Classifying Rules by In-out Traffic Direction to Avoid Security Policy Anomaly

Sunghyun Kim and Heejo Lee
Dept. of Computer Science and Engineering, Korea University
Seoul, Korea
[e-mail: afshkim, heejo@korea.ac.kr]
*Corresponding author: Heejo Lee

Received February 11, 2010; revised May 7 and July 18, 2010; accepted July 23, 2010; published August 27, 2010

Abstract

The continuous growth of attacks in the Internet causes to generate a number of rules in security devices such as Intrusion Prevention Systems, firewalls, etc. Policy anomalies in security devices create security holes and prevent the system from determining quickly whether allow or deny a packet. Policy anomalies exist among the rules in multiple security devices as well as in a single security device. The solution for policy anomalies requires complex and complicated algorithms. In this paper, we propose a new method to remove policy anomalies in a single security device and avoid policy anomalies among the rules in distributed security devices. The proposed method classifies rules according to traffic direction and checks policy anomalies in each device. It is unnecessary to compare the rules for outgoing traffic with the rules for incoming traffic. Therefore, classifying rules by in-out traffic, the proposed method can reduce the number of rules to be compared up to a half. Instead of detecting policy anomalies in distributed security devices, one adopts the rules from others for avoiding anomaly. After removing policy anomalies in each device, other firewalls can keep the policy consistency without anomalies by adopting the rules of a trusted firewall. In addition, it blocks unnecessary traffic because a source side sends as much traffic as the destination side accepts. Also we explain another policy anomaly which can be found under a connection-oriented communication protocol.

Keywords: Firewall, security policy, policy anomalies, network security, ACL

This work was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2009-(C1090-0902-0016)) and the IT R&D program of MKE/KEIT (KI001863, The Development of Active Detection and Response Technology against Botnet). Additionally supported by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development.

DOI: 10.3837/tiis.2010.08.013

1. Introduction

The basic function of a firewall is to screen network communications to prevent unauthorized access to or from a computer network [1]. Firewalls decide whether to deny or allow traffic according to a predefined set of rules. Because firewalls provide fundamental protection for the target network, they play a crucial role in the network traffic management. To cope with increasing attacks and threats for network, most firewalls have a large number of rules. A rule consists of predicates for protocol fields and appropriate action. When all predicates of a rule are matched, firewalls allow or deny the packet based on the action of rules. Firewalls are generally defined by order-sensitive and list-based rule set. Therefore, firewalls find the first rule applied to packets among a set of rules.

As Wool observed [2], most firewalls include various types of configuration errors, because the rule management is a complicated, complex and error-prone tasks for network administrators and system managers. Though firewalls have been improved to handle high speed network traffic and a lot of rules, unnecessary rules may downgrade the performance. If there are rules which have different decisions to the same packet, a minor modification of predicates or orders in a rule set can generate security holes. When new rules are added or existing rules are deleted, unless you carefully consider relationships among the rules, firewalls may deny normal services or permit attack traffic.

Predicates in a rule can be presented by multi-dimensional domain regions. Policy anomalies in a rule set result from the overlap of the domain region among the rules. If no overlap exists in all domain regions of predicates, in other words, if all predicates among the rules are completely disjointed with others, policy anomalies will not happen except wrongly-configured rules. Policy anomalies among the rules occur in both single and multiple devices. Unlike in a single firewall, the overlap of predicates among the rules in distributed firewalls can be normal or abnormal relations according to decision of rules.

With the observation of network traffic and dependency among the rules, we propose a new method to remove policy anomalies in a single firewall and avoid policy anomalies in distributed firewalls. The proposed method classifies rules by in-out traffic as our previous research did [3]. It is similar to our previous research in that the proposed method splits overlapping regions from predicates of rules and generates completely disjointed rules without dependency among the rules. However, instead of comparing rules to find anomaly among firewalls, the proposed method uses the aggressive way which could avoid anomalies by replacing the one's rules with the others. The contributions of this study are as follows:

- We found a new kind of misconfiguration based on communication protocols. When some addresses in a rule set send without receiving or receive without sending traffic under a two-way communication protocol, the network connection cannot be established. We included the way to find and resolve the rules having an asymmetrical communication.
- We reduced the unnecessary cost of rule comparisons to find policy anomalies in firewalls. The proposed method classifies rules of all firewalls into two categories, rules for incoming traffic and rules for outgoing traffic. One firewall's rules for incoming traffic have to be matched the other firewalls' rules for outgoing traffic and vice versa. Therefore, without detecting rule anomalies, we replaced the other firewalls' incoming rules by one firewall's outgoing rules or vice versa.

