing a BDD

Let us now see a more complex example of reverse engineering a BDD. A path to node '1' in figure 8 (a) translates to $\sim v_1 \wedge v_2 \wedge v_5$. We are missing variables v_3 and v_4 here. So how does this translate to binary form? Every missing variable denotes a '*don't care*' condition, and we denote it by a '-' (hyphen) in the binary form. The binary representation of Figure 8 is 01- -1. Now, substitute 0 and 1 in place of the dash in all

possible ways to get 01001, 01011, 01101 and 01111. The decimal representation is 9, 11, 13 and 15 respectively.

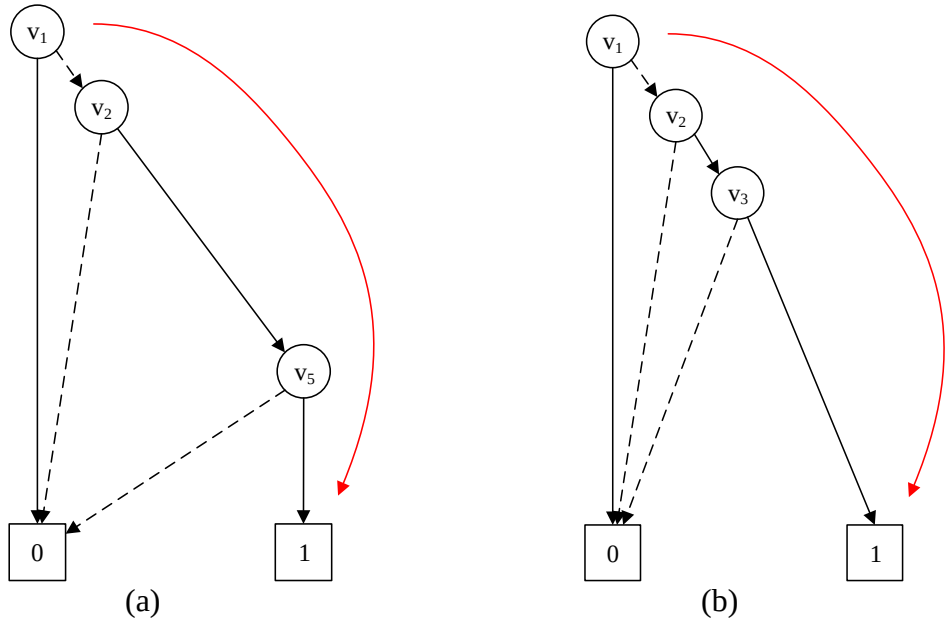

Figure 8. BDD with missing variables.

BDDs like the one shown in figure 8(a) can be used to denote non-consecutive integers, such as port numbers. Figure 8(b) shows a BDD denoting a range of consecutive integers. The path from root to '1' node generates $\sim v_1 \wedge v_2 \wedge v_3$. A binary expression involving five Boolean variables is 011- -, which gives us 01100, 01101, 01110 and 01111. These are the integers 12 to 15, therefore it represents the condition <=15. In a similar fashion it is possible to decode BDDs that represent greater than and not equal to conditions. This is all that is needed to decode BDDs and translate them back to firewall rule. Next section describes 'split-n-merge', a strategy used to remove anomalies in the firewall rule set.

Split-n-Merge

The split-n-merge strategy works by splitting rules, removing anomalous ones and then merging the result. Splitting is carried out during various set operations in Algorithm 2. Instead of discarding the results, we can use them to create anomaly free rules. There are various stages where splitting occurs. The first stage is where algorithm 2 performs set subtraction. The result of this stage can be used to remove generalization, correlation and overlapping anomalies. Let us see an example of each.

Resolving Generalization

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80	10.1.2.23	any
2	permit	tcp	192.168.1.2	any	10.1.2.23	any

Table 6: Resolving non-conflicting generalization anomaly.

In table 6, rule 1 is generalization of rule 2. Because the first two rules have the same action and R_1 is subset of R_2, it is safe to remove R_1. R_2 alone takes care of packets belonging to R_1 and therefore removing R_1 from the firewall rule set preserves its behavior. Now consider a different case of conflicting rules as shown in table 7.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80	10.1.2.23	any
2	deny	tcp	192.168.1.2	any	10.1.2.23	any

Table 7: Resolving conflicting generalization anomaly.

Resolving generalization in table 7 depends on the default behavior of the firewall. If the firewall default is ‘deny’ then R_2 can be removed in order to get rid of the anomaly. However if the firewall does not default to ‘deny’ then we have to consider splitting R_2.

Subtracting the packet space of R_1 from R_2 gives two source port ranges: 0-79 and 81-65535. The rule set size now increases to 3 but it is free from anomaly. The side effect is that an increase in rule set size affects the efficiency of the firewall. It depends on the administrator to decide which option they want. Do they need an anomaly free firewall with more rules or can they compromise on the anomaly with fewer rules that are perhaps easier to understand.

Resolving Correlation

A correlation anomaly can also be resolved by the split-n-merge strategy. Consider the rules in table 8. R_1 is correlated with R_2 because they overlap on source ports 85-90 (inclusive). To resolve this anomaly, we first perform a split. Using a BDD, perform the intersection between R_1 and R_2. Result of this operation gives a BDD that translates to:

$R_1 \cap R_2$ = *tcp 192.168.1.2 85-90 10.1.2.23 any*

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80-90	10.1.2.23	any
2	deny	tcp	192.168.1.2	85-100	10.1.2.23	any

Table 8: Correlation Anomaly Resolution

This is a conflicting segment. We resolve the conflicting action based on rule priority. Because R_1 takes precedence over R_2, $R_1 \cap R_2$ has the action from R_1. This gives firewall rule:

$R_1 \cap R_2$ = permit *tcp 192.168.1.2 85-90 10.1.2.23 any*

We now perform two more set operations. The first operation is of subtracting R_2 from R_1, which yields:

$R_1 - R_2$ = *permit tcp 192.168.1.2 80-84 10.1.2.23 any*

Next we perform subtraction of R1 from R2 which gives:

$R_2 - R_1$ = *deny tcp 192.168.1.2 91-100 10.1.2.23 any*

The final step is merging. Since source ports between ($R_1 \cap R_2$) and ($R_1 - R_2$) are consecutive and both have same action, merging can be performed.

