Security policy rules optimization and its application to the Iptables firewall

Stanislav Y. Radomskiy

University of Tampere
School of Information Sciences
Computer Science M.Sc. thesis
Supervisor: Jyrki Nummenmaa
April 2011

University of Tampere
School of Information Sciences
Computer Science
Stanislav Y. Radomskiy: Security policy rules optimization and its application to the Iptables firewall
M.Sc. thesis, 57 pages
April 2011

The security of a network which is being connected to the Internet is of the highest importance especially if this network belongs to a private company or a bank. Many different methods have been invented and are currently used in order to prevent harm that can be caused by perpetrators to the local network. One of the most commonly used methods is a firewall.

The important feature of the firewall is the efficiency of its filtering. Since each packet is compared against filtering rules in the firewall, the firewall introduces a delay. The less the delay, the more efficient is the filtering.

In order to make network more secure, we have to make sure that firewall implements a given security policy and makes it efficiently. The ruleset should be examined for the presence of errors such as duplicated, shadowed and conflicting rules and those errors should be eliminated if found. Rules should be also rearranged to make filtering optimally efficient. Rule rearrangement could accidentally cause a change in the security policy and we have to make sure that it will not happen.

We study two different optimization approaches that can be applied to the ruleset of the firewall. The optimization approach that examines the ruleset and eliminates the errors is called static optimization since it uses a given ruleset and does not need any additional data. Another optimization approach that rearranges the rules' order in the ruleset is called dynamic optimization since it uses dynamic statistical information of the traffic which flows through the firewall.

In this work we consider different existing optimization approaches, both static as well as dynamic, and suggest a combination of their best features and improvements. We make sure that the suggested optimizations will not change the security policy of the initial ruleset at any time. We also consider peculiarities of the Iptables firewall and show how the suggested method can be applied to its ruleset.

Contents

1. Introduction

When the Internet and TCP/IP protocols were created nobody was concerned about security, yet nowadays, when the Internet has become a global system everyone can easily gain access to, the amount of threats has increased dramatically. If no security measures are taken, the local network and private data of any company or bank can easily be accessed by perpetrators. This can result in valuable information being stolen, people's personal data viewed and denial of service (DOS) attacks used against the company or bank services.

In order to prevent the afore-mentioned from occurring, many different solutions have been found, ranging from anti-viruses and firewalls to IDS (Intrusion Detection Systems). Of these, firewalls are the most important and commonly used means to protect networks from threats coming from the Internet. They are often called ”core elements in network security” [1].

There are many different types of firewall. They can be classified in different ways: personal and corporate; software and hardware; packet filter, application level gateway and session level gateway. Although it is sometimes hard to say which type a particular firewall is as it can contain features of several different types, the most commonly used type is a packet filter. The security policy in a packet filter is implemented by using a table of filtering rules, a *ruleset*.

However, simply installing such a firewall is not enough to secure a network. Setting up and maintaining the firewall is a rather sophisticated task. Managing a firewall in a large network with 500 or more computers is extremely difficult. The number of filtering rules is considerable and it is almost inevitable that the administrator of this network will make an error. The scenario of when one administrator creates firewall rules and another one changes this ruleset afterwards is even more error-prone.

The majority, if not all packet filters follow the first-hit-principle [2]. Each rule in the ruleset has a number of parameters the types of which usually correspond to the headers of a network packet. Each packet, traversing through the firewall is compared against the filtering table of rules. The rule whose parameters have the same values as the parameters of the packet under comparison is applied to it. At this point the decision is usually made whether to drop the packet or let it go further. The rules succeeding the matched rule are not considered. If such a rule is not found, the *global rule* of the ruleset is applied to the packet. The global rule is usually checked last after comparison to all other rules of the ruleset.

Such firewall behaviour can cause a security breach if misconfigured by the administrator. A good example of an administrator error could be placing the rule whose parameters are a subset of an already existing rule after that rule. In this case the newly created rule will never be used and the firewall behaviour might differ from that anticipated by the administrator.

There are several reasons why the administrator would have to change the ruleset. For example, the security policy can be modified or traffic from a local infected computer which generates malicious network packets can be blocked until the problem is solved.

Another important point is the efficiency of firewall filtering. Because each packet is compared against filtering rules in the firewall, the firewall introduces a delay. The shorter the delay, the more efficient the filtering.

In order to make a network more secure, we have to make sure its firewall implements the security policy of the organization this network belongs to and does it efficiently. The ruleset should be examined for the presence of errors such as duplicated, shadowed and conflicting rules and those errors should be eliminated if found. Rules should be also rearranged to make filtering optimally efficient. It is not easy to rearrange the rules because of the precedence property of the ruleset coming from the the first-hit-principle of the packet filters.

Some work has been done in this field. Al-Shaer et al. [1,4,10] describe possible conflicts and dependencies between any two given rules and suggest an algorithm for searching such problems. At the same time, Chomsiri et al. [9] suggest the Raining 2D-Box Model approach to consider the case of several rules causing the problem. In addition to that, Gouda and Liu [8] cover a subset of problems described by Ehab Al-Shaer et al. by applying decision trees. Katic et al. [3] also extend Ehab Al-Shaer et al.'s work by merging some rules and considering specific features of the Iptables firewall such as log rules and limit parameters. The dynamic analysis of a ruleset has also been carried out with Acharya et al. [2], Al-Shaer et al. [6] and Fulp [7] giving an insight on how statistical characteristics of the firewall rules such as probability of their match can be used in a dynamic online ruleset optimization.

Katic et al. [3] and Acharya et al. [2] allege that their approaches are suitable for the Iptables firewall but neither an extensive study of rule syntax nor all possible rule errors were considered.

During our study of the afore-mentioned research in this area we discovered that it suggests dealing with the same problem in many different ways. All existing methods have good aspects as well as drawbacks. We have noticed that it would be beneficial to study all of these methods and combine their good features into one algorithm for static as well as dynamic optimization of the ruleset.

We have also noticed that previous research does not provide an extensive rule syntax or rules relationship study in connection with this optimization problem, security policy maintenance and the most commonly used firewall for the Linux platform, Iptables, and it would be beneficial to carry out this study and use it in the new optimization algorithm.

In this thesis we consider the most commonly used type of a firewall which is a packet filter. We concentrate on the corporate type of firewalls since they are more important in the sense that they secure the whole company's network and not just one particular computer. A further reason we decided to consider corporate firewalls is that they are much harder to set up and maintain than simple, yet not trivial, personal ones. A corporate firewall can be also called an internetwork since it connects two or more different networks and filters traffic going between them.

The particular packet filter which we focus on, Iptables, comes with all Linux distributions, and a Linux kernel usually contains a number of modules which can be loaded to extend the features of this firewall. Many private networks use Linux as an operating system for their gateways and it is a good place to have the firewall installed, since all traffic between local network and the Internet passes through the gateway, therefore Iptables is widely used and its optimization is an interesting topic to research.

In this work we analyze research which is related to ruleset optimization and its application to Iptables, analyze suggested optimization algorithms and emphasize their advantages and disadvantages. We try to combine their best features and create our own algorithm for error elimination and ruleset optimization. We study the peculiarities of Iptables firewalls and their rule syntax. We then show how to apply our new algorithm to rulesets.

We consider *static* as well as *dynamic* rules optimization. Static rules optimization does not need the knowledge of the statistical traffic properties whereas dynamic optimization takes them into consideration when optimizing a ruleset. Due to the limited scope of this work we concentrate mostly on static optimization methods and their improvement.

The thesis is organized as follows: Chapter 2 describes what static rules optimization is, considers and combines the best features of the existing approaches, suggests a way of considering more than two rules at a time when searching for conflicts between the rules and provides a way to keep security policy unchanged during the optimization process. Dynamic rules optimization is described in Chapter 3, where existing methods are analyzed and combined, and an improved version of one of these methods is suggested. Chapter 4 describes the Iptables firewall. Chapter 5 discusses the application of suggested static and dynamic optimization methods to the Iptables firewall. Chapter 6 talks about related problems that are beyond the scope of this work and considers future work. Finally, Chapter 7 concludes the thesis.

2. Static rules optimization

As we have already mentioned in the introduction, due to the limited time-frame and scope of this work, we mostly concentrate on the static rules optimization techniques and their application to the Iptables firewall.

In this chapter we consider static rules optimization methods which do not require knowledge of the statistical characteristics of traffic going through the firewall, such as rule matching probability. This means that we analyze possible relations between the filtering rules and recognize possible errors. The problem of rule relations and possible errors between them is considered in detail by Al-Shaer et al. [1,4,10] and Mohamed et al. [5]. Many other articles which analyze the rulesets have references to one of these works to describe the relations between the rules and possible errors. These works contain extensive analysis of all possible relations between rules and their fields. One of them by Mohamed et al. [5] concentrates on errors which can arise in distributed security systems consisting of several firewalls. In this work we concentrate on a single independent firewall solution.

It is possible to use any field of IP/TCP/UDP header to filter the traffic, but the following five are most often used in traffic filtering: protocol, source ip, destination ip, source port and destination port [4].

The security policy in a packet filter is implemented by using a table of filtering rules (see Table 2.1). We call this table a *ruleset*.

Rank	Incoming interface	Outgoing interface	Protocol	Source ip	Source port	Destination ip	Destination port	ICMP code	ICMP type	Action
1	0	1	tcp	192.168.1.1	1024-65535	10.0.0.1	22	-	-	accept
2	0	1	udp	192.168.1.0/255.255.255.0	1024-65535	10.0.0.53	53	-	-	accept
3	1	0	tcp	10.0.0.9	1024-65535	192.168.1.15	22	-	-	accept
4	all	all	all	all	all	all	all	all	all	drop

Table 2.1. Possible filtering rules table (ruleset).

A rule in a ruleset has the following parameters which we take into consideration:

- Rank
  The rule's position in a ruleset.
- Interface
  The incoming or outgoing physical network interface of the packet. In our case we assume that the firewall is installed on the gateway which has several physical interfaces.
- Protocol
  In this paper we consider tcp, udp and icmp protocols.

- Source ip
  The source ip address of the packet.
- Source port (ICMP type)
  The source port or icmp type of the packet depending on the protocol.
- Destination ip
  The destination ip address of the packet.
- Destination port (ICMP code)
  The destination port or icmp code of the packet depending on the protocol.
- Action
  The action which is performed on the packet. It can be accepted or dropped depending on the security policy.

We also consider interface and ICMP fields (type and code) in addition to the fields considered by Al-Shaer et al. [1,4,10]. We assume that our firewall is located on the gateway and that is the reason why interface fields are also important.

We describe all possible relations between fields of the same type of two different rules. We denote rule by R_{rank}. The generic i-field of the rule is denoted by $R_{rank}[i]$.

2.1. Relations between fields

In this thesis we consider all possible relations between fields of the same type of rules. Relations between fields are considered briefly by Al-Shaer et al. [1,4,10] when relations between the rules are defined. We decided to present a more detailed description of relations between the rule fields which will help us to understand and define better the relations between firewall rules.

Rule fields can be seen as sets of values. We can give definitions to all possible relations which make sense between the rule fields. It is obvious that only comparison of the rule fields of the same type makes sense.

Definition 2.1.1. Field $R_x[i]$ is a *subset* of field $R_y[i]$ $(R_x[i] \subset R_y[i])$, if the value of $R_y[i]$ contains the value of $R_x[i]$.

Definition 2.1.2. Field $R_x[i]$ is a *superset* of field $R_y[i]$ $(R_x[i] \supset R_y[i])$, if the value of $R_x[i]$ contains the value of $R_y[i]$.

Definition 2.1.3. Field $R_x[i]$ *is equal* to field $R_y[i]$ $(R_x[i] = R_y[i])$, if the value of $R_x[i]$ is equal to the value of $R_y[i]$.

Definition 2.1.4. Field $R_x[i]$ *intersects* with field $R_y[i]$ $(R_x[i] \cap R_y[i])$, if the value of $R_x[i]$ intersects with the value of $R_y[i]$.

Definition 2.1.5. Field $R_x[i]$ is *adjacent* to field $R_y[i]$ $(R_x[i] \parallel R_y[i])$, if the value of $R_x[i]$ is adjacent to the value of $R_y[i]$. This means that the value of $R_y[i]$ is a logical continuation of the sequence of values in $R_x[i]$. For example, value 'ports 23-25' is a logical continuation of 'ports 20-22'.

Definition 2.1.6. Field $R_x[i]$ is *disjoint* with field $R_y[i]$ $(R_x[i] \neq R_y[i])$, if the relation between these fields is not among those defined by Definitions 2.1.1.-5.

In the following section we analyze all possible relations between two rules based on the field relations described in this section.

2.2. Relations between rules

We have almost the same relations as used by Al-Shaer et al. [1,4,10] as well as new ones which are based on the adjacency relation of the rule fields. These are not taken into consideration by Al-Shaer et al. [4] who only use *partially disjoint rules* for the translation of the security policy to a natural language. Since this is not the aim of our work, we will not take into account *partially disjoint* relation and treat it simply as *disjoint*.

Definition 2.2.1. Rule R_x is *equal* to rule R_y $(R_x = R_y)$, if $\forall$ i $R_x[i] = R_y[i]$.

Definition 2.2.2. Rule R_x is *subset* to rule R_y $(R_x \subset R_y)$, if $\exists i : R_x[i] \subset R_y[i]$ and $\forall j \neq i \; R_x[j] \subseteq R_y[j]$.

Definition 2.2.3. Rule R_x is *superset* to rule R_y $(R_x \supset R_y)$, if $\exists i : R_x[i] \supset R_y[i]$ and $\forall j \neq i \; R_x[j] \supseteq R_y[j]$.

Definition 2.2.4. Rule R_x is *adjacent* to rule R_y $(R_x \| R_y)$, if $\exists! i : R_x[i] \| R_y[i]$ and $\forall j \neq i \; R_x[j] = R_y[j]$.

Definition 2.2.5.: Rule R_x is *intersecting* with rule R_y $(R_x \cap R_y)$, if $\exists! i : R_x[i] \cap R_y[i]$ and $\forall j \neq i \; R_x[j] = R_y[j]$.

Definition 2.2.6. Rule R_x is *in correlation* with rule R_y $(R_x \wedge R_y)$, if

$\exists i : R_x[i] \# R_y[i], where \# \in \{\cap, \subset\}, \exists j : R_x[j] \# R_y[j], where \# \in \{\cap, \supset\}$
and $\forall k \neq j, i \; R_x[k] \# R_y[k], where \# \in \{\cap, \supset, \subset, =\}$.

Definition 2.2.7. Rule R_x is *disjoint* with rule R_y $(R_x \neq R_y)$, if the relation between these rules does not correspond to any of the cases in Definitions 2.2.1.-6.

We are not interested in relations which contain inequality since it means that there are no packets which can correspond to both rules in those relations. The rules which are in *adjacency* relation cannot be executed on the same network packet, since adjacent fields do not share values. The reason why we consider adjacency is the ability it provides to merge and decrease the number of rules, hence making the ruleset more efficient. The idea of rules merging, or union as we call it, comes from Acharya et al. [2], Katic and Pale [3], and Chomsiri et al. [9]. This is described in more detail later during the analysis of possible rule errors.

In the next section we describe possible errors in the rulesets of a firewall. In this thesis we call these errors *anomalies*.

2.3 Anomalies

Anomalies can be defined based on the rule relations defined in the previous section. The idea of anomalies was taken from the work done by Al-Shaer et al. [1,4,10].

Definition 2.3.1. Anomaly is a relation between rules in a ruleset which can lead to conflicts or unoptimized behaviour of the firewall.

We distinguish between two different types of anomalies – *anomaly warnings* and *anomaly errors:*

Definition 2.3.2. An anomaly error is an anomaly which clearly results in the unoptimized ruleset and the remedy is known.

Definition 2.3.3. An anomaly warning is an anomaly which may lead to rule conflicts or logical errors.

We take into consideration the fact that only the relation between two rules is considered by Al-Shaer et al. [1,4,10] when talking about anomalies. There can be more complicated dependencies and an anomaly can be caused by larger groups of rules. This problem was partially dealt with by Chomsiri et al. [9]. In Section 2.4. we tackle this problem fully, filling the gaps left by the work done by Al-Shaer et al. [1,4,10] as well as by Chomsiri et al. [9]. We also show how to deal with the fact that an anomaly can be caused by a larger group of rules.

Here we define anomalies based on rule relations defined in the previous section.

Definition 2.3.4. Shadowing anomaly. Rule R_x *shadows* rule R_y if [4]

1. $x < y$ (rank of the R_x rule is smaller than rank of R_y rule)
2. $R_x \supseteq R_y$, i.e. R_x is a superset of R_y or equal to it.
3. Actions of R_x and R_y are different.

In other words, if rule R_x shadows rule R_y, then rule R_y will be never used, since all packets that could hit rule R_y will be processed by rule R_x. We can easily delete rule R_y without changing the security policy of the firewall. Shadowing anomaly is an anomaly error. If R_x shadows R_y, then it can also be said that R_y *is shadowed* by R_x.

Definition 2.3.5. Redundancy anomaly. Rule R_x is *redundant* to rule R_y if [4]

1. $R_x \subseteq R_y$, i.e. R_x is a subset of R_y or equal to it.
2. Actions of these two rules are similar.
3.

If rule R_x is redundant to rule R_y, then R_x can be deleted without changing the security policy of the firewall. The redundancy anomaly is an anomaly error.

Definition 2.3.6. Exception (Generalization [4]) anomaly. Rule R_x is an *exception* to rule R_y if

1. x < y (rank of the R_x rule is smaller than rank of R_y rule)
2. $R_x \subset R_y$, i.e. R_x is a subset of R_y.
3. The actions of these two rules are different.

The exception anomaly is called *generalization anomaly* by Al-Shaer et al. [1,4,10]. They look at this anomaly from the opposite perspective. In our opinion, the term *exception* is more appropriate, since a more general rule can be defined and an exception to it might be defined later. The behaviour of the firewall depends on the relative order of these two rules and a logical error is possible. This is the reason why this relation is considered an anomaly warning.

Definition 2.3.7. Correlation anomaly. Rule R_x is in *correlation* with rule R_y if

1. $R_x \wedge R_y$, i.e., R_x and R_y are in a relation of correlation.
2. Actions of these two rules are different.

This correlation essentially means that *some* packets, processed by rule R_x can also be matched by rule R_y.

The rules in the correlation anomaly match a common set of packets. An example is shown in Table 2.3.1 (we are considering only some fields of rules for the sake of simplicity).

Rank	Source port	Destination port	Action
1	10 – 30	50 – 85	accept
2	15 – 35	75 – 140	deny

Table 2.3.1. Ruleset containing rules in corellation anomaly.

In the example of Table 2.3.1 Rule 1 and Rule 2 are in a correlation relation and form a correlation anomaly. These two rules match the common set of packets with the range of parameters given in Table 2.3.2.

Source port	Destination port
15 – 30	75 – 85

Table 2.3.2. Common part of the rules in correlation anomaly. Correlation part.

The definition of correlation is not complete in articles by Al-Shaer et al. [1,4,10] as the authors talk about subsets and supersets only, forgetting the intersections of the fields. This problem was also mentioned by Chomsiri et al. [9]. We are also taking intersections into consideration by adding them to the definition of correlation relation. We cannot delete a rule in a correlation anomaly without affecting the security policy of the firewall. The behaviour of the firewall depends on the relative order of these two rules and a logical error is possible. This is the reason why this relation is considered an anomaly warning.

Definition 2.3.8. Union anomaly. Rule R_x can be *united (is in union anomaly)* with rule R_y if

1. $R_x \parallel R_y$, i.e., rule R_x is adjacent to rule R_y or
   $R_x \cap R_y$, i.e., rule R_x is intersecting with rule R_y.
2. Actions of the rules are similar.
3. And this union will not change the security policy.

If we can combine two rules without changing the security policy of the firewall, we have an anomaly error.

Definition 2.3.9. Irrelevance anomaly. A rule is *irrelevant* if it does not match any packet going through the firewall.

The irrelevance anomaly is considered by Mohamed et al. [5]. This anomaly can happen if both the source destination address fields of the rule do not match any network domain reachable through the firewall in question [3].

Discovery of an irrelevance anomaly requires additional information and cannot be done just by analyzing the ruleset. This anomaly is an anomaly error. We are not considering this anomaly in this work.

When considering anomalies we are concentrating on two rules only and forgetting other rules in the ruleset. Described anomalies consider only two rules at a time, which might not be enough, especially if there are other rules between those in the presumable anomaly. This is the main problem in the work carried out by Al-Shaer et al. [1,4,10] and is also described by Chomsiri et al. [9].

Chomsiri et al. [9] tackle this problem, and have imposed limitations on the anomaly descriptions. In some cases rules have to be consecutive to be in the anomaly. The union and redundancy anomalies have the same limitation in that work – the rules have to be consecutive. The authors take other rules into account and not only those in the anomaly. Rules recombination is suggested in order to make them consecutive and at the same time the security policy should be kept unchanged, which is not always possible. The fact that one rule can be shadowed by the combination of the preceding rules is also considered by Chomsiri et al. [9] and the Raining 2D-Box model is used to solve this problem.

In this work we suggest another more general approach which will be described in the following section. It takes into consideration all the problems described above.

2.4. Anomalies and limitations

In the previous section we considered anomaly errors and warnings. Anomaly errors are discovered in the case of a suboptimal ruleset and mean that some rules can be *deleted* and/or *united* to make the rule set more optimal, i.e., having fewer rules. A combination of the rules can be also considered as a deletion of one of the rules and a modification of the other.

When talking about anomalies we consider two rules at a time without looking at the whole ruleset. To make it clearer, consider the ruleset given in Table 2.4.1.

Rank	Incoming interface	Outgoing interface	Protocol	Source ip	Source port	Destination ip	Destination port	ICMP code	ICMP type	Action
1	0	1	tcp	192.168.1.1	1024-65535	10.0.0.1	20-30	-	-	accept
2	0	1	tcp	192.168.1.1	1024-65535	10.0.0.1	19-31	-	-	drop
3	0	1	tcp	192.168.1.1	1024-65535	10.0.0.1	15-35	-	-	accept

Table 2.4.1. Ruleset containing anomalies.

According to the definitions given in the previous section, Rule 1 of Table 2.4.1 is redundant to Rule 3 and we can delete it without changing the security policy. In this example this will not work, because Rule 2 will be ”opened”. It means that traffic going to ports 20-30 will be denied by the firewall after deletion of Rule 1.

The problem of Al-Shaer et al. [1,4,10] is to consider only two rules while searching and discovering anomalies. As stated earlier, this problem was recognized and partially solved by Chomsiri et al. [9] by the Raining 2D-Box Model. However, the model proposed is not complete and does not solve the problem of the non-consecutive rules combination when the combined rules are in relation with the other rules between them. In this thesis we introduce another approach which is more practical and easier to implement than the Raining 2D-Box Model and handles the case of combining non-consecutive rules without changing the security policy.

In order to make sure that security policy is maintained intact we have to introduce limitations on the anomalies. Before talking about limitations let us introduce new concepts. These concepts consider actions which are used to optimize the ruleset and can lead to the change in the security policy. These actions are *delete* and *unite*. The rule order can be also changed to optimize the firewall and this is considered in the dynamic optimization part of this thesis.

We can simplify the ruleset, assuming that all rules in it have the same fields except rank, destination port and action.

The first rule in Table 2.4.2 is redundant to the second rule and can be deleted. The rule which was the reason of the deletion has higher rank and situated below the deleted rule. In such case the deletion can be considered as moving the first rule downwards until it is ”absorbed” by the second one. This is called *deletion downwards.*

Rank	Destination port	Action
1	10 – 30	accept
2	5 – 40	accept

Table 2.4.2. Rules in redundancy anomaly. First rule is redundant to the second rule.

The second rule in Table 2.4.3 is redundant to the first rule and can be deleted. The rule which was the reason of the deletion has lower rank and situated above the deleted rule. In such case the deletion can be considered as moving the second rule upwards until it is ”absorbed” by the first one. This is called *deletion upwards.*

Rank	Destination port	Action
1	5 – 40	accept
2	10 – 30	accept

Table 2.4.3. Rules in redundancy anomaly. Second rule is redundant to the first rule.

The first rule in Table 2.4.4 can be united with the second rule. This can be done in two ways:

1. Add the second rule to the first one (part of the second rule is added to the first one. The value of the destination port field of the first rule will become 5 – 50) and delete the second rule. The second rule is moved upwards and ”absorbed” by the first one – union upwards.
2. Add the first rule to the second one (part of the first rule is added to the second one. The value of the destination port field of the second rule will become 5 – 50) and delete the first rule. The first rule is moved downwards and ”absorbed” by the second one – union downwards.

Rank	Destination port	Action
1	5 – 40	accept
2	41 – 50	accept

Table 2.4.4. Rules in union anomaly.

Consider Rule 3 in Table 2.4.5. Rule 3 is redundant to Rule 1 and in correlation anomaly with Rule 2. Deletion of Rule 3 due to redundancy to Rule 1 is called in this case deletion upwards over correlation. Rule 3 is moving upwards in the direction of Rule 1 and ”crosses” Rule 2. Rule 3 is in correlation anomaly with Rule 2.

Rank	Source port	Destination port	Action
1	10 – 15	5 – 40	accept
2	12	7 – 35	drop
3	10 – 13	10 – 30	accept

Table 2.4.5. Union or Deletion upwards and downwards over other anomaly.

In Subsections 2.4.1 and 2.4.2 we consider different cases of unions and deletions over other anomalies and analyze whether security policy is getting changed. This analysis will help us to take into consideration relations between all rules while searching for anomalies and leave the security policy unchanged. In each case we consider three or more rules and assume that they are consecutive. This also works for non-consecutive rules since our anomaly search algorithm should check each rule with every other rule to find anomalies. When the anomaly search algorithm finds out that two rules are in the union or deletion anomaly, it should already know about all rules between these two rules and whether deleted/united rule is moving over any other anomaly. This helps the anomaly search algorithm to keep security policy unchanged.

2.4.1. Union limitations

Firstly, we consider unions upwards and downwards over different anomalies and discover their influence on the security policy. In each case we consider the rule set consisting of at least three rules to make our reasoning more clear. As in the previous section we simplify the ruleset, assuming that all rules in it have the same fields except rank, source port, destination port and action.

2.4.1.1. Union downwards over union anomaly

In the example of Table 2.4.1.1 Rule 1 and Rule 3 are in the union anomaly. Rule 1 and Rule 2 are in the union anomaly, too. If we unite Rule 1 with Rule 3 it will be the union of Rule 1 with Rule 3 downwards over the union anomaly. The resulting ruleset is as shown in Table 2.4.1.2.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31 – 40	22	accept
3	31 – 50	22	accept

Table 2.4.1.1. Union downwards over union anomaly.

Rank	Source port	Destination port	Action
2	31 – 40	22	accept
3	25 – 50	22	accept

Table 2.4.1.2. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25-30 will be still accepted by the modified ruleset.

2.4.1.2. Union downwards over shadowing anomaly

In the example of Table 2.4.1.3 Rule 1 and Rule 3 are in the union anomaly. Rule 1 and Rule 2 are in the shadowing anomaly. Rule 2 is shadowed by Rule 1. If we unite Rule 1 with Rule 3 it will be the union of Rule 1 with Rule 3 downwards over the shadowing anomaly. The resulting ruleset is as shown in Table 2.4.1.4.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	30	22	drop
3	31 – 50	22	accept

Table 2.4.1.3. Union downwards over shadowing anomaly.