R_{merged} = *permit tcp 192.168.1.2 80-90 10.1.2.23 any*

After replacing R_1 and R_2 we get:

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
merged	permit	tcp	192.168.1.2	80-90	10.1.2.23	any
R_2-R_1	deny	tcp	192.168.1.2	91-100	10.1.2.23	any

These rules preserve firewall behavior and are free from any anomaly. Again, the benefit of applying this solution depends on the type of firewall. If every accept and deny action has to be specified explicitly by the user, this is a good solution. But if the firewall defaults to 'deny' then only R_2 in table 8 must be deleted.

Resolving Overlapping

An overlapping anomaly occurs between two rules when they have same action on an overlapped packet space. Resolving overlapping is easier than dealing with conflicting rules. Consider a simple example from table 9. To get rid of overlapping between rules 1 and 2, merge continuous/overlapping segments together. R_1 and R_2 overlap on source port and merging them together gives: *permit tcp 192.168.1.2 80-100 10.1.2.23 any*

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
1	permit	tcp	192.168.1.2	80-90	10.1.2.23	any
2	permit	tcp	192.168.1.2	85-100	10.1.2.23	any

Table 9: Resolving Overlapping Anomaly.

Both shadowing and redundancy anomalies can always be resolved by removing the rule with lower precedence. Because in both cases the lower precedence rule is never executed, removing it from the rule set does not affect the traffic. Techniques described to remove anomalies can be applied through algorithm 4.

```
Input: Firewall rule set F, set of anomalies R,S,G,C,O denoting
redundancy, shadowing, generalization, correlation and overlapping.
Output: Modified firewall F without anomalous rules.
for (Rule r: F)
   F.remove (r.redundantRules);
   F.remove (r.shadowRules);
   for ( Rule g: r.generalizedRules)
      if (r.action = permit && g.action = deny)
         F.remove(g);
      else if (r.action = g.action)
         F.remove (r);
      else
         notify user and resolve manually;
   for (Rule o: r.overlappingRules)
      r ⟵ merge(r, o);
      F.remove(o);
   for (Rule c: correlatedRules)
      r1 ⟵ createRule (action(r), segment (r-c));
      r2 ⟵ createRule (action(r), segment (r ∩ c));
      r3 ⟵ createRule (action(c), segment (c-r));
      r ⟵ r1;
      c ⟵ r3;
      F.add (r2);
return F;
```

Algorithm 4: Anomaly removal.

Algorithm 4 begins with a set of anomalous rules as the input. An anomaly set consists of rule identifiers that point to a firewall rule. For example if firewall F has rules $\{r_1, r_2, r_3\}$ such that r_2 is shadow of r_1 then the shadow anomaly set of r_1 can be found by calling function *shadowRule*. In this case r_1*.shadowRule* returns the set $\{r_2\}$. Getter functions are

defined for other anomalies in a similar fashion. Removal of redundant and shadow rules is straightforward. Once again, this algorithm is assumed to work on a firewall that has a default policy of dropping a packet when no rule applies to it. Since redundant and shadow rules are never executed, removing them does not change the firewall's behavior. Overlapping rules can be merged together and this operation can benefit from BDD subtraction and intersection operations. Resolving generalization is easier if rules are non-conflicting, however conflicting rules may require manual intervention. Correlated rules are resolved by creating three new rules r1, r2 and r3. Rules r1 and r2 can be used to modify existing firewall rules and r3 is inserted.

We have presented various situations that give rise to anomalies and have explained how to tackle each of them. In the next chapter we will present results of our visualization techniques.

Chapter 5

DATA VISUALIZATION

It is well said that "a picture is worth a thousand words". Huge data sets, no matter how simplified, are sometimes hard to comprehend. It becomes a challenging task to extract meaningful information in a short period of time. This is where descriptive statistics come to the rescue. According to [44] "[Data visualization] involves the creation and study of visual representation of data". Visualization began with simple graphs and pie charts but with the advent of computer graphics and programming it has become more complex and is now capable of representing multidimensional data. The results obtained from our algorithms are complex and do not fit into the traditional matrix and tree based representations such as sunburst. In this chapter we present some new pictorial representations of a firewall rule set anomaly data. We will also cite some very interesting visualizations that have been presented by researchers earlier and we will reason about why they are not sufficient in conveying the right information to the target audience.

Previous Visualizations

One of the early visualization tools for showing firewall rules was 'Fang', developed by Mayer et al. [1]. Their user interface was very simple, and void of any graphics. Fang makes it easier for an administrator to view the rules but they are shown in a simple text format. In this form, the analysis of large rule sets is very difficult. The Lumeta firewall analyzer [3] was an improvement over Fang. It uses HTML pages to show evaluation

results and makes use of some graphics to represent certain information such as the action of a rule, type of source and destination address (host/network/any). The web page report tables heavily rely on a text description of the rule and therefore this tool too shares the same drawbacks of Fang.

A closely related work in visualizing firewall rule set anomalies is the Firewall Policy Advisor Tools [18]. Al-Shaer and Hamed developed a tool to display simplified versions of complex firewall rules. Their tool is capable of showing anomaly information in a tabular format. This tool also lacks use of data visualization techniques and displays rules in text format.

PolicyVis [39] was designed to facilitate rule anomaly analysis. Tran et al. developed a novel visualization technique that uses a row-column overlapping bars to represent the type of anomaly. Bars are color coded, green for an accept rule and red for deny. Partially overlapping bars represent a correlation anomaly, and a complete overlap represents shadowing or redundancy. Generalization is represented by placing a small bar inside another shaded bar. PolicyVis provides several options for administrators to zoom into the details of a rule. It allows a change of scope based on ports and addresses. However, the anomalies can be viewed only when a certain scope is defined.

Another recent work in visualizing firewall anomalies is by Hu et al. [24] [25] with the tool FAME. They used a matrix table to show conflicting and non-conflicting segments. A segment table is a novel way to represent anomalies but such a matrix representation is not sufficient to display exactly the fields where these rules overlap with each other. A segment table approach also fails to scale with a larger set of firewall rules. For example,

a firewall with over 200 rules and 500 segments creates 200x500 sized table which is hard to comprehend. The segment table is also not a good representation for showing different anomalies such as generalization and correlation.

Mansmann et al. [45] used a sunburst visualization to represent firewall rules. Their objective is not to display anomaly information but only to visualize the rule set. The sunburst visualization technique basically a radial representation of a tree and it is therefore not suitable for plotting anomalies which can resemble a graph. Sunburst can only represent a subset or superset data but cannot represent overlapping fields which are the essence of firewall rule anomalies.

We have seen that results from rule set analysis are complex and inter-dependent and therefore traditional data representations such as a matrix table, tree diagrams and sunbursts are not appropriate to show the summary such analysis. We will now describe some data visualization approaches that are suitable for showing rule anomalies.

Segment Table

Figure 9 shows a segment table which is a matrix representation of rules and segments. Each row represents a rule and every column denotes a segment. Segments and rules are color coded for better understanding. If an input data set of *n* rules generates *m* segments, then the segment table has dimensions *n* x *m*. Each shaded block for n^{th} row and m^{th} column denotes that nth rule belongs to segment at position m. Blocks are shaded green if the rule has a permit action and red if it is a deny action. The column headers show the segment index. A segment header can have three colors, each denoting if this is a conflicting segment, non-conflicting segment or a non-overlapping segment. The

segment table that we have described is very similar to the one mentioned in [24]. The design has been changed to accommodate more number of rules in a small space. Increasing the density of rows and columns gives a unified feel and makes it easier to understand.

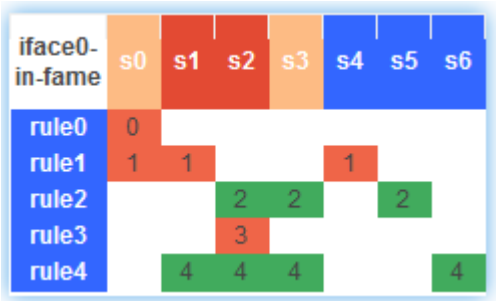

Figure 9. Segment table.

One drawback of a segment table based approach is that it does not scale well. For a large data set with many anomalies, the size of segment table becomes huge and understanding it becomes a tedious task.

A good solution for this problem is to partition the segment table into a set of smaller tables. The table canoe be partitioned from any point. The partition has to preserve the anomaly relationship between the rules so that no rule appears in more than one segment table and every rule is taken into account. Hu et al. [25] have defined a ‘correlation group’ as a group of segments where a rule belongs to exactly one correlation group. Here a new approach is presented to generate these correlation groups and for the sake of simplicity and distinctiveness, it is called a ‘community’. Algorithm 5 presents details of how communities are built. Each community can be presented in its own segment table.

Implementing the community generation algorithm 3 gives a set of smaller segment tables but then another problem arises. Some segments tables are too small. Consider for example, segment table shown in figure 10. Running algorithm 5 on this segment table generates 3 small segment tables, which by themselves are very small and many of these smaller tables clutter the visualization. These smaller segment tables are called a "micro community".

```
Input: set of segments S
Output: set of communities
G ⟵ new set of communities;
for (Segment s ∈ S)
    flag ⟵ false;
    c ⟵ new Community;
    R ⟵ set of rules belonging to s;
    if (G is empty)
        G.add (new Community with segment s );
    else
        for (Community g ∈ G)
            R2 ⟵ set of rules belonging to g;
            R3 ⟵ R\R2;
            if (R3 is not empty)
                if (flag is not set)
                    g.add (s);
                    c ⟵ g;
                    flag ⟵ true;
                else
                    for (Segment x ∈ g)
                        c.add (x);
                    G.remove(g);
            else if (g is last element in G)
                G.add (new Community with segment s);
                break;
return G;
```

Algorithm 5. Community generation from segment set.

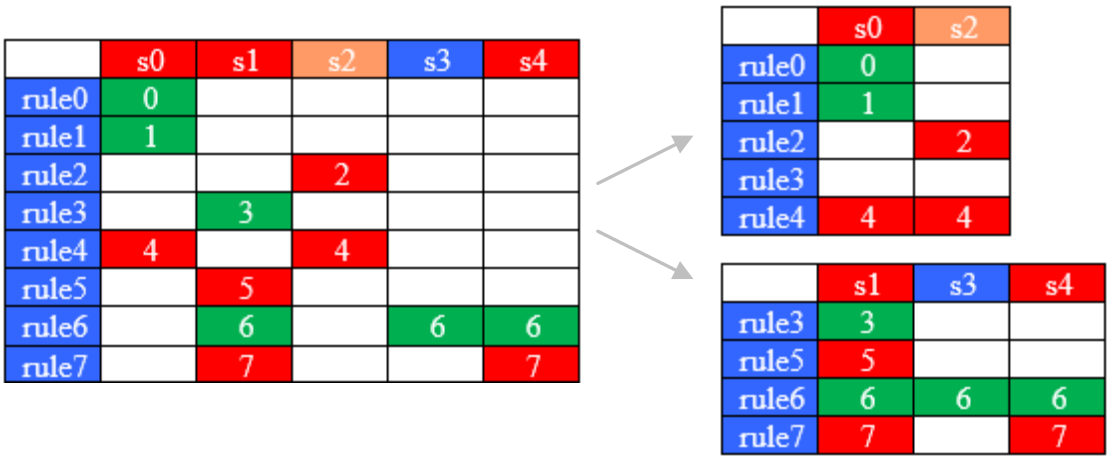

Figure 10. Community generation from segment table.