Rank	Source port	Destination port	Action
2	30	22	drop
3	25 – 50	22	accept

Table 2.4.1.4. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value 30 will be dropped by the modified ruleset.

2.4.1.3. Union downwards over redundancy anomaly

In the example of Table 2.4.1.5 Rule 1 and Rule 3 are in the union anomaly. Rule 1 is redundant to Rule 2. If we unite Rule 1 with Rule 3 it will be the union of Rule 1 with Rule 3 downwards over the redundancy anomaly. The resulting ruleset is as shown in Table 2.4.1.6.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	25 – 30	22	accept
3	31 – 50	22	accept

Table 2.4.1.5. Union downwards over redundancy anomaly.

Rank	Source port	Destination port	Action
2	25 – 30	22	accept
3	25 – 50	22	accept

Table 2.4.1.6. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25-30 will be still accepted by the modified ruleset.

2.4.1.4. Union downwards over exception (generalization) anomaly

In the example of Table 2.4.1.7 Rule 1 and Rule 3 are in the union anomaly. Rule 1 and Rule 2 are in the exception anomaly. Rule 1 is the exception to Rule 2. If we unite Rule 1 with Rule 3 it will be the union of Rule 1 with Rule 3 downwards over the exception (generalization) anomaly. The resulting ruleset is as shown in Table 2.4.1.8.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	20 – 40	22	drop
3	31 – 50	22	accept

Table 2.4.1.7. Union downwards over exception (generalization) anomaly.

Rank	Source port	Destination port	Action
2	20 – 40	22	drop
3	25 – 50	22	accept

Table 2.4.1.8. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value from range 25 – 30 will be dropped by the modified ruleset.

2.4.1.5. Union downwards over correlation anomaly

In the example of Table 2.4.1.9 Rule 1 and Rule 3 are in the union anomaly. Rule 1 and Rule 2 are in the correlation anomaly. If we unite Rule 1 with Rule 3 it will be the union of Rule 1 with Rule 3 downwards over the correlation anomaly. The resulting ruleset is as shown in Table 2.4.1.10.

Rank	Source port	Destination port	Action
1	25 – 30	22 - 25	accept
2	20 – 40	23	drop
3	31 – 50	22 - 25	accept

Table 2.4.1.9. Union downwards over correlation anomaly.

Rank	Source port	Destination port	Action
2	20 – 40	23	drop
3	25 – 50	22	accept

Table 2.4.1.10. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets that match the correlation part of Rule 1 and Rule 2, and were accepted before the union will be dropped by the modified ruleset.

2.4.1.6. Union upwards over union anomaly

In the example of Table 2.4.1.11 Rule 3 and Rule 1 are in the union anomaly. Rule 3 and Rule 2 are in the union anomaly too. If we unite Rule 3 with Rule 1 it will be the union of Rule 3 with Rule 1 upwards over the union anomaly. The resulting ruleset is as shown in Table 2.4.1.12.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	51 – 60	22	accept
3	31 – 50	22	accept

Table 2.4.1.11. Union upwards over union anomaly.

Rank	Source port	Destination port	Action
1	25 – 50	22	accept
2	51 – 60	22	accept

Table 2.4.1.12. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25 – 30 and 31 – 50 will be still accepted by the modified ruleset.

2.4.1.7. Union upwards over shadowing anomaly

In the example of Table 2.4.1.13 Rule 3 and Rule 1 are in the union anomaly. Rule 3 and Rule 2 are in the shadowing anomaly. Rule 3 is shadowed by Rule 2. If we unite Rule 3 with Rule 1 it will be the union of Rule 3 with Rule 1 upwards over the shadowing anomaly. The resulting ruleset is as shown in Table 2.4.1.14.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31 – 60	22	drop
3	31 – 50	22	accept

Table 2.4.1.13. Union upwards over shadowing anomaly.

Rank	Source port	Destination port	Action
1	25 – 50	22	accept
2	31 – 60	22	drop

Table 2.4.1.14. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value from range 31 – 50 will be accepted by the modified ruleset.

2.4.1.8. Union upwards over redundancy anomaly

In the example of Table 2.4.1.15 Rule 3 and Rule 1 are in the union anomaly. Rule 3 is redundant to Rule 2. If we unite Rule 3 with Rule 1 it will be the union of Rule 3 with Rule 1 upwards over the redundancy anomaly. The resulting ruleset is as shown in Table 2.4.1.16.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31 – 60	22	accept
3	31 – 50	22	accept

Table 2.4.1.15. Union upwards over redundancy anomaly.

Rank	Source port	Destination port	Action
1	25 – 50	22	accept
2	31 – 60	22	accept

Table 2.4.1.16. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25 – 30 will be still accepted by the modified ruleset.

2.4.1.9. Union upwards over exception (generalization) anomaly

In the example of Table 2.4.1.17 Rule 3 and Rule 1 are in the union anomaly. Rule 3 and Rule 2 are in the exception anomaly. Rule 2 is exception to Rule 3. If we unite Rule 3 with Rule 1 it will be the union of Rule 3 with Rule 1 upwards over the exception (generalization) anomaly. The resulting ruleset is as shown in Table 2.4.1.18.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	32 – 40	22	drop
3	31 – 50	22	accept

Table 2.4.1.17. Union upwards over exception (generalization) anomaly.

rank	Source port	Destination port	action
1	25 – 50	22	accept
2	32 – 40	22	drop

Table 2.4.1.18. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value from range 32 – 40 will be accepted by the modified ruleset.

2.4.1.10. Union upwards over correlation anomaly

In the example of Table 2.4.1.19 Rule 3 and Rule 1 are in the union anomaly. Rule 3 and Rule 2 are in the correlation anomaly. If we unite Rule 3 with Rule 2 it will be union of Rule 3 with Rule 1 upwards over the correlation anomaly. The resulting ruleset is as shown in Table 2.4.1.20.

Rank	Source port	Destination port	Action
1	25 – 30	22 - 25	accept
2	20 – 60	23	drop
3	31 – 50	22 - 25	accept

Table 2.4.1.19. Union upwards over correlation anomaly.

Rank	Source port	Destination port	Action
1	25 – 50	22 – 25	accept
2	20 – 60	23	drop

Table 2.4.1.20. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets that match correlation part of Rule 3 and Rule 2 and were dropped before the union will be accepted by the modified ruleset.

2.4.2. Deletion limitations

In this subsection we consider deletions upwards and downwards over different anomalies and discover their influence on the security policy. In each case we consider the ruleset consisting of at least three rules to make our reasoning more clear. We simplify the ruleset, assuming that all rules in it have the same fields except rank, source port, destination port and action. Deletions can be executed in case of redundancy of the rule.

2.4.2.1. Deletion downwards over union anomaly
In the example of Table 2.4.2.1 Rule 1 is redundant to Rule 3. Rule 1 and Rule 2 are in the union anomaly. If we delete Rule 1 it will be the deletion of Rule 1 downwards over the union anomaly. The resulting ruleset is as shown in Table 2.4.2.2.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31 – 40	22	accept
3	20 – 50	22	accept

Table 2.4.2.1. Deletion downwards over union anomaly.

Rank	Source port	Destination port	Action
2	31 – 40	22	accept
3	20 – 50	22	accept

Table 2.4.2.2. Ruleset after anomaly fix.
From the above example we can see that the security policy of the firewall was not

changed. Packets with the source port field value in the range of 25 – 30 will be still accepted by the modified ruleset.

2.4.2.2. Deletion downwards over shadowing anomaly

In the example of Table 2.4.2.3 Rule 1 is redundant to Rule 3. Rule 1 and Rule 2 are in the shadowing anomaly. Rule 2 is shadowed by Rule 1. If we delete Rule 1 it will be the deletion of Rule 1 downwards over the shadowing anomaly. The resulting ruleset is as shown in Table 2.4.2.4.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	30	22	drop
3	20 – 50	22	accept

Table 2.4.2.3. Deletion downwards over shadowing anomaly.

rank	Source port	Destination port	action
2	30	22	drop
3	20 – 50	22	accept

Table 2.4.2.4. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value 30 will be dropped by the modified ruleset.

2.4.2.3. Deletion downwards over redundancy anomaly

In the example of Table 2.4.2.5 Rule 1 is redundant to Rule 3. Rule 1 is also redundant to Rule 2. If we delete Rule 1 because it is redundant to Rule 3 it will be the deletion of Rule 1 downwards over the redundancy anomaly. The resulting ruleset is as shown in Table 2.4.2.6.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	25 – 30	22	accept
3	20 – 50	22	accept

Table 2.4.2.5. Deletion downwards over redundancy anomaly.

rank	Source port	Destination port	action
2	25 – 30	22	accept
3	25 – 50	22	accept

Table 2.4.2.6. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25-30 will be still accepted by the modified ruleset.

2.4.2.4. Deletion downwards over exception (generalization) anomaly

In the example of Table 2.4.2.7 Rule 1 is redundant to Rule 3. Rule 1 and Rule 2 are in the exception anomaly. Rule 1 is the exception to Rule 2. If we delete Rule 1 it will be the deletion of Rule 1 downwards over the exception (generalization) anomaly. The resulting ruleset is as shown in Table 2.4.2.8.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	20 – 40	22	drop
3	20 – 50	22	accept

Table 2.4.2.7. Deletion downwards over exception (generalization) anomaly.

Rank	Source port	Destination port	Action
2	20 – 40	22	drop
3	20 – 50	22	accept

Table 2.4.2.8. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets with the source port field value from range 25 – 30 will be dropped by the modified ruleset.

2.4.2.5. Deletions downwards over correlation anomaly

In the example of Table 2.4.2.9 Rule 1 is redundant to Rule 3. Rule 1 and 2 are in the correlation anomaly. If we delete Rule 1 it will be the deletion of Rule 1 downwards over the correlation anomaly. The resulting ruleset is as shown in Table 2.4.2.10.

Rank	Source port	Destination port	Action
1	25 – 30	22 - 25	accept
2	20 – 40	23	drop
3	20 – 50	22 - 25	accept

Table 2.4.2.9. Deletions downwards over correlation anomaly.

Rank	Source port	Destination port	Action
2	20 – 40	23	drop
3	20 – 50	22	accept

Table 2.4.2.10. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was changed. Packets that match correlation part of Rule 1 and Rule 2, and were accepted before the union will be dropped by the modified ruleset.

2.4.2.6. Deletion upwards over union anomaly

In the example of Table 2.4.2.11 Rule 3 is redundant to Rule 1. Rule 3 and Rule 2 are in the union anomaly. If we delete Rule 3 it will be the deletion of Rule 3 upwards over the union anomaly. The resulting ruleset is as shown in Table 2.4.2.12.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	31 – 60	22	accept
3	25 – 30	22	accept

Table 2.4.2.11. Deletion upwards over union anomaly.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	31 – 60	22	accept

Table 2.4.2.12. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 25 – 30 will be still accepted by the modified ruleset.

2.4.2.7. Deletion upwards over shadowing anomaly

In the example of Table 2.4.2.13 Rule 3 is redundant to Rule 1. Rule 3 and Rule 2 are in the shadowing anomaly. Rule 3 is shadowed by Rule 2. If we delete Rule 3 it will be the deletion of Rule 3 upwards over the shadowing anomaly. The resulting ruleset is as shown in Table 2.4.2.14.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	20 – 60	22	drop
3	25 – 30	22	accept

Table 2.4.2.13. Deletion upwards over shadowing anomaly.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	20 – 60	22	drop

Table 2.4.2.14. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value from range 25 – 30 will be still dropped by the modified ruleset.

2.4.2.8. Deletion upwards over redundancy anomaly

In the example of Table 2.4.2.15 Rule 3 is redundant to Rule 1. Rule 3 is also redundant to Rule 2. If we delete Rule 3 because of Rule 1 it will be the deletion of rule upwards over the redundancy anomaly. The resulting ruleset is as shown in Table 2.4.2.16.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	31 – 60	22	accept
3	30 – 50	22	accept

Table 2.4.2.15. Deletion upwards over redundancy anomaly.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	31 – 60	22	accept

Table 2.4.2.16. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets with the source port field value in the range of 30 – 50 will be still accepted by the modified ruleset.

2.4.2.9. *Deletion upwards over exception (generalization) anomaly*

In the example of Table 2.4.2.17 Rule 3 is redundant to Rule 1. Rule 3 and Rule 2 are in the exception anomaly. Rule 2 is the exception to Rule 3. If we delete Rule 3 it will be the deletion of Rule 3 with upwards over the exception (generalization) anomaly. The resulting ruleset is as shown in Table 2.4.2.18.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	32 – 40	22	drop
3	31 – 50	22	accept

Table 2.4.2.17. Deletion upwards over exception (generalization) anomaly.

Rank	Source port	Destination port	Action
1	20 – 50	22	accept
2	32 – 40	22	drop

Table 2.4.2.18. Ruleset after anomaly fix.

From the above example we can see that security policy of the firewall was not changed. Packets with the source port field value from range 20 – 50 will still be accepted by the modified ruleset.

2.4.2.10. Deletion upwards over correlation anomaly

In the example of Table 2.4.2.19 Rule 3 is redundant to Rule 1. Rule 3 and Rule 2 are in the correlation anomaly. If we delete Rule 3 it will be the deletion of Rule 3 upwards over the correlation anomaly. The resulting ruleset is as shown in Table 2.4.2.20.

Rank	Source port	Destination port	Action
1	20 – 50	22 - 25	accept
2	20 – 60	23	drop
3	30 – 50	22 - 25	accept

Table 2.4.2.19. Deletion upwards over correlation anomaly.