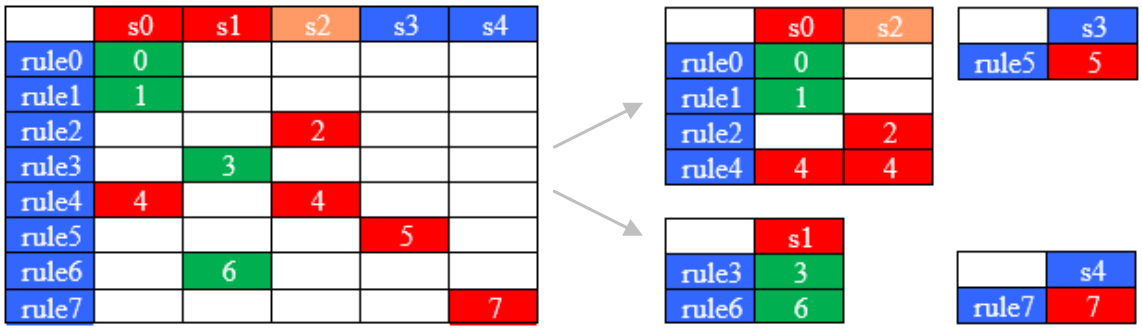

Figure 11. Micro-communities.

Micro communities are an unwanted byproduct of the community generation algorithm. There is a way to correct this problem. If small communities are merged together, a good sized community is created. Micro community removal is described in Algorithm 6. The algorithm requires an input threshold value denoting the desired size of community. An example of running the micro-community removal algorithm is shown in Figure 12. Merging is performed with a threshold value 3. Segments s3, s4 and s1 are merged together to conform to the threshold size.

```
Input: set of communities G
Output: set of communities with merged micro-communities
Sort G in ascending order of community size;
G′⟵ new set of communities;
for (i = 0; i < G.size; i++)
    if (G[i].size ≥ t)
        add G[i] and remaining elements to G′;
        return G′;
    if (i = G.size-1)
        add G[i] to last element in G′;
        return G′;
    merge G[i] with G[i+1];
    i⟵ i+1;
    g⟵ G[i];
    n⟵ g.size;
    while (n < t)
        if (i < G.size)
            g′⟵ G[++i];
            merge g with g′;
            g⟵ g′;
            n⟵ g′.size;
        else
            G′.add(G[i]);
            return G′;
    G.add(g);
    i⟵ i+1;
return G′;
```

Algorithm 6: Micro-community removal.

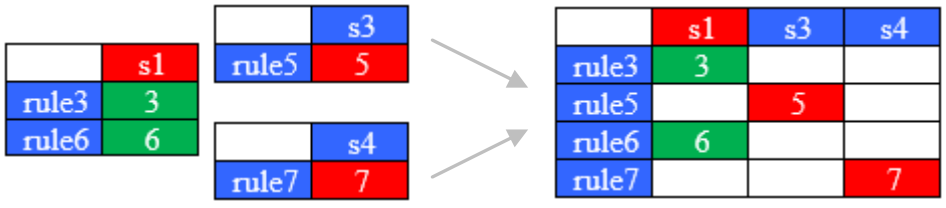

Figure 12. Merging micro-communities

We have seen how to deal with large segment tables, and partition them into smaller, more manageable data sets. Segment tables are important in understanding the overall

state of firewall however they miss something: one cannot tell exactly which portion of packet space of a rule is conflicting with another rule. All that is observed is the presence of a conflict but it does not give a clear idea of how the conflict is actually happening. To view this information, a new visualization called 'Rule table' is presented in the next section.

Rule Table

The benefits of segment table have been explained as well as some side effects of the segment table partitioning algorithm. The segment table is a good approach to convey high level information about firewall anomalies. However, it misses lower level information such as details of firewall fields. To represent this low level information, a new visualization called Rule Table is designed as given in Figure 13. The same input set from [25, Table1] is used and shown in Table 10. Each row represents a firewall rule and columns denote different fields of the rule.

Rule#	Action	Protocol	Src Address	Src Port	Dst Address	Dst Port
0	deny	udp	10.1.2.0/24	any	173.32.1.0/24	53
1	deny	udp	10.1.0.0/24	any	173.32.1.0/24	53
2	permit	tcp	10.1.0.0/24	any	192.168.0.0/24	25
3	deny	tcp	10.1.1.0/24	any	192.168.1.0/24	25
4	permit	any	10.1.1.0/24	any	any	any

Table 10. Sample firewall rule set

The rule table helps in zooming into details of each rule. The overlapping of rules is very easily detected through this representation. The table looks simple and minimal although it conveys detailed information. We now go into the dynamics of creating rule table and how to decode it visually to extract information. When designing the rule table, the

objective was to present more information with fewer numbers, to help network administrators.

A rule table consists of five columns, showing the protocol, source address/port and destination address/port. Each shaded box inside the table is used to denote the range on the input dataset that this particular firewall rule acts upon. Consider for example rule5 from table 10. Because the protocol is ‘any’ the corresponding box in the table is shaded completely. Similarly, the protocol box for rule 1 is partially shaded because UDP is only one part of the protocol spectrum.

Figure 13. Rule table.

The bar size of each field is dynamically generated and depends on diversity of the input data set. For example, the protocol field size for the input data set is 3 because there are three distinct values any, tcp and udp. It makes more sense to allocated one third of the available box space. There are more than a hundred protocols and allocating space for each of them even if it is not mentioned in the rule is a waste of box space and introduces too many voids.

The IP/network address field follows the same approach. The source address set contains three distinctive addresses, 10.1.2.*, 10.1.*.* and 10.1.1.*. Since 10.1.1.* is the superset of all other source addresses, it occupies the entire box space. One third of the space is

occupied by 10.1.2.* network and other one third by the 10.1.1.* network. Source address bars are positioned based on how they are related to other rules. Since the 10.1.1.* network comes before the 10.1.2.* network, its address bar is positioned to the left of 10.1.2.*.

Representing the port numbers follows a slightly different approach. Because the port numbers can be specified as a range (<80, 22-80) and also as a fixed value (=443), and the fact that this range extends from 0 to 65535, the visualization results using the previously defined approach don't fit properly into the small space.

As a solution, the density of ports in the box is varied. Since most service ports fall in the range 1 to 300, 90% of the available box space is utilized to represent values from this range. The remaining 10% of the box space packs port numbers greater than 300. By varying the data density, the visualization results look good.