Rank	Source port	Destination port	action
1	20 – 50	22 – 25	accept
2	20 – 60	23	drop

Table 2.4.2.20. Ruleset after anomaly fix.

From the above example we can see that the security policy of the firewall was not changed. Packets that match correlation part of Rule 3 and Rule 2 are still dropped by the modified ruleset.

2.4.2.11. Deletion of the rule upwards will never change the security policy

Based on all previous findings we can say that deletion of the rule upwards will never change the security policy of the firewall even though it can be in relation or anomaly with any other rule in between. The deletion of the rule upwards means that there is another rule with the lower rank which can match the same packets that are matched by the deleted rule. This rule can be called the ”*reason of the deletion*”. We can say that even if the deleted rule was in anomaly with other rules that had lower rank or higher rank the security policy will not be changed after the deletion. The packets will be still processed by the rule which has lower rank and can be one of the following:

1. *reason of the deletion* rule
2. rule which shadows the *reason of the deletion* rule
3. rule to which *reason of the deletion* rule is redundant.

2.4.3 Limitations summary

Table 2.4.3.1 summarizes limitations described in the previous two subsections.

Actions / Anomalies	Union downwards	Union upwards	Deletion downwards	Deletion upwards
Union	NC	NC	NC	NC
Shadowing	C	C	C	NC
Redundancy	NC	NC	NC	NC
Exception	C	C	C	NC
Correlation	C	C	C	NC

Table 2.4.3.1. Anomalies' limitations summary. C – changes the security policy, NC – does not change the security policy

2.5. Anomaly search and correction algorithm

In this section we consider an algorithm for searching and correcting anomalies. The algorithm starts with pairwise comparison of the rules in the ruleset. The comparison is done upwards as well as downwards since some anomalies can be detected in both directions. As an example, consider the ruleset of Table 2.5.1 (again, we simplify the ruleset, assuming that all rules in it have the same fields except rank, source port, destination port and action).

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31 – 40	22	accept
3	20 – 50	22	accept

Table 2.5.1. Ruleset for applying anomaly search algorithm.

The algorithm first compares Rule 1 to Rule 2 and Rule 3, then Rule 2 to Rule 3 and Rule 1 and, at last, Rule 3 to Rule 2 and Rule 1. The sequence of comparisons is important since it follows the relations of the compared rules to all rules between them and helps figuring out the limitations described in the previous section. If the algorithm finds an anomaly error it asks the administrator to choose from three possible options: fix, ignore and edit. Fix action depends on the anomaly error. If anomaly error is *redundancy* or *shadowing*, the rule which is redundant or shadowed will be deleted. If anomaly error is *union*, rules in the anomaly will be united. If the administrator thinks that there is no anomaly error or it was created on purpose, he/she can ignore it by choosing ignore action and the algorithm will continue from the next comparison in the comparison sequence. Administrator can also select edit action and fix the rules manually according to his/her own preference.

Comparison of two different rules involves comparison of fields of these rules belonging to the same type. The result of comparison between two fields is saved to the special array for convenience. Each position of array depicts relation between rules' fields. Possible relations are described in Section 2.1.

The relations array includes the following positions: incoming interface, outgoing interface, protocol, source ip, source port, destination ip and destination port. The comparison result of two fields is stored in the respective field of the array. The relation type can be represented with number:

- 0 – disjoint
- 1 – equal
- 2 – superset
- 3 – subset
- 4 – adjacent
- 5 – intercects.

If the result of comparison of any fields is 0 (disjoint), the rules are disjoint. If none of the results is disjoint, the algorithm continues with assessment of the resulting array. Based on Section 2.2 and given array the relationship between the rules can be easily found out.

The next stage is to figure out if these two rules are in the anomaly. The rank of the rules and their actions are used as well as limitations. Limitations are very important since they help maintaining security policy unchanged. Limitations can be presented as ranks of the rules which define the upper and lower limits of the compared rule. It means that compared rule cannot be moved higher or lower the respective limit without changing the security policy. As we discussed in the previous section, rule movement meant its deletion or union. We have two different limits:

- upper limit
- lower limit.

Since deletion upwards does not change the security policy (as we found out in the previous section), we do not need the upper limit for deletion. It can be confusing, but upper limit is smaller number than lower limit since rules with lower ranks are situated higher than rules with higher ranks in the ruleset.

When comparing rules and searching for anomalies we define the upper and lower limits and if currently compared rules are in the anomaly, we check them against these limits. Suppose, for example, that rule 230 is in exception anomaly with rule 240. It means that 240 is the lower limit for rule 230. If rule 230 is also in, say, correlation anomaly with rule 235, then this rule will be its lower limit. If during the comparison we find out that rule 230 can be united with rule 250 downwards, we will have to check it first against the limits. We see that its lower limit is 235 and we cannot unite it with rule 250, since 250 is lower than its lower limit. In this case algorithm ignores the anomaly since it cannot be fixed without changing the security policy. At the same time, if rule 250 has upper limit higher than 235, it can be united with rule 235 upwards. This possibility will be found out while comparing rule 250 with other rules of the ruleset. This emphasizes the importance of comparisons in both directions.

Anomaly warnings are mostly used for detection of limitations and cannot be fixed automatically by the algorithm. They can be, of course, viewed and fixed by the administrator if need arises.

The suggested method can be given as in Algorithms 2.5.1 and 2.5.2.

```
StaticOptimizationAlgorithm(ruleset)
    deletionLowLimit ← MAX_INT
    deletionUpLimit ← 1
    unionLowLimit ← MAX_INT
    unionUpLimit ← 1

    limits[] ← {deletionLowLimit, deletionUpLimit,
                unionLowLimit, unionUpLimit}

    for i ← 1 to ruleset.size() do
        //compare with rules that have higher rank
        for j ← i + 1 to ruleset.size() do
            anomaly ←  HasAnomaly(ruleset[i], ruleset[j], limits[])

            if anomaly != None do
                //administrator is asked what to do here (fix, ignore, edit)
                //in case of ignore – algorithm just continues
                continue;
                //in case of fix or edit algorithm has to be started all over
                //since ruleset was changed and new rule relationships
                //might have been introduced
                StaticOptimizationAlgorithm(new ruleset)
            endif
        endfor
        //compare with rules that have lower rank
        for j ← i - 1 to 1 do
            anomaly ←  HasAnomaly(ruleset[i], ruleset[j], limits[])

            if anomaly != None do
                //administrator is asked what to do here (fix, ignore, edit)
                //in case of ignore – algorithm just continues
                continue
                //in case of fix or edit algorithm has to be started all over
                //since ruleset was changed and new rule relationships
                //might have been introduced
                StaticOptimizationAlgorithm(new ruleset)
            endif
        endfor
        //compare to the global rule
        anomaly ← HasAnomaly(ruleset[i], globalRule, limits[])

        if anomaly != None do
            //administrator is asked what to do here (fix, ignore, edit)
            //in case of ignore – algorithm just continues
            continue
            //in case of fix or edit algorithm has to be started all over since
            //ruleset was changed and new rule relationships might have been
```

```
                    //introduced
                    *StaticOptimizationAlgorithm(new ruleset)*
            **endif**
    **endfor**
```
Algorithm 2.5.1 Static rules optimization algorithm.

```
*HasAnomaly(rule1, rule2, limits[])*
    //returns relation between two rules
    ruleRelation ← CompareRules(rule1, rule2)
    // checkForAnomaly algorithm takes ruleRelation and limits[] into account
    // and returns anomaly or its absence based on these parameters.
    // Anomaly limitations are important in order to keep the security policy
    // unchanged.
    // This algorithm also updates the limits based on definitions from the previous
    // discussions.
    anomaly ← checkForAnomaly(ruleRelation, limits[])
    **return** anomaly
```
Algorithm 2.5.2. Auxiliary algorithm which compares two rules and returns a possible anomaly.

In the next section we describe and summarize how the following three cases can be handled by our optimization algorithm:

- Rules between the ones in the anomaly.
- Shadowing or redundancy anomaly formed by several rules.
- Union anomaly formed by sets of rules.

2.6. Special cases of static rules optimization

2.6.1. Handling rules between the ones in the anomaly

This special case of handling rules, between the ones in an anomaly, is already handled by our static rules optimization algorithm. At first we consider just two rules and suggest that they are in the anomaly. This is the place where anomaly limitations step in. Since our algorithm has a step-by-step nature, by the time we compare this two rules we know everything about rules in between. In order to make it more clear we can consider the example in Table 2.6.1.1.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	27	22	drop
3	31 – 50	22	accept

Table 2.6.1.1. Example ruleset

Consider now the steps of the algorithm and possible actions of the administrator:

- Rule 1 is compared to Rule 2 – shadowing anomaly is detected. Rule 1 shadows Rule 2.

- Limits for deletion and union are updated. Lower limits for deletion and union are now equal to 2.
- Administrator ignores the detected anomaly.
- Rule 1 is compared to Rule 3 – no anomaly is detected. Even though Rule 1 can be united with Rule 3 it will change the security policy of the firewall. Lower limit of the union is taken into account.

From this example we can see that all rules between the ones under comparison are considered and limitations are created.

2.6.2. Shadowing or redundancy anomaly formed by several rules

Shadowing or redundancy anomalies can be formed by several rules in the ruleset. This was mentioned also by Chomsiri et al. [9]. We can extend our algorithm to take this case into account. Consider the example of shadowing anomaly in Table 2.6.2.1.

Rank	Source ip	Destination ip	Destination port	Action
1	10.0.0.5	20.0.0.1	20-21	accept
2	10.0.0.5	20.0.0.1	21-22	accept
3	10.0.0.5	20.0.0.1	23-25	drop
4	10.0.0.5	20.0.0.1	22-23	accept

Table 2.6.2.1. Shadowing by multiple rules [9].

Rule 4 is shadowed by Rule 3 and Rule 2 together. Our algorithm will make the following findings:

- Rule 1 can be united with Rule 2
- Administrator applies the fix – Rule 1 is now united with Rule 2 – Rule 2 contains both rules
- Rule 3 is in correlation with Rule 4
- Rule 4 is in correlation with Rule 3 (same as previous).

We can see that this case is not handled by our algorithm. Table 2.6.2.2 shows an example where the redundancy anomaly is formed by several rules.

Rank	Source ip	Source port	Destination port	Action
1	10.0.0.5	20-25	20-50	accept
2	10.0.0.5	15-30	15-25	accept
3	10.0.0.5	20-30	15-30	accept

Table 2.6.2.2. Redundancy by multiple rules.

No anomalies will be detected in this ruleset by our algorithm. Though, Rule 3 will be never used and redundant to Rule 1 and Rule 2 together.

In order to tackle this problem we can extend comparison upwards for each rule in the ruleset. When rule is compared to every other rule in the ruleset upwards we can save comparison results of each field of the rule under comparison. We are only interested in the following relationships: subset, superset, intersect and match. If we encounter match or superset, this field is considered completely covered by the preceding rule. If relation is intersect or subset, we consider intersection or subset values to be covered by the preceding rule. If we recognize that all values are covered by one or more preceding rules, this rule is considered to be shadowed by preceding rules and can be deleted without any change to the security policy. This action can be considered as deletion upwards and there are no limitations on deletion upwards as we already know.

We can also keep an eye on rule actions and ranks. If we encounter any of the following relationships: subset, superset, intersect, match, we save rule rank and action. After we have recognized that rules values are completely covered by preceding rules we can check whether we have action that is different to the one of the rule under consideration. We can talk about shadowing if it is different and about redundancy otherwise. We can also show rule ranks which did shadow our rule under consideration, since we have saved them.

Note that all these steps have to be done to every rule independently while comparing it upwards to the other rules of the ruleset.

We apply the modified algorithm to the previously considered examples. For the example in Table 2.6.2.1 we perform the following steps:

- Rule 1 can be united with Rule 2
- Administrator applies the fix – Rule 1 is now united with Rule 2 – Rule 2 contains both rules
- Rule 3 is in correlation with Rule 4
- Rule 4 is in correlation with Rule 3 (same as previous). Destination port 22 is covered by rule 3. All other fields are covered by Rule 3 (they are equal).
- Destination port 23 is covered by Rule 2. Rule 2 has different action – Rule 4 is shadowed by Rule 2 and Rule 3.

In the example in Table 2.6.2.2 we have

- Rule 3. Source ports are covered by Rule 2. Destination ports 15 – 25 are covered by Rule 2. All other fields are equal – completely covered. Destination ports 20 – 30 are covered by Rule 1. Now all fields are covered. All rules have the same action – Rule 3 is redundant to Rule 2 and Rule 1 together.

The examples show that extended algorithm covers the cases of redundancy and shadowing anomalies created by a group of rules.

2.6.3. Union anomaly formed by sets of rules

The case of union anomaly formed by the sets of rules is easily handled by our optimization algorithm. The union anomaly formed by the sets of rules is recognized in several steps. Consider the example ruleset in Table 2.6.3.1 where all three rules generate one union anomaly.

Rank	Source port	Destination port	Action
1	25 – 30	22	accept
2	31	22	accept
3	32 – 50	22	accept

Table 2.6.3.1. Example ruleset

The steps of our algorithm are the following:

- Rule 1 can be united with Rule 2.
- Administrator should apply the fix.
- Rule 2 can be united with Rule 3.
- Administrator should apply the fix.

From this example we can see that our algorithm handles union anomaly formed by the sets of rules as several different union anomalies. Anomaly limitations are handled and the security policy remains unchanged.

3. Dynamic rules optimization

As we have already mentioned, due to the limited time-frame and scope of this work, we mostly concentrate on the static rules optimization techniques and their application to the Iptables firewall. Though, we want to study also the problem of dynamic rules optimization and its possible application to the Iptables firewall.

In this chapter we consider dynamic rules optimization which requires knowledge of the statistical characteristics of traffic going through the firewall. Knowledge from the previous chapter is taken into account. Statically optimized (anomaly errors free) ruleset is used as an input for dynamic optimization.