As with the segment table, rule tables also suffer from the scalability issue. With more than a thousand rules, the table has the same number of rows. To solve this problem, the community generation approach can be applied to the rule table. Generate the segment tables and then draw a rule table for all the rules in each segment table.

Hive Plot

Both the segment table and the rule table are not sufficient to understand the anomaly relationship between firewall rules. The segment table gives an idea of how good or bad the firewall is, and rule table helps understand the overlapping of the packet space. To view the relationship between rules with respect to the five anomalies defined previously, the results are visualized with a 'Hive Plot' as show in figure 14. Krzywinski et al. [46]

describe a novel approach to visualize networks and their visualization is extended to show anomalies.

Each axis in the plot represents a firewall rule. Figure 14 mentions five rules rule0-rule4 and the results are derived from table 10. Starting from the top central axis and moving in clockwise direction, are rule0 through rule4. The anomalies are represented as nodes on every axis. From inside to outside, every node represents an anomaly in the order shadow, generalization, correlation, redundancy and overlapping. A link between two nodes shows an anomaly between the rules that these linked nodes belong to. For example, rule4 is a generalization of rule3. This information is shown by drawing a link from the second node of rule3 axis to the second node on axis representing rule4.

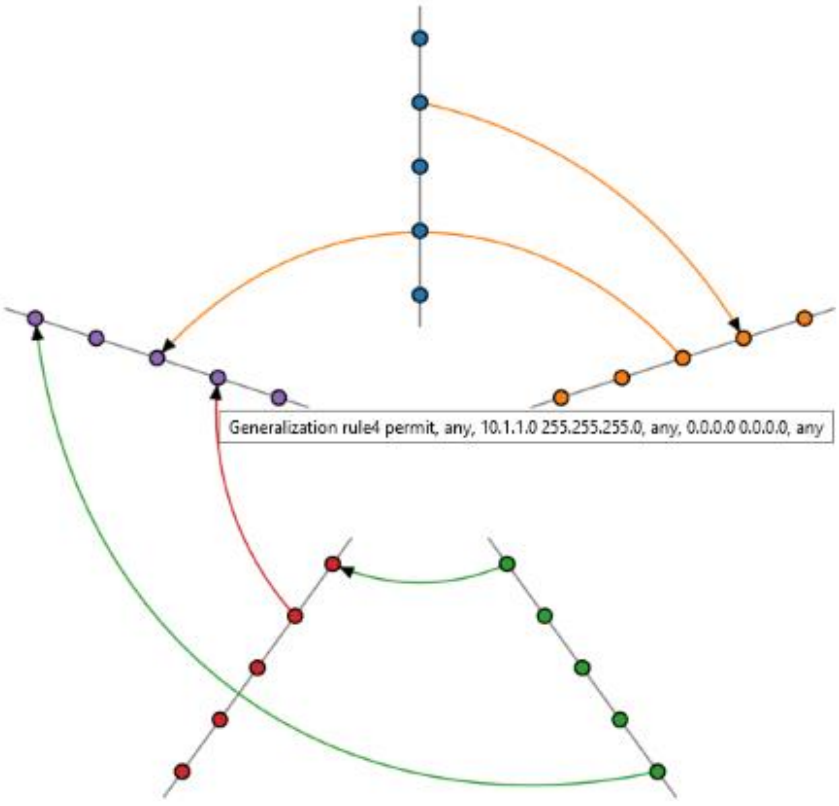

Figure 14. Hive plot.

A hive plot representation is very helpful in viewing how many anomalies exist in the firewall and what rules are involved. The distribution of axis in the hive plot is dynamic and the density increases with increase in the number of rules in the input rule set. Therefore, this approach does not scale well. When the number of rules exceeds 50, the hive plot becomes very dense and it is hard to understand the anomaly relationship. An example of densely populated hive plot is show in figure 15.

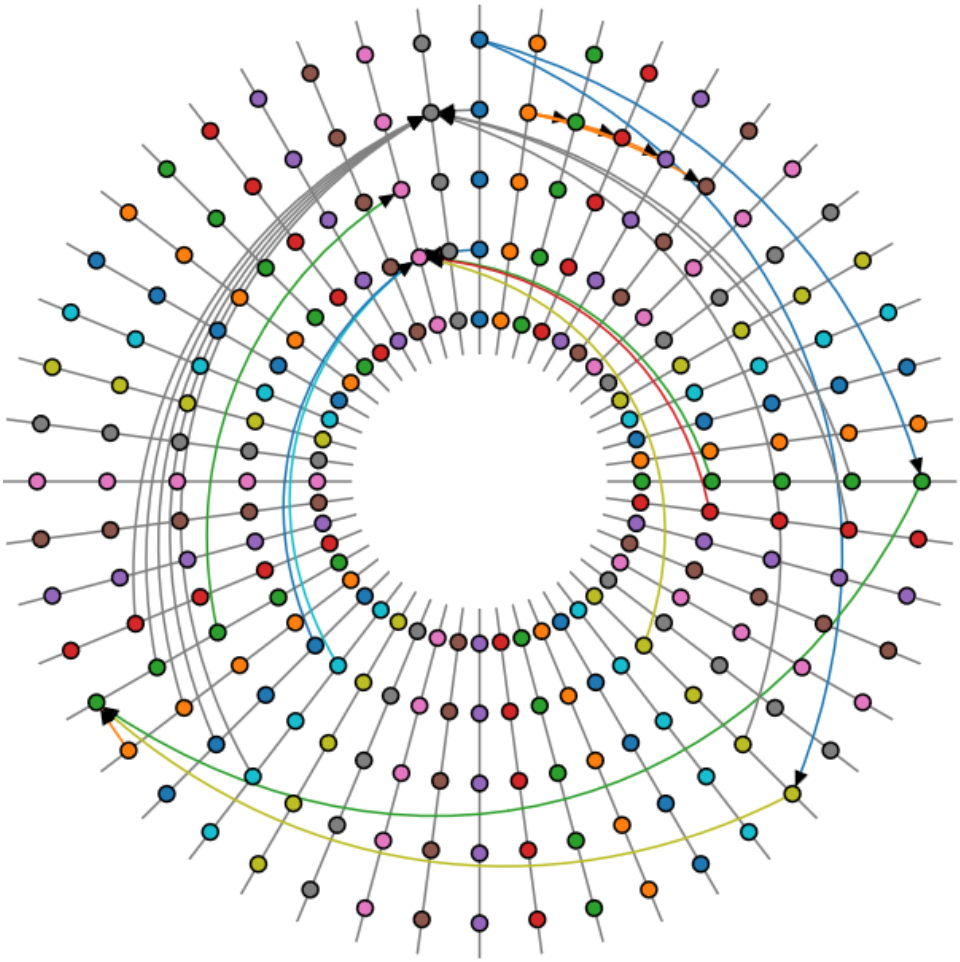

Figure 15. Densely populated hive plot.

Using Visualizations

We have seen three visualizations, a segment table, a rule table, and a hive plot. In order to make the most out of these graphical representations, this section describes some helpful tips.

For the segment table, the best way to tell if your firewall is efficient is by taking a look at the segment headers and looking for red colored segments. If all segment headers are blue, it means there are no overlapping rules in the firewall and therefore no anomalies. If a red colored segment exists it means conflicting overlapping rules exist and it is time to check the rule table.

Examine the hive plot and watch for any links. A link denotes involvement of two rules in an anomaly. Make a list of rules that have any outgoing and incoming links. Given this information the anomaly the rule is generating is known but the reason is not known. If there are no links, good news! No anomaly exists in the firewall.

In order to find the answer to ‘why this anomaly?’ The rule table must be interpreted. Take the list of rules prepared previously from the hive plot and compare the rule fields in the rule table. It is easy to find if two rules are overlapping and where they are overlapping.

Chapter 6

PEFORMANCE EVALUATION

This section evaluates performance of the tool against several rule sets. The firewall configuration data was collected from real world sources of varying size, from a small computer lab (less than 10 hosts) to a big corporation (more than 1000 hosts). Anomaly detection and visualization generation tests were carried out on computer with 2.4GHz Intel i5 CPU and 4GB memory and the data set belonged to Cisco ASA firewall.

Figures 16 and 17 show performance of our tool for preparing initial data and anomaly detection respectively. X-axis represents the number of rules that were parsed from the firewall configuration file. Time (in milliseconds) is represented on the y-axis. After rules are parsed, they are converted to BDD and tool's performance is shown with a green bar.

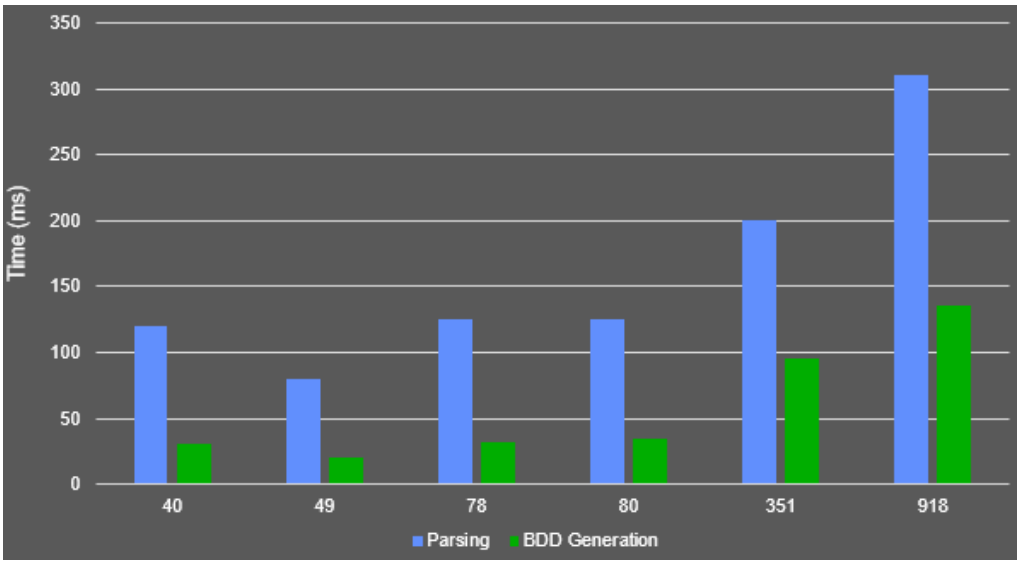

Figure 16. Parser and BDD generation performance.

In Figure 17, x-axis represents two values in the format A/B. A is the number of rules and B is the number of segments generated. Y-axis shows the time (in ms) it took for computing anomalies between all rules in the set. Interestingly, the processing time for 882 segments is lower than 320 segments. This is because anomalies are independent of the size of segment table. A smaller segment table can have multiple anomalies and a huge segment table can be completely void of anomalies. Segmentation takes almost no time if no anomalies are present in the rule set. The processing time increases with a rise in the number of segments.