This area is previously studied by Acharya et al. [2], Al-Shaer et al. [6] and Fulp [7]. We present and analyze the existing methods.

Suppose every rule in the ruleset have a probability of its match by an incoming packet. The aim of dynamic rules optimization is to reorder rules according to the match probability so that average time to determine action to be applied to this packet will be minimal. We have to keep in mind at the same time that rules reordering may lead to the change in the security policy of our firewall.

As we learned from the previous chapter, the mutual location of two different rules may be important to maintain the security policy unchanged. For example, in the case of Table 3.1 we cannot change the order of these two rules so that Rule 20 is before Rule 10.

Rank	Source port	Destination port	Action
10	25 – 30	22	accept
20	any	22	drop

Table 3.1. Ruleset with rules in a precedence relationship.

In other words, as mentioned by Fulp [7], a ruleset has a precedence relationship where certain rules must appear before others if the security policy is to be maintained. Fulp [7] suggests to use directed acyclic graph (DAG) to model the precedence relationship between the rules.

> **Definition 3.1.** Two rules are in the precedence relationship if there exists a packet that matches both rules.

We can say that two rules are in the precedence relationship if they are in one of the following relations: equal, subset, superset, intersect or correlate. Rules in these relations have a common part which can match a network packet. According to the definitions of rule relations, the same packet cannot be matched by rules in other two relations, i.e., disjoint and adjacent.

We can model the precedence relationship between the rules as described by Fulp [7] with a help of a DAG. Let G = (V, E) be a DAG for our ruleset, where vertices V are rules and edges E are the precedence relationships [7]. Linear arrangement of DAG is to be found that improves firewall performance and minimizes average number of comparisons between packets and rules. A theorem introduced by Fulp [7] proves that any linear arrangement of a policy DAG maintains integrity. Hence, topological sort [11] can be used.

As mentioned by Fulp [7] and Al-Shaer et al. [6], some rules have a higher probability of matching a packet than others. The *policy profile* $P = \{p_1, p_2, \ldots, p_n\}$, where p_i is the probability that a packet will match rule i, can be constructed which shows the frequency of matches for every rule in the ruleset. Since Iptables are complete in the sense of Gouda and Liu [8], we say that each packet has a matching rule in the ruleset. It follows that

$$\sum_{i=1}^{n} p_i = 1. \quad (3.1.1)$$

In other words, the average time for required comparisons and determining action over the packet can be written as

$$E = \sum_{i} i\, p_i. \quad (3.1.2)$$

The main aim of dynamic ruleset optimization is to find such permutation of the ruleset that the security policy of the firewall will be maintained and the sum (3.1.2) is minimized.

If all rules are independent and there is no precedence relationship between the rules, the problem can be easily solved by arranging rules in the ruleset according to their matching probabilities in descending order.

If precedence relationships exist, finding optimal permutation is a rather difficult problem. Fulp [7] states that finding the optimal permutation can be considered as job scheduling problem for a single machine with precedence constraints. This problem is *NP-hard* [7].

In the following three sections we present and analyze existing algorithms for dynamic rules optimization.

3.1. A simple heuristic algorithm for finding ruleset permutation (Fulp method)

We describe a heuristic suggested by Fulp [7] which finds the rules permutation which is close to optimal for the problem of finding the best permutation.

Of course, we could analyze all possible rule permutations and test all possible rule orders but it is infeasible to do within realistic time on the large rulesets.
The Fulp heuristic starts with the original ruleset and information about matching probabilities. It sorts rules in non-ascending order based on their matching probabilities and makes sure that sorting will not change the security policy. The precedence relations are taken into consideration while sorting as given in Algorithm 3.1.1.

```
done = false

while(!done)
    done = true
    for(i ← 1; i < n; i++)
        if($p_i < p_{i+1}$ and rules are not in precedence relation)
            interchange rules and probabilities
            done = false
        endif
    endfor
endwhile
```

Algorithm 3.1.1. Heuristic algorithm suggested by Fulp [7]. Here i is the actual position of the rule in the ruleset. Rule ranks are preserved for better understanding.

If all rules in the firewall's ruleset are not in the precedence relation, Algorithm 3.1.1 will give the optimal solution. On the other hand, if all rules are in the precedence relation, it will be absolutely impossible to rearrange the rules in such ruleset without changing the security policy.

Consider now an example presented by Fulp [7] and apply the suggested simple algorithm on it. We will suggest an improvement for this algorithm later.

The ruleset in Table 3.1.1 was considered by Fulp [7].

Rank	Protocol	Src. ip	Src. port	Dst. ip	Dst. port	Action	Match prob.
1	UDP	1.1.0.0/16	any	any	80	drop	0,01
2	TCP	1.0.0.0/8	any	1.0.0.0/8	90	accept	0,02
3	TCP	2.0.0.0/8	any	2.0.0.0/8	20	accept	0,25
4	UDP	1.0.0.0/8	any	any	any	accept	0,22
5	any	any	any	any	any	drop	0,5

Table 3.1.1. Ruleset for applying suggested heuristic.

We can build a DAG representing the precedence relationships between rules in the ruleset in Table 3.1.1. Let us first analyze this ruleset and write down all rules which are in one of the following relations: equal, subset, superset, intersect or correlate. The relations equality, intersection and correlation are symmetric. It means that if Rule 1 is equal to Rule 2, then Rule 2 is equal to Rule 1. The relations subset and superset are reverse relations. It means that if Rule 1 is a subset of Rule 2, then Rule 2 is a superset of Rule 1. Here it is not important in which of the mentioned relations rules are. One pass of comparisons downwards is enough to build the DAG. Moreover, going downwards repeats the way of the packet during its comparison to the rules in the ruleset, it also defines the direction of graph's edges. Our analysis starts with the first rule in the given ruleset and its comparison to the second rule. Each rule in the ruleset is to be compared to every other in the ruleset which is located lower (has higher rank). Comparison upwards is not needed here, unlike in Algorithm 2.5.1. A part of the relations is symmetric but it is not important whether the rules are in subset or superset

relation.
The results of the analysis are the following:

- Rule 1 is a subset of Rule 4
- Rule 1 is a subset of Rule 5
- Rule 2 is a subset of Rule 5
- Rule 3 is a subset of Rule 5
- Rule 4 is a subset of Rule 5.

Based on this information we can build a DAG of this ruleset. The relationships found during the analysis will be used as DAG's edges and rules will be used as DAG's vertices. Direction of edges is set according to ranks of the rules. The resulting DAG is shown in Figure 3.1.1.

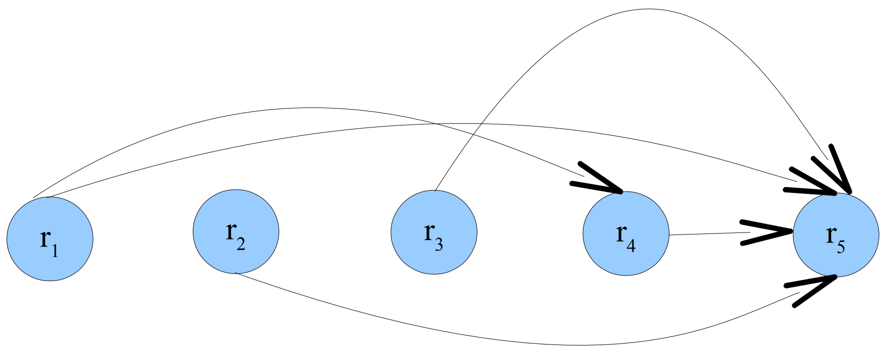

Figure 3.1.1. DAG of the original ruleset.

Now, we can apply the simple heuristic suggested by Fulp [7]. After applying the algorithm we will get the arrangement of the original DAG depicted in Figure 3.1.2. Note, that order of rules in a precedence relation is not changed.

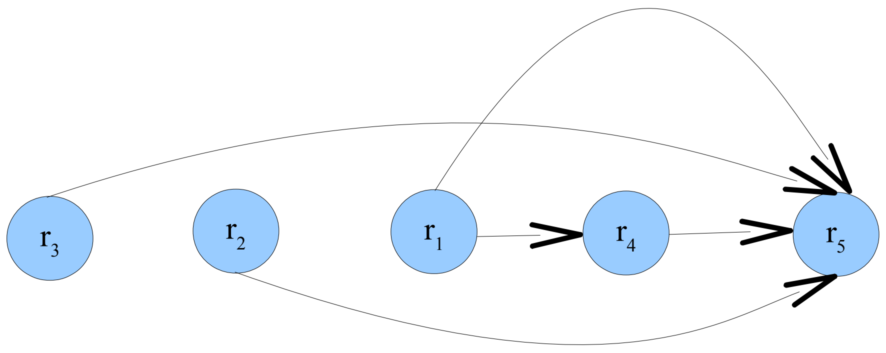

Figure 3.1.2. DAG after applying Algorithm 3.1.1.

According to experimental results of Fulp [7] this rearrangement has 11% less comparisons than the original one. The optimal arrangement of this ruleset was found by applying exhaustive algorithm. It is shown in Figure 3.1.3.

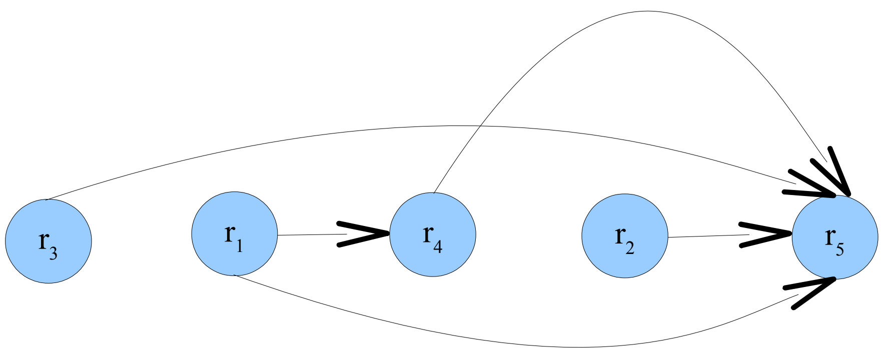

Figure 3.1.3. The optimal arrangement of the ruleset.

Fulp [7] compares only neighbouring rules. This algorithm can be improved by adding an internal loop for comparison of a rule to every other lower rule in the ruleset as shown in Algorithm 3.1.2.

i corresponds to the rule here, not to its position as in Algorithm 3.1; j corresponds to the rule position in the ruleset:

```
for(i ← 1; i <= n; i++)
    for(j ← i.position + 1; j < last_position; j++ )
        if($p_i < p_j$ and rules are not in precedence relation)
            interchange rules and probabilities
        elif rules are in precedence position
            i ← i + 1
            break
        endif
    endfor
endfor
```

Algorithm 3.1.2. Improved heuristic algorithm.

If a rule is in a precedence relation with the other rule in the ruleset, it makes no sense to compare it to other rules. This rule cannot be moved past its precedence relation rule. Therefore it cannot be interchanged with any other rule located lower than precedence relation rule.

The results of our algorithm are better at least on the ruleset used by Fulp [7], since our algorithm gives the solution which coincides with the optimal solution to the given ruleset. This algorithm needs to be tested on the other rulesets and experimental studies are to be done on it, but it is out of this work's scope. Of course the suggested algorithm is just another heuristic and we can introduce other “improvements” increasing the number of comparisons. The main problem is that we cannot prove that this algorithm will give us the expected optimal solution on every ruleset.

Next, we consider another attempt to solve the problem of finding the optimal permutation. Internet flow properties and other statistical characteristics of network traffic were studied by Al-Shaer et al.[6].

3.2. Al-Shaer et al. method

According to work done by Al-Shaer et al. [6], about 20% of the Internet flows consist of ten packets or more, and carry about 70% of the total traffic. It means that Internet traffic consists mostly of small number of heavy-weight flows. Therefore, the number of heavy-weight flows' packet comparisons to the rules should be decreased while filtering traffic [6].

Another finding is that about 20% of the flows last five seconds or more and carry about 60% of the total traffic. It means that Internet traffic consists mostly of small number of long-lived flows. Therefore, the number of long-lived flows' packet comparisons to the rules should be decreased while filtering traffic [6].

Al-Shaer et al. [6] introduce the definition of *locality of flow matching* in the firewall filtering. It states that only a small portion of rules in the security policy is used to match most of the traffic during rather long intervals of time. That is why the problem of dynamic ruleset optimization and matching time reduction is very important. Matching time is the time which firewall spends figuring out what action is to be applied to a packet. [6]

Al-Shaer et al. [6] as well as Fulp [7] notice that the problem of optimal rule ordering with preserving precedence relations and maintaining security policy is NP-hard. Al-Shaer et al. [6] formalize the problem as

$$min \sum_{i=1}^{n} w_i d_i \quad (3.2.1),$$

where d_i is the order of the rule in the ruleset (practically its rank) and w_i is rule's weight. Rule's weight represents its dominance in packet matching. [6]

Fulp [7] does not describe how to calculate the rule's probability. Al-Shaer et al. [6] calculates it based on statistical information of the network traffic and represents it by the rule's weight. We describe the process of rule weight calculation later in this thesis.

Next, we present the heuristic algorithm suggested by Al-Shaer et al. [6]. Experiments performed by Al-Shaer et al. [6] show that it is sufficient to optimize the most active 25-45% of the rules in the policy. Therefore *optimization limit* is used which is the upper bound on the total weight of the selected rules in the optimization process, these are called the *active rules*.

The algorithm described by Al-Shaer et al. [6] works as follows: The algorithm first creates the Max-Heap data structure [11] of the firewall rules based on their weights. The rules are stored with the maximum weight on the top of the structure and can be retrieved in constant time. Rules are taken from the heap one by one in descending order based on their weights. When a rule is taken from the heap, all rules that are in precedence relation with this rule and must precede it are also removed from the heap and inserted in the correct order in the optimized ruleset. Every dependent rule is inserted in the active ruleset such that the rules in the list are in the descending order based on their weights. Then, the rule removed from the heap is appended to the optimized list. This procedure is repeated until the sum of all rule weights in the optimized list exceeds the optimization limit. After that all the remaining rules are removed from the original list and appended to the optimized rule list in their original order. [6] (See Algorithm 3.2.1.)

```
OptimizeActiveRules(rule_list, opt_limit)
    weight ← 0
    H ← BuildMaxHeap(H, rule_list)    //Cormen et. al. [11]
    while H is not empty do
        $R_b$ ← HeapExtractMax(H)
        foreach $R_d \in \{rules\ dependent\ on\ R_b\}$ do
            if $R_d \notin \{active\_rules\}$ then
                current ← ListTail(active_rules)
                find position to insert the rule
                while current ≠ nil do
                    $R_a$ ← ListGet(current)
                    if Weight($R_a$) < Weight($R_d$) and $R_a$
                                not depending on $R_d$ then
                        current ← ListPrevious(current)
                    else
                        break
                    endif
                end while
                ListInsertAfter(current, $R_d$)
                weight ← weight + Weight($R_d$)
                HeapRemove(H, $R_d$)
                ListRemove(rule_list, $R_d$)
            end if
        end foreach
        ListInsertTail(active_rules, $R_b$)
        ListRemove(rule_list, $R_b$)
        if $weight \geqslant opt_{limit}$ then
            break
        end if
    end while
    foreach $R_m \in rule\_list$ do
        ListInsertTail(active_rules, $R_m$)
    end foreach
    return active_rules
```

Algorithm 3.2.1. Heuristic algorithm suggested by Al-Shaer et al. [6]

In Algorithm 3.2.1, R_b represents the rule extracted from the top of the heap; R_d is the rule dependent on the rule R_b; R_a is the rule in the active_list, optimized list; R_m is remaining rules which are not processed because the optimization limit is exceeded.

If Fulp [7] was only concentrating on how to rearrange rules optimally based on given probabilities, Al-Shaer et al. [6] go further and tell how to calculate those probabilities. Since traffic in the Internet is not a constant entity, its statistical parameters always change, hence it was also important to constantly update the probabilities of the rules in the ruleset and keep the statistical parameters up-to-date.

Following Al-Shaer et al. [6] we describe the process of probabilities calculation and how they are kept up-to-date.

As we know, each rule is given a weight which describes its dominance in packet matching. The rule weight is calculated based on two parameters, *frequency* and *recency.*

Frequency tells how frequent the rule has been matched by the packet and recency tells how recent.

Frequency F_i can be expressed as the ratio of number of packets f_i matching rule R_i to the total number of packets matched in the firewall within the same time interval, that is,

$$F_i = \frac{f_i}{\sum_{i=1}^{n} f_i}. \quad (3.2.2)$$

Recency T_i can be expressed as the ratio of the time t_i at which rule R_i was matched last to the time t_{last} of the last rule match in the firewall in the same interval, that is,

$$T_i = \frac{t_i}{t_{last}}. \quad (3.2.3)$$

Al-Shaer et al. [6] mention that calculation of frequency as well as recency involves computationally intensive summation and floating point division operations and suggest using packet-based virtual clock approach instead of real time. It means that number of so far received packets by the firewall can be used as a virtual time. P is a global packet counter. When a rule matches a packet, the frequency of the rule f_i is incremented and the value of P is recorded for this rule as p_i. By using virtual time, frequency and recency can be represented as:

$$F_i = \frac{f_i}{P} \quad (3.2.4)$$

and

$$T_i = \frac{p_i}{P}. \quad (3.2.5)$$

Rule frequency and recency are used to indicate how likely the rule will match a packet in the future. [6]

Empirical study performed by Al-Shaer et al. [6] shows that rule recency is more sensitive to long-lived bursty flows, while rule frequency is sensitive to heavy-weight bulky flows. The weight of the rule was defined by taking this fact into consideration as following:

$$w_i = (1 - \rho)F_i + \rho T_i. \quad (3.2.6)$$

The *recency ratio* ρ was introduced and it tells how much of the rule weight should rely on the rule recency. Experiments performed by Al-Shaer et al. [6] clearly show that increasing the recency factor favours bursty traffic since bursts contain a small number of packets and last for a short period of time. Decreasing the recency factor favours bulky traffic. [6]

Since dynamic optimization should be as fast as possible and rules' weights are calculated online, during the normal work of the firewall, Al-Shaer et al. [6] suggest to avoid the divisions as follows

$$w_i = (1-\rho) Fi + \rho Ti = \rho \frac{p_i}{P} + (1-\rho)\frac{f_i}{P} = \frac{1}{P}[\rho p_i + (1-\rho) f_i] \quad (3.2.7)$$

and

$$w'_i = P w_i = \rho p_i + (1-\rho) f_i. \quad (3.2.8)$$

Equation 3.2.8 still measures the importance of rule R_i relative to other rules in the ruleset and can be used to represent rule weight in heuristic algorithms by Al-Shaer et al. [6] with less processing overhead. It is obvious that weight optimization limit should be multiplied by P to be consistent.

We present now the algorithm suggested by Al-Shaer et al. [6] which integrates Algorithm 3.2.1 as well as frequency and recency calculation into the packet filtering process.

We want to achieve as optimal rule order as possible. Traffic is not constant and its statistical parameters are changing all the time. Rules frequencies and recencies are also changing together with the traffic parameters and rule weights have to be kept up-to-date. Al-Shaer et al. [6] propose two types of rule list updates: performance-based triggered updates and time-based periodic updates. This makes rule ordering as close as possible to the optimal one minimizing this updates in order to avoid excessive processing overhead.

In order to make things more clear before presenting the algorithm we present here how performance-based triggered updates as well as time-based periodic updates are calculated by Al-Shaer et al. [6].

3.2.1. Performance-based triggered updates

The performance-based update is initiated immediately when the filtering performance goes down below expected optimal one. In order to measure the deviation of the actual average number of packet matches from the optimal average number of matches calculated in the last rule list update, *performance deviation factor ε* was introduced. The *update interval* is the time elapsed between two consecutive rule list update events. Performance deviation factor is calculated as follows [6]:

$$\varepsilon = \frac{\sum_{i=1}^{n} p_i d_i}{\sum_{i=1}^{n} q_i d_i} - 1, \quad (3.2.1.1)$$

where d_i is the depth (order) of rule R_i, p_i and q_i are the ratios of packets matching R_i in the current and preceding update intervals, respectively. [6]

Equation 3.2.1.1 is quite accurate but the processing overhead is huge since it has to be calculated for every packet received by the firewall online. It is suggested to calculate the average number of packets matched so far by using an exponential moving average $\bar{h}$:

$$\bar{h}_j = (1-\omega)\overline{h_{j-1}} + \omega h_j, \quad (3.2.1.2)$$

where h_j is the depth (order) of the filtering rule that matched packet j. The coefficient ω represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher ω discounts older observations faster [12]. By using Equation 3.2.1.2, the performance deviation can be computed at any time as

$$\varepsilon = \frac{\bar{h}}{\sum_{i=1}^{n} q_i d_i} - 1 = K\bar{h} - 1\,, \qquad (3.2.1.3)$$

where constant

$$K = \frac{1}{\sum_{i=1}^{n} q_i d_i} \qquad (3.2.1.4)$$

is calculated every time the rule list is optimized. The deviation factor is computed after every packet is matched against the rule list. If value of the deviation factor exceeds a certain *deviation threshold*, a rule list is updated, i.e., a new optimized rule list is constructed. The deviation threshold ε_{thr} is a user configurable parameter to specify the maximum acceptable deviation from the optimal average matching. [6]

3.2.2. Time-based periodic updates

Performance-based triggered updates help increase firewall performance when it becomes significantly lower than user-defined threshold. This updates are not sufficient to detect average performance deviation that is just below the threshold even for a long time. Periodic updates can be performed at fixed time intervals that are relatively long. The matching performance will be closer to its optimum level since up-to-date rule weights will be used. Update periods are also user configurable and should be based on the computational capacity of the filtering device. [6] Too frequent updates can cause the firewall to work even slower than without any optimization.

In the next subsection we present an algorithm suggested by Al-Shaer et al. which integrates Algorithm 3.2.1 as well as frequency and recency calculation into the packet filtering process.

3.2.3. Dynamic rule optimization algorithm integrated into packet filtering process

Algorithm 3.2.3.1 is an integration of the dynamic rule optimization into a typical firewall packet matching (filtering) process. This algorithm performs the standard packet matching procedure after which the corresponding rule action is applied to the incoming packet. Global packet counter used for frequency and recency calculation is incremented as new packets are received. Frequency and recency are updated as well as current average number of packets matched and performance deviation are computed. If the deviation threshold is exceeded or the last periodic update interval expires, rule order optimization algorithm is called and new optimized ruleset with new weights is generated. Global packet counter, frequency and recency of every rule are set to zero. [6]

```
Algorithm MatchPacket(p)
    packet_count ← packet_count + 1
    time ← GetCurrentTime()
    H ← GetPacketHeader(p)
    rule ← MatchRule(H, rule_list)
    if rule ≠ nil then
        action ← rule.action
```

```
            rule.frequency ← rule.frequency + 1
            rule.recency ← packet_count
    else
            action ← DEFAULT_ACTION //matched global rule
    end if
    if action = ALLOW then
            ForwardPacket(p)
    else
            DiscardPacket(p)
    end if
    $\bar{h} \leftarrow (1-\omega) \times \bar{h} + \omega \times rule.order$
    $\varepsilon \leftarrow K \times \bar{h} - 1$
    if $\varepsilon > \varepsilon_{thr}$ or (time – last_time) > UPDATE_PERIOD then
            CalculateRuleWeights(rule_list)
            OptimizeActiveRules(rule_list, OPT_THR)
            foreach rule ϵ rule_list do
                    rule.frequency ← 0
                    rule.recency ← 0
            end foreach
            packet_count ← 0
            last_update ← time
    end if
```

Algorithm 3.2.3.1. Heuristic optimization algorithm suggested in [6] integrated into rule filtering process.

Empirical studies performed by Al-Shaer et al. [6] show that additional calculations have minor overhead in comparison to optimization results. They also show that the suggested algorithms create rule lists that are on average approximately 10% from optimal. Empirical experiments performed by Al-Shaer et al. [6] also show how user-configurable parameters of the algorithm can be tuned.

3.3. Acharya et al. method

Acharya et al. [2] suggest different approach than that described in the previous section. Their rule optimizer consists of two parts – ruleset based optimization and traffic based optimization.

Rule based optimization is done by removing all redundant rules and canceling out all dependencies between the rules by creating new ones. Essentially it makes all rules disjoint in the ruleset. Then it unites all adjacent rules. [2]

Traffic based optimization uses ruleset provided by the rule based optimization. It uses current traffic characteristics to reorder rules in the ruleset and optimize its performance. Four different schemes are used: hot caching, total reordering, default proxy and online adaptation. [2]

The basic idea of hot caching is to identify a small set of rules with large number of packet matches and cache them at the top of the ruleset [2]. This idea is similar to the one described by Al-Shaer et al. [6] which suggests to scope down optimization to a number of rules located on top of the ruleset.

Total reordering of all rules in the ruleset is performed based on current traffic characteristics. It takes into account not only the frequency of the rule but also its size. The size of the rule is the number of bits necessary to determine a match between the rule and incoming packet. It is the number of bits needed to store the values of the rule fields [2]. Since all rules are disjoint it is simple to reorder rules and to be sure that the security policy is not changed.

The default rule of the firewall is invoked if a packet is not matched by any other rule in the ruleset. Usually, the default rule has the highest match ratio. Before the default rule is matched, an incoming packet is checked against every other rule in the ruleset which increases dramatically matching time of the firewall. Additional rules with the same action as default rule can be added to the firewall to alleviate this problem. Values of the packet headers which were matched by the default rule can be used to create additional firewall rules. [2]

A long-term match profile is build offline using traffic characteristics. The short term traffic pattern is then compared with a long term traffic profile. The latter is used to optimize the ruleset. If there is a big difference between traffic characteristics in short term traffic pattern and long-term profile the rules are reordered and explicit default action rules are added. [2]

3.4. Summary

In this chapter we have presented three different approaches to dynamic rules optimization. We briefly summarize advantages and disadvantages of every described method.

3.4.1. Fulp method's advantages and disadvantages

The first method we described uses DAG to depict rule dependencies. We have found out that adding additional for-loop to the original algorithm gives better result since it makes comparison of all rules in the ruleset and not only neighbouring ones. The updated algorithm found the optimal solution to an example problem. Since the problem is NP-hard it is difficult to check whether this algorithm always works optimally. Additional check in the improved algorithm makes sure that dependent rules will maintain mutual order. The main disadvantage of this method is that it does not provide any information on how to calculate probabilities and how to dynamically update them. This method only describes how to find the closest to optimal order of rules based on the given probabilities.

3.4.2. Al-Shaer et al. method's advantages and disadvantages

The second method we described concentrates on the bulky and bursty traffic difference and presents quite detailed description on how to calculate the rule probabilities. This method goes away from the probability term and uses rule weights consisting of matching frequency and recency which describe dominance and importance of the rule in the ruleset. It takes rule dependencies into consideration while reordering the rules. It also keeps rules order as close to optimal as possible by taking the changing nature of traffic into account. It introduces such parameters as deviation threshold and time interval to update rules order depending on its optimality and lifetime. The full algorithm is presented in this work together with formulas needed for rules recency and frequency calculation. Calculations of frequencies and recencies are simplified in order to make calculations overhead smaller, since those values need to be calculated for every incoming packet. It is empirically shown [6] that the solution given by this method is on average not more than 10% far from the optimal solution.