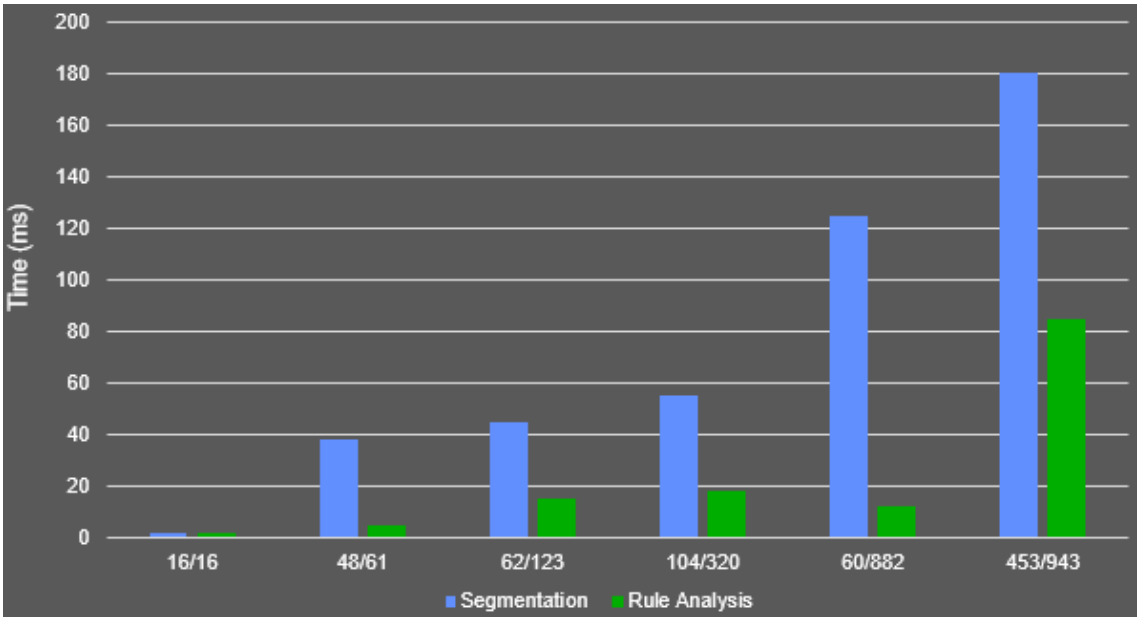

Figure 17. Segmentation and Anomaly detection performance

The data required to visualize the results is translated to JSON format and the performance is linear. Visualizations are created using a set of JavaScript files and an open source library d3js [47]. The program for firewall analysis is written in Java using Eclipse IDE. HTML, CSS3 and JavaScript are used for generating graphics.

Chapter 7

CONCLUSION AND FUTURE WORK

Firewall rule set analysis and its visualization is an important research area. Although much research has been done on the analysis of firewalls, there is much less on how to visualize such results in the most effective manner. This thesis describes strategy for anomaly detection and classifies anomalies into five categories. Various visual representations such as a segment table, a rule table, and a hive plot are used. Such pictorial representations are helpful in understanding of firewall rule sets and support revision of previous rules. They also help network administrators in determining where new rules can be inserted and which rules contribute to efficiency.

This research can be extended in different directions. One direction is to apply the anomaly analysis for Software Defined Networks (SDN). SDN primarily works on OpenFlow protocol and the syntax of rules and their behavior is different from traditional network firewalls. One can also extend this work by improving visualizations and designing new techniques for better representation of the data.

This research can also be extended towards resolution of anomalies. The existing algorithms on anomaly resolution work on re-ordering of rules and are not efficient. In place modification of rules needs to be studied. Also, appropriate tools to verify the firewall behavior need to be developed so that any modifications to the rules can be verified against proper functioning of the firewall. The anomaly analysis can also be extended to distributed firewalls.

BIBLIOGRAPHY

[1] A. Mayer, A. Wool and E. Ziskind. “Fang: A Firewall Analysis Engine.” *Proc. IEEE Symposium on Security and Privacy*, pages 177-187, 2000.

[2] Y. Bartal., A. Mayer, K. Nissim and A. Wool. “Firmato: A Novel Firewall Management Toolkit.” *Proc. IEEE Symposium on Security and Privacy*, pages 17-31, 1999.

[3] A. Wool. “Architecting the Lumeta Firewall Analyzer.” *Proc. 10th USENIX Security Symposium*, pages 85-97, 2001.

[4] E. Lupu and M. Sloman. “Conflict Analysis for Management Policies.” *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM’1997)*, pages 430-443, 1997.

[5] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine and C. Xu. “IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution.” *Policies for Distributed Systems and Networks*, pages 35-96, 2001.

[6] D. Eppstein and S. Muthukrishnan. “Internet Packet Filter Management and Rectangle Geometry.” *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 827-835, 2001.

[7] B. Hari, S. Suri and G. Parulkar. “Detecting and Resolving Packet Filter Conflicts.” *Proc. IEEE INFOCOM’00*, pages 1203-1212, 2000.

[8] J. L. Bentley. “Multidimensional binary search trees used for associative searching.” *Communications of the ACM*, 18(9), pages 509–517, 1975.

[9] S. Hazelhusrt. “Algorithms for Analyzing Firewall and Router Access Lists.” *Technical Report TR-WitsCS-1999*, Department of Computer Science, University of the Witwatersrand, South Africa, 1999.

[10] V. Srinivasan, S. Suri and G. Varghese. “Packet Classification Using Tuple Space Search.” *Computer ACM SIGCOMM Communication Review*, pages 135-146, 1999.

[11] T. Woo. “A Modular Approach to Packet Classification: Algorithms and Results.” *Proc. IEEE INFOCOM’00,* pages 1213-1222, 2000.

[12] L. Qiu, G. Varghese, and S. Suri. “Fast Firewall Implementations for Software and Hardware-based Routers.” *Proc. 9th International Conference on Network Protocols (ICNP'2001)*, pages 241-250, 2001.

[13] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. “Router Plugins: A Software Architecture for Next-Generation Routers.” *Proc. ACM Sigcomm*, pages 919-202, 1998.

[14] P. Gupta and N. McKeown. “Packet Classification on Multiple Fields.” *Proc. ACM Sigcomm*, pages 147-160, 1999.

[15] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. “Fast and Scalable layer for Switching.” *Proc. ACM Sigcomm*, pages 191-202, 1998.

[17] D. Stiliadis and T.V. Lakshman. “High-Speed Policy-Based Packet Forwarding using efficient Multi-Dimensional Range Matching.” *Proc. ACM Sigcomm*, pages 203-214, 1998.

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

[19] E. Al-Shaer and H. Hamed. “Discovery of Policy Anomalies in Distributed Firewalls.” *Proc. IEEE Infocomm*, Vol. 4, pages 2605-2616, 2004.

[20] L. Yuan, J. Mai, Z. Su, H. Chen, C.N. Chuah and Mohapatra P. “FIREMAN: A toolkit for FIREwall Modelling and Analysis.” *Proc.2006 IEEE Symposium on Security and Privacy*, 2006.

[21] J. Govaerts, A. Bandara and Kevin Curran. “A formal logic approach to firewall packet filtering analysis and generation.” *Artificial Intelligence Review* 29.3-4, pages 223-248, 2008.