3.4.3. Acharya et al. method's advantages and disadvantages
The third method we described suggests different approach to dependency handling. Firstly, it suggests deleting all redundant rules. If we apply the algorithm from the second section it will not contain any anomaly errors and redundancy is one of them. Secondly, it suggests removing all dependencies by adding new rules mostly by splitting the old ones and then uniting adjacent rules. Then fully independent ruleset can be dynamically optimized by using four different strategies (hot caching, total reordering, default proxy and online adaptation) described earlier. This method creates new rules in order to avoid dependencies it also creates new rules in default proxy strategy. Empirical experiments show that firewall with the resulting ruleset works faster than with original one. The fact that new rules appeared may be surprising for the firewall administrator and may decrease the ruleset readability. It can also make it more difficult for the firewall administrator to edit such ruleset and lead to anomaly creation.

Summarizing all these methods and all their advantages and disadvantages we prefer the method by Al-Shaer et al. The main reasons for this choice are: the readability of the ruleset is maintained, quite good results in the empirical study and the possibility to calculate rules' weights based on frequency and recency. It also provides tunable parameters to configure method's sensibility and updatability.

Other two methods are also worth trying. Fulp method as well as its enhanced version can be tried out by using rule weights from Al-Shaer et al. method in place of probabilities. Unfortunately, empiric comparison of these methods is out of this work's scope.

4. From theory to practice – Iptables optimization

The main target of this work is to analyze how the ruleset optimization methods can be applied to the Linux firewall Iptables. In this chapter we briefly describe the Iptables firewall and its peculiarities.

Iptables belong to the type of the firewalls that are called *packet filters*. Packet filters filter network traffic based on the values of the headers residing in the packet. Iptables is an advanced firewall that supports filtering on different layers of OSI network model. It can be called a packet filter with many different modules and extensions including third party ones. Packet filters operate mostly on the second layer of the TCP/IP stack [13]. However, Iptables supports filtering based on some headers of the transport (TCP, UDP) and data link (MAC) layers. Iptables does not track high level data which flows through its interfaces, in other words it cannot filter based on the application level data. It simply cannot filter based on several packets' headers at the same time and treats each packet as an independent object that has to be either accepted or dropped. Iptables, however, supports connection tracking and NAT which have to be enabled by using several options in the Linux kernel.

4.1. Chains and tables

In order to understand how Iptables work we have to present here its tables and chains and show the way packet makes after entering the firewall. The best description of the Iptables firewall that we found was given by Andreasson [13]. Table 4.1.1 shows a description of Iptables' chains and tables.

Table	Table function	Packet Transformation Chain in the Table	Chain Function
Filter	Packet filtering	FORWARD	Filters packets that traverse through the firewall. Destination and source of the packets are different from the machine where the firewall is installed.
		INPUT	Filters packets destined to the firewall.
		OUTPUT	Filters packets originating from the firewall
NAT	Network address translation	PREROUTING	Address translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address, also known as **destination NAT** or **DNAT**.
		POSTROUTING	Address translation occurs after routing. This implies that there is no need to modify the destination IP address of the packet as in pre-routing. Used with

			NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as **source NAT**, or **SNAT**.
		OUTPUT	Network address translation for packets generated by the firewall.
Mangle	Packet header modification	PREROUTING POSTROUTING OUTPUT INPUT FORWARD	Changing TOS and TTL headers of the packet

Table 4.1.1. Iptables chains and tables [13, 14]

Each chain can have a default policy, either ACCEPT or DROP. This means that if no rule is matching the packet, the default rule will be applied to it.

4.2. Connection tracking

A connection tracking is done by a special framework residing in the kernel module. It helps Iptables to track network connections and to keep track of different streams of data. Connection tracking makes filtering more secure and gives freedom to control the initiation of new sessions. All basic protocols, such as TCP, UDP and ICMP, are supported by the connection tracking framework. [13]

4.3. Iptables rule fields (matches) and their syntax

We describe the fields of the Iptables' rules or matches. In Iptables' documentation they are called matches, because the fields of the packets are matched to this rule fields. We scope down the work by considering only the following fields of the rule:

- Rank
  The rule's position in a ruleset.
- Interface
  The incoming or outgoing physical network interface of the packet.
- Protocol
  In this paper we consider tcp, udp and icmp protocols.
- Source ip
  Source ip address of the packet.
- Source port (icmp type)
  Source port or icmp type of the packet.
- Destination ip
  Destination ip address of the packet.
- Destination port (icmp code)
  Destination port or icmp code of the packet.
- Action
  Can be accept or drop depending on the security policy.

We also consider interface and ICMP fields (type and code) in addition to fields considered by Al-Shaer et al. [1,4,10]. We assume that our firewall is located on the gateway and that is the reason why interface field is also important. Rule fields that are related to session tracking are also important and we consider matches related to TCP flags and conntrack framework.

We apply the methods described in Chapters 2, 3 and 4 to the Iptables firewall and understanding its rule fields and rule syntax is very important. We follow Andreasson [13], since it is the most detailed and comprehensive Iptables description we have found.

Protocol
The protocol field is used to check for certain protocols, transport level protocols such as TCP, ICMP and UDP. The protocol can be also one of those, listed in /etc/protocols file or just an integer value. ALL value means that the rule will match a packet with any protocol value specified in its header.

Source
The source field is used to match packets based on their source IP address. Netmask can be also use to match whole networks. Netmask can be specifyed in two different ways — number of bits used to describe the network, e.g. 192.168.0.0/24; or a regular netmask, e.g. 192.168.0.0/255.255.255.0.

Destination
The destination field is used to match packets based on their destination IP address. Netmask can be also used to match whole networks. Netmask can be specified in two different ways — number of bits used to describe the network, e.g., 192.168.0.0/24; or a regular netmask, e.g., 192.168.0.0/255.255.255.0.

Input interface
The input interface field is used to match packets based on the physical interface the packets came into the firewall.

Output interface
The output interface field is used to match packets based on the physical interface the packets are going to go out of the firewall.

Source port
The source port field is used to match packets based on their source port. The default value of this field is any which implies matching packets with any source port. Source port can be defined as number or name of the service taken from the list stored in the /etc/services file. Source post field supports also ranges, e.g., 22:80 means that rule will match any packet with source port value in a range of 22-80.

Destination port
The destination port field is used to match packets based on their destination port. The default value of this field is any which implies matching packets with any destination port. Destination port can be defined as number or name of the service taken from the list stored in the /etc/services file. Destination post field supports also ranges, e.g., 22:80 means that rule will match any packet with destination port value in a range of 22-80.

TCP flags
This field is used to match packets based on the TCP flags set. The flags can be defined as a list, e.g., SYN, ACK. TCP flags field consists of two subfields: flags that have to be checked in the packet and flags that have to be set in the packet.

ICMP type
This field is used to match packets based on ICMP type and code.

IP range
IP range match can be used if we want to match packets with IP address in specific range. IP range match can be used either for source IP or destination IP.

Limit match
Limit match can be used to limit how many times a certain rule may be matched in a certain time frame. It can be used, for example, to lessen the effects of DoS syn flood attacks.

Multiport match
Multiport match can be used if we want to specify source or destination port list in the rule. Unfortunately Iptables does not support usage of both, standard port and multiport matches in one rule.

State match
As we were talking before, Iptables supports connection tracking and can filter based on that information, such as current state of the connection.

Conntrack match
Conntrack match is an extended version of state match and allows usage information from conntrack session table other than state of the connection. Such information as IP address, port and protocol can be used.

In all the cases above, excluding limit match, match can be also inversed by specifying ! sign.

Iptables is a rather sophisticated piece of software and the scope of this work does not allow us to describe all possible matches of it. That is why we decided to concentrate on the mostly used ones and the ones that we will need to apply algorithms described in the previous sections.

4.4. Iptables targets and jumps

Here we describe possible actions that Iptables can perform on the packet.

Jumps
Jump target can be used when we tell firewall what to do with the packet that matched the rule. Instead of just accepting it or dropping, the packet can be passed to another rule chain within the same table. Then, the packet will be sequentially compared to rules in the new chain. If no rule matches this packet it gets back to the previous chain and matching continues there. Otherwise, if the packet is accepted or dropped in this chain it is also accepted or dropped in the superset chain. [13]

ACCEPT target
When packet is perfectly matched to the rule with ACCEPT target, it is accepted and will not continue traversing this chain or any other chain of the current table. However, this packet might still be checked in the chains within other tables and can be dropped there. [13]

DROP target
When packet is perfectly matched to the rule with DROP target, it is completely dropped and will not continue traversing this chain or any other chain of the current or any other table. [13]

REJECT target
REJECT target works almost the same as DROP target with the only difference that it sends back an error message to the host which sent the packet that was blocked. It can be specified which error message has to be send: TCP with RST flag set, ICMP error message like net unreachable, host unreachable, etc.

LOG target
LOG target is used for logging information about the packets that flow through the firewall. This information can be used for finding the errors in the rules of the firewall. Such information as packet headers can be logged. LOG target can be used instead of DROP one if we want to test the new dropping rule on the firewall and not cause severe disturbance to the working environment. After we make sure that the rule is correct we can change its target from LOG to DROP.

5. Applying optimization methods to the Iptables firewall

In this chapter we apply the optimization methods described in Chapter 2 and Chapter 3 to the Iptables firewall. We take into account all peculiarities of Iptables rule fields. The Iptables firewall is a rather complicated software and it is not feasible to take into consideration all of its features in this work. We consider its restricted and simplified version here.

5.1. Static optimization

Let us go through all rule fields we are interested in and note all possible relationships based on rule field relationships described in the second section.

5.1.1. Rule fields

As we were talking in the previous chapters we have narrowed down our analysis to consider only the following fields:

- Rank
  The rule's position in a rule set.
- Interface
  The incoming or outgoing physical network interface of the packet
- Protocol
  In this paper we will consider tcp, udp and icmp protocols
- Source ip
- Source port (icmp type)
- Destination ip
- Destination port (icmp code)
- Action
  Can be ACCEPT, DROP, REJECT and LOG
- TCP flags.

We consider the Iptables firewall as an internetwork firewall. It means that it is installed on the gateway. In this case we are mostly interested in optimizing FORWARD chain of the filter table. We narrow down our task to optimizing this rule chain. Optimization of other chains of the Iptables firewall can be done in the same manner.

Rank
Rank is one of the most important fields of the rule, since it defines rule's order in the ruleset. As we were talking in the previous sections that rules' order is the most important aspect of the ruleset and it can be crucial for the security policy. Rank field is an integer value that can be given to a rule during its definition. If no rank is specified, the rule will be given a rank automatically by Iptables according to its order in the ruleset.

Input interface
Input interface field was described in the previous section and can be used to filter based on the physical interface of the firewall. The list of values is supported. Subset, superset, equality, intersection, inequality (disjoint) and adjacency relations are possible.

Output interface
Output interface field was described in the previous section and has the same properties as input interface field.

Protocol
Protocol field was described in the previous section and used to filter based on the protocol. TCP, UDP and ICMP protocols will be considered in this work. Neither lists nor ranges are supported by this field. Only equality and inequality (disjoint) relations are supported.

Source IP
Source IP field was described in the previous section and used to filter based on the IP address of the source. IP field can consist of two parts: ip address with or without netmask and ip address' range. Subsets, supersets, equality, intersection, inequality (disjoint) and adjacency relations are supported. Both parts comprising the source ip field value should be taken into consideration while analyzing rule relations.

Source port
Source port field was described in the previous section and used to filter based on the source port value. Source port can be a single value, a range or a list. Subsets, supersets, equality, intersection, inequality(disjoint) and adjacency relations are supported. List and range of source ports cannot be used in the same rule. List of ports has a limit of 15 values. During static optimization algorithm it has to be decided whether to use lists or ranges while uniting the rules based on the port field, since either range or list can be used in the same rule.

Destination IP
Destination IP field was described in the previous section and has the same properties as source IP field.

Destination port
Destination port field was described in the previous section and has the same properties as source port field.

Action
Action field was described in the previous section and can contain a single value which define the action to be performed over the packet. As we mentioned before, we are limiting our consideration to only ACCEPT, DROP, REJECT and LOG values of this field.

TCP flags
TCP flags field was described in the previous section and is used to apply the rule to only the packets with particular flags set. List of values is supported. Note that it does not make sense to unite rules based on this field since they were specially split into different to match packets with the same values but different TCP flags.

Negation
If we take into consideration negation value, which is depicted by ”!” sign in the Iptables ruleset, all relations are also possible for protocol and ICMP type and code fields. Negation sign just means: all other fields except this one. Negation can be

applied to any of the fields mentioned in the table. Negation sign cannot be applied to the list of ports though.

5.1.2. Internal redundancy

As mentioned by Acharya et al. [2], internal redundancy inside the rule is possible. Internal redundancy means that we have repeated values in the same rule field. Iptables syntax allows to define the same value more than once in the fields where list of values is supported. For example, the following value can be given to the source port field: 1024, 1027, 1024, 1029. We can see that value of 1024 is written twice and causes internal redundancy. The algorithm of searching and fixing anomalies should be extended to deal with the internal redundancy.

5.1.3. Rules union extension

We saw that some fields of Iptables can contain lists of values. This can be used to extend our static optimization algorithm. Rules that have the same values in all fields except one and values in that field are disjoint but support lists can be united. For example, consider the ruleset in Table 5.1.3.1.

Rank	Source port	Destination port	Action
1	25	22	accept
2	31	22	accept

Table 5.1.3.1. Rules union extension

Consider Rule 1 and Rule 2. All fields are equal except source port. Since source port field supports lists, we can unite this two rules into the one shown in Table 5.1.3.2.

Rank	Source port	Destination port	Action
1	25, 31	22	accept

Table 5.1.3.2. Rules union extension. After union was applied.

We mentioned previously that during static optimization algorithm it has to be decided whether to use lists or ranges while uniting the rules based on the port field, since either range or list can be used in the same rule and rule list has a limit of fifteen values.

Next, we consider two important Iptables specific parameters that have to be handled during ruleset optimization. The special emphasis is put on these two parameters by Katic and Pale [3].

5.1.4. LOG target and limit match

Taking into consideration rules with LOG target

Iptables can have a special type of rules, rules with a LOG target. As mentioned, this target is used for logging information about the packets that flow through the firewall. Handling of a rule with additional target will change a bit our optimization algorithms. A ruleset can have redundant rules with a LOG target that are not a mistake but created by the administrator on purpose, for debugging. This fact has to be taken into consideration during the static optimization of the firewall and redundancy of this rule should be considered as a warning but not an error. A LOG target differs from other targets in the firewall in the sense that even if a packet matches this rule the processing of it is not finished. Rules following the LOG rule or default rule are applied to that packet. That is why a LOG rule cannot create problems related to shadowing or correlation and its position in the ruleset does not affect the security policy.

Let us consider a LOG rule and define when it is redundant in the ruleset and is a cause of anomaly error:

1. If a LOG rule is redundant to another LOG rule – it is an anomaly error, since this rule will never be used and can be deleted or its order has to be changed.
2. If a LOG rule is shadowed by any other rule – it is an anomaly error, since this rule will never be used and can be deleted or its order has to be changed.

LOG rules have to be taken into consideration also during dynamic optimization. As mentioned by Katic and Pale [3], processing of LOG rules is quite expensive since it may include disk operations and it is desirable to place a LOG rule as low as possible in the ruleset. Relation to other rules should be taken into consideration and a LOG rule should be placed so that it will be matched before the packet is dropped or accepted by some preceding rule.

Taking into consideration limit match

Handling limit option introduces additional problem into the rule optimization process. We mentioned before that it can be used to limit the number of times a certain rule may be matched in a certain time frame. Katic and Pale emphasize [3] that this option can be used to lessen the effects of DoS flood attacks or limit the amount of logging of some specific traffic. The anomaly detection algorithm treats rules with distinct limit options in a bit different way [3]:

1. If a rule is shadowed by any other rule in the ruleset, the shadowing anomaly exists and it can be deleted, since it will never be matched.
2. Any other anomaly error will not be detected, since both of the rules can be matched and they have different limit option. Though anomaly warnings will still be detected, because both of these rules can match the same packets and their order is crucial in maintaining the security policy. If rules are to be united, their limit option parameter should be also equal.

5.1.5. Jumps

As mentioned, we do not consider all possible Iptables rule targets in this work but we want to consider jumps since they are widely used in the Iptables rulesets.

Jump target of the Iptables firewall can look like this:

```
iptables -A FORWARD -p tcp -j tcp_packets
```

The filtering rule match algorithm would then jump from the FORWARD chain to the tcp_packets chain and start traversing that chain. When the end of that chain is reached, the filtering rule match algorithm will get dropped back to the FORWARD chain and the packet starts traversing from the rule one step below where it jumped to the other chain (tcp_packets in this case). If a packet is accepted within one of the subchains, it will be also accepted in the superset chain. This packet will however traverse all other chains in the other tables. If a packet is DROPped, its processing is finished and the firewall will start the rule match algorithm against the next packet in the packet stream. [13]

Next, we discuss about application of static ruleset optimization described in Section 2 to the ruleset of the Iptables firewall.

The ruleset of the Iptables firewall can be defined as a script which adds rules to the ruleset. This script can be taken as an input to the static optimization. The static optimization algorithm can be applied in a straightforward manner as it is described in Chapter 2. We have to take into account all peculiarities of Iptables that we have just mentioned. The static optimization algorithm expects the ruleset to be a linear list and all its nonlinear parts caused by having jumps have to be changed during its application. This can be done by creating temporary ruleset which will be used during the algorithm and that will not have jumps but have the same behaviour as the original one.

Tables 5.1.5.1 and 5.1.5.2 contain an example of the ruleset that has a jump.

Rank	Protocol	Source port	Destination port	Action
1	tcp			Jump to tcp_packets
2	udp	31	22	accept

Table 5.1.5.1. FORWARD chain of the filter table.

Rank	Protocol	Source port	Destination port	Action
1	tcp	25	22	accept
2	tcp	31	22	accept

Table 5.1.5.2. tcp_packets chain.

The temporary ruleset for static optimization algorithm will be the one given in Table 5.1.5.3.

Rank	Protocol	Source port	Destination port	Action
1	tcp	25	22	accept
2	tcp	31	22	accept
3	udp	31	22	accept

5.1.5.3. Temporary ruleset for static optimization.

The rank of the rules have to be updated in order to reflect correct rule order in the ruleset. Anomaly search and correction algorithm can be applied now and its output is an anomaly error free ruleset.

5.2. Dynamic optimization

We consider here the dynamic optimization of the Iptables firewall. As in Chapter 3 we assume that the ruleset is anomaly error free. Method by *Al-Shaer et al.* [6] was chosen in Chapter 3 and we show here how it can be applied for the Iptables ruleset dynamic optimization.

The most important part of the algorithm is calculating and updating rule weights. Calculating rule weight includes calculating its frequency and recency.

The process of calculating weights of rules, frequency and recency is integrated in the process of matching packets. It means that Iptables rule matching algorithm has to be altered to include weights recalculation process.

Applying algorithm without altering Iptables itself is almost impossible. According to the algorithm we need to recalculate frequency and efficiency of the rule every time it was matched. There are no modules of Iptables currently available that can trigger some action or call a script after some rule is matched. The easiest way to overcome this issue is to periodically poll packet counters of Iptables. Every rule of the Iptables has a packet counter which shows how many packets were matched by it.

The main drawback of this approach is that rule recency is impossible to calculate and rule weight will be dependent only on the frequency. The weight of the rule is equal to its frequency in this case and calculated as

$$w_i = F_i = \frac{f_i}{P} = \frac{1}{P} f_i \quad (5.2.1)$$

and

$$w'_i = P w_i = f_i . \quad (5.2.2)$$

This algorithm is still valid and easier to apply to the Iptables without introducing change to its matching algorithm. Weight is calculated in the same way as in Al-Shaer et al. algorithm with the only difference that ρ is equal to 0. We know from Al-Shaer et al. [6] that increasing the recency factor favours bursty traffic and in our case algorithm will favour bulky traffic more. This limitation has to be overcome in the future work.

Performance-based triggered updates are also performed in a different way. When the packet counters of the rules are polled, deviation from optimal matching is calculated using the formulas

$$\varepsilon = \frac{\sum_{i=1}^{n} p_i d_i}{\sum_{i=1}^{n} q_i d_i} - 1, K = \sum_{i=1}^{n} p_i d_i , M = \sum_{i=1}^{n} q_i d_i . \quad (5.2.3)$$

We do not use the optimization method suggested in Chapter 3 when we used exponential moving average. There is no need in it, since we do not update this value after each incoming packet but after time interval has elapsed. We actually cannot calculate that moving average value since it has to be updated after each received packet.

Time-based periodic updates can be performed in the same manner as by Al-Shaer et al. [6]. It has to be noted that time-based periodic updates have to be relatively rare. Al-Shaer et al. [6] suggest that the update period should be determined based on the computational capacity of the filtering device.

Counters have to be polled quite frequent to keep frequency values of the rules up-to-date. The optimal frequency of polling can be calculated based on the empirical study.

It has to be kept in mind that in the original algorithm suggested by Al-Shaer et al. [6] frequency, recency and deviation recalculation was triggered by every incoming packet.

Taking into consideration all described limitations we can write down dynamic optimization algorithm for the Iptables firewall based on the periodic counters polling. The resulting Algorithm 5.2.1 is triggered by timer for periodic counters polling (cron utility can be used):

```
Algorithm DynamicOptimization(ruleset)
    time ← GetCurrentTime()
    foreach rule ϵ rule_list do
        rule.frequency ← rule.counters.packets_matched
    end foreach
    $\varepsilon \leftarrow \frac{K}{M} - 1$
    if $\varepsilon > \varepsilon_{thr}$ or (time – last_time) > UPDATE_PERIOD then
        OptimizeActiveRules(rule_list, OPT_THR) //algorithm from section 3
        last_update ← time
        reset_rule_counters_to_zero
        foreach rule ϵ rule_list do
            rule.frequency ← 0
            rule.counters.packets_matched ← 0
        end foreach
    end if
```

Algorithm 5.2.1. Dynamic rules optimization algorithm applied to the Iptables firewall.

”iptables -Z” command can be used to reset all counters without using any loop.

6. Future work

In this chapter we describe the problems and findings that were discovered in this work but were out of its scope. At the same time we consider them very important and we plan to tackle them in the future.

Empirical study of dynamic rules optimization algorithms
When we were considering dynamic rules optimization in Chapter 3 we suggested an improvement to the heuristic algorithm of Fulp [7]. We have considered algorithms described by Acharya [2] and Al-Shaer et al. [6] as well. The scope of this work and given timeframe did not allow us to perform empirical study and compare these algorithms in practice in their application to the Iptables firewall.

Diverse targets of the Iptables rules
Only a subset of Iptables targets was considered in this work. Other targets should be taken into consideration, especially RETURN target which makes ruleset nonlinear and difficult to apply the static optimization algorithm in Chapter 2.

Empirical study of the suggested algorithms
Empirical study of suggested algorithms should be performed on the rulesets taken from the real corporate firewalls. Firewalls' performance should be analyzed before and after application of the described optimization algorithms. It would be worth analyzing the performance improvement given by the dynamic optimization. It should be analyzed whether calculations overhead of the dynamic optimization cancels out the given improvement and what can be done to lessen its effect.

Optimization algorithm extension
Optimization algorithm can be extended so that it will cover all chains of the Iptables firewall.

7. Conclusions

The importance of the firewall becomes visible when connecting local corporate network to the Internet. It helps to prevent the local corporate network from different threats and intruders.

We have shown that just installing the firewall does not secure the network. The firewall has to be configured appropriately and this is shown to be an error-prone and time consuming process.

We have noticed that there are two important things in the firewall – correctness and efficiency. This means that firewall's ruleset has to reflect the security policy and filter traffic in the most possibly efficient way. It is inevitable that the firewall introduces delay and it has to be as small as possible.

In this work we have considered two different optimization techniques – static and dynamic optimization of the ruleset of the firewall and hence of the firewall itself. Static optimization is mostly about finding errors in the rules and conflicts between them as well as deleting unused ones. Dynamic optimization is mostly about optimization of the ruleset constantly based on the statistical traffic characteristics.

In the second chapter of this work we have analyzed existing static rules optimization approaches and introduced our own approach based on them. We have also identified and filled in the gaps left by existing approaches. Relations between rule fields and rules were introduced. Possible rule errors, anomalies were identified. The problem of maintaining the security policy was raised and it was tackled in the anomalies search and fixing algorithm presented in this thesis.

In the third chapter we have analyzed existing dynamic rules optimization approaches and emphasized advantages and disadvantages of each of them. We have also selected the best approach from our point of view.

In the fourth chapter we have introduced the Iptables firewall and described how does it work and its peculiarities.

The fifth chapter described application of suggested algorithms for dynamic and static ruleset optimization to the Iptables firewalls and shows how to deal with its peculiarities.

Sixth chapter discussed important problems that were not tackled in this work.

References

1. Ehab S. Al-Shaer and Hazem H. Hamed, Firewall policy advisor for anomaly discovery and rule editing. *IFIP/IEEE Eighth International Symposium on Integrated Network Management*, 2003, 17 – 30.

2. Subrata Acharya, Jia Wang, Zihui Ge, Taieb F. Znati and Albert Greenberg, Traffic-aware firewall optimization strategies. *IEEE International Conference on Communications,* 2006, 2225 – 2230.

3. Tihomir Katic and Predrag Pale, Optimization of firewall rules. *29th International Conference on Information Technology Interfaces*, 2007, 685 – 690.

4. Ehab S. Al-Shaer and Hazem H. Hamed, Design and implementation of firewall policy advisor tools. *DePaul CTI Technical Report, CTI-TR-02-006,* August 2002.

5. Mohamed Taibah, Ehab Al-Shaer and Hazem Hamed, Dynamic response in distributed firewall systems. DePaul CTI Technical Report, CTI-TR-05-002, 2004.

6. Hazem Hamed and Ehab Al-Shaer, Dynamic rule-ordering optimization for highspeed firewall filtering. *ACM Symposium on Information, Computer and Communications Security,* 2006*,* 332 – 342.

7. Errin W. Fulp, Optimization of network firewall policies using directed acyclical graphs. *IEEE Internet Management Conference,* 2005.

8. Mohamed G. Gouda and Xiang-Yang Alex Liu, Firewall design: consistency, completeness, and compactness. *24th International Conference on Distributed Computing Systems*, 2004, 320 – 327.

9. Chotipat Pornavalai and Thawatchai Chomsiri. Firewall rules analysis, *International Technical Conference on Circuits/Systems, Computers & Comm. (ITC-CSCC 2004),* 2004.

10. Ehab S. Al-Shaer and Hazem H. Hamed. Modeling and management of firewall policies, *IEEE Transactions on Network and Service Management*, **1** (1), 2004, 2-10.

11. Thomas Cormen, Charles Leiserson, Ronald Rivest and Clifford Stein, *Introduction To Algorithms,* Second edition. The MIT Press, 2002.

12. Wikipedia, *Moving average*. http://en.wikipedia.org/wiki/Moving_average. Checked, March 20, 2011.

13. Oskar Andreasson, *Linux Packet Filtering And Iptables*, 2005.

14. http://www.linuxhomenetworking.com, *Linux firewalls using iptables.*