[22] R. Oliveira, S. Lee, and H. Kim. “Automatic detection of firewall misconfigurations using firewall and network routing policies.” *IEEE Dependable Systems & Networks Workshop on PFARM*, 2009.

[23] T. Nelson, C. Barratt, D. Dougherty, K. Fisler, S. Krishnamurthi. “The Margrave Tool for Firewall Analysis.” *LISA*, 2010.

[24] H. Hu, G. Ahn, and K. Kulkarni, “Detecting and resolving firewall policy anomalies.” *IEEE Transactions on Dependable and Secure Computing*, pages 318-331, 2012.

[25] H. Hu, G. Ahn, and K. Kulkarni, “FAME: A Firewall Anomaly Management Environment.” *Proc. 3rd ACM workshop on Assurable and Usable Security Configuration*, pages 17-26, 2010.

[26] Cisco Secure Policy Manager 2.3 Data Sheet. [Online]. Available: http://www.cisco.com/warp/public/cc/pd/sqsw/sqppmn/prodlit/spmgr_ds.pdf

[27] R. Becker, S. Eick, and A. Wilks. “Visualizing network data.” *IEEE Transactions on Visualization and Computer Graphics*, 1(1), pages 16-21, 1995.

[28] E. Bertini, P. Hertzog, and D. Lalanne. “SpiralView: towards security policies assessment through visual correlation of network resources with evolution of alarms.” *IEEE Symposium on Visual Analytics Science and Technology*, pages 139-146, 2007.

[29] J. Yang, M. Ward, and E. Rundensteiner. “Interring: An interactive tool for visually navigating and manipulating hierarchical structures.” *IEEE Symposium on Information Visualization (INFOVIS)*, pages 77-84, 2002.

[30] S. Foresti, J. Agutter, Y. Livnat, S. Moon, and R. Erbacher. “Visual correlation of network alerts.” *IEEE Computer Graphics and Applications*, pages 48-59, 2006.

[31] J. Goodall, W. Lutters, P. Rheingans, and A. Komlodi. “Preserving the big picture: Visual Network traffic analysis with tnv.” *IEEE Workshop on Visualization for Computer Security (VizSEC’05)*, pages 47-54, 2005.

[32] D. A. Keim, J. Schneidewind, and M. Sips. “Fp-viz: Visual frequent pattern mining.” *In Proc. IEEE Symposium on Information Visualization (InfoVis '05)*, 2005.

[33] H. Koike and K. Ohno. “SnortView: visualization system of snort logs.” *In Proc. ACM workshop on Visualization and data mining for computer security*, pages 143-147, 2004.

[34] C. Lee, J. Trost, N. Gibbs, R. Beyah, and J. Copeland. “Visual Firewall: real-time network security monitor.” *IEEE Workshop on Visualization for Computer Security*, pages 129-136, 2005.

[35] F. Mansmann, D. A. Keim, S. C. North, B. Rexroad, and D. Sheleheda. “Visual Analysis of Network Traffic for Resource Planning, Interactive Monitoring, and Interpretation of Security Threats.” *IEEE Transactions on Visualization and Computer Graphics*, 13(6), pages 1105-1112, 2007.

[36] J. McPherson, K. Ma, P. Krystosk, T. Bartoletti, and M. Christensen. “Portvis: a tool for port-based detection of security events.” *In Proc. ACM workshop on Visualization and data mining for computer security*, pages 73-81, 2004.

[37] S. Morrissey and G. Grinstein. “Visualizing firewall configurations using created voids.” *In 6th International Workshop on Visualization for Cyber Security (VizSec)*, pages 75-79, 2009.

[38] H. Shiravi, A. Shiravi, and A. Ghorbani. “A survey of visualization systems for network security.” *IEEE Transactions on Visualization and Computer Graphics*, pages 1313-1329, 2011.

[39] T. Tran, E. Al-Shaer, and R. Boutaba. “PolicyVis: firewall security policy visualization and inspection.” *In Proc. 21st conference on Large Installation System Administration Conference*, pages 1-16, 2007.

[40] R. Bryant. “Symbolic boolean manipulation with Ordered Binary-Decision Diagrams.” *ACM Computing Surveys*, 24(3), pages 293-318, 1992.

[41] L. Xiao, J. Gerth, and P. Hanrahan. “Enhancing visual analysis of network traffic using a knowledge representation.” *IEEE Symposium on Visual Analytics Science and Technology*, pages 107-114, 2006

[42] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald. “An evaluation of space-filling information visualizations for depicting hierarchical structures.” *International Journal of Human-Computer Studies*, 53(5), pages 663-694, 2000.

[43] “Tenable Network Security.” [online] Available: http://www.tenable.com/products/nessus

[44] “Data Visualization.” [online] Available: http://en.wikipedia.org/wiki/Data_visualization

[45] F. Mansmann, T. Göbel, and W. Cheswick. “Visual analysis of complex firewall configurations.” *Proc. 9th International Symposium on Visualization for Cyber Security*, ACM, pages 1-8, 2012.

[46] M. Krzywinski, I. Birol, S. Jones, and M. Marra. “Hive plots - rational approach to visualizing networks.” *Briefings in bioinformatics*, 13.5, pages 627-644, 2011.

[47] “Data Driven Documents.” [online] Available: http://d3js.org/

BIOGRAPHICAL SKETCH

Pankaj Kumar Khatkar was born in Haryana, India. He earned Bachelor of Engineering degree in Computer Engineering from University of Mumbai, India in 2010. In 2011 he joined graduate program in Computer Science at Arizona State University (ASU). While pursuing his degree at ASU he worked as a research associate at the Secure Networking And Computing (SNAC) Lab under guidance of Dr. Dijiang Huang. His research interests are primarily in cyber security with focus on network security. Other research areas like Software Defined Network, natural language processing and artificial intelligence also catch his attention. During his program at ASU he held teaching associate position for more than a year during which he taught Java programming and network security to both graduate and undergraduate students. His skills have been utilized for various technical articles and a journal paper.