- We blocked the unnecessary traffic from source devices. The proposed method adopts rules from the other devices. It means that the source devices send the packets as much as the destination devices accept and vice versa.

The remainder of this paper is organized as follows: Chapter 2 briefly outlines related work. Chapter 3 explains anomaly problem in security policy. Chapter 4 describes the proposed method for a single security device. Chapter 5 presents the proposed method for multiple security devices. Chapter 6 presents implemented application and experimental results. In Chapter 7, we summarize our experience.

2. Related Work

A firewall is the network equipment that denies or accepts a packet based on policy. Policy anomalies occur when multiple rules are applied a packet in a single device. In list-based firewalls, since only the first matching rule is applied to the packet, the others are useless. However, policy anomalies in distributed firewall are more complicated than in a single firewall. Both network topologies and data paths should be considered when detecting policy anomalies. There have been many challenges to solving such anomalies and maintaining the configuration integrity of the security policy.

Al-Shaer et al. [4][5][6] analyzed anomalies that can occur in a single firewall or in multiple firewalls. They formalized the relations among the rules and represented the firewall policy by a policy tree. They also devised a state diagram for discovering firewall anomalies. This technique was implemented in a software tool called the Firewall Policy Advisor (FPA). The FPA finds potential problems in legacy firewalls and supports anomaly-free policy editing for insertion, removal, and modification of rules.

Hamed et al. [7] provided the taxonomy of policy anomalies classified into access-list conflicts and map-list conflicts in network security devices. They tried to find policy conflicts in various types of security devices and implemented the Security Policy Advisor (SPA) tool, which used the Ordered Binary Decision Diagram (OBDD) [8] to present and manipulate the policy expressions. The SPA supports automatic discovery of security policy conflicts among firewalls including IPSec devices.

Liu and Gupda [9][10][11] proposed three design principles for a firewall: consistency, which means that the rules are ordered correctly; completeness which means that every packet satisfies at least one rule in the firewall; and compactness which means that the firewall has no redundant rules. They developed the Firewall Decision Diagram (FDD) to implement them. They applied a sequence of five algorithms to FDD to generate, to reduce, and to simplify the target firewall rules for maintaining consistency, compactness, and completeness of the original FDD.

Lu et al. [12] proposed a method of representing the firewall rule table that allows for a comparison of two tables. They compared the similarities between a set of packets that are permitted by the two tables. If the sets of packets are same, the two tables are deemed equivalent. This method can also be used to analyze changes to a rule table and to determine whether desired changes are made correctly by comparing the original rule table and the modified one.

Yuan et al. [13] proposed a system known as the FIREwall Modeling and Analysis (FIREMAN), which applies static analysis techniques to check for misconfigurations or policy anomalies in distributed firewalls as well as in individual firewalls. The FIREMAN discovers the violation of user-specific security policies and inconsistencies among firewall rules.

FIREMAN uses the Binary Decision Diagrams (BDDs) that have been used successfully in hardware verification and model checking. The FIREMAN performs symbolic model checking of the firewall configurations for all possible IP packets along all possible data paths. It evaluates the firewall configuration as an entire set that is not just limited to relations between two firewall rules in distributed firewalls.

Alfaro et al proposed the MIsconfiguRAtion manaGer (MIRAGE) [14][15][16], which detects anomalies in a network security policy. They pointed out that some previous research studies were incomplete in their efforts to find all anomalies. They described a set of algorithms to manage policy consistency based on the analysis of relationships between the set of filtering rules. They detected and removed anomalies among rules both in a single device and in multiple devices. In addition, they generated a completely independent rule set that removed correlation among the rules. They compared all rules in firewalls and Intrusion Detection Systems on the network path from the network topology.

Algorithms to find policy anomalies require the high cost because the complexity of comparison among the rules. Pozo et al. [17] proposed the Potential Conflicts Graph (PCG) to diagnose the consistency of the firewall rule set. The PCG isolates all inconsistencies among every pair of rules in an order-independent process and identifies the minimum number of conflicting rules. However, it cannot find all kinds of policy anomalies as they stated. Furthermore, they proposed a diagnostic method to use the Constraint Satisfaction Problem in Artificial Intelligence [18]. Abedin et al. [19] proposed a method to generate a new rule set without anomaly. It simultaneously detects and resolves any anomaly present in the rules by reorder and split operations. Yoon et al. [20] proposed a method to reduce the size of the rule set. The algorithm for the reduction of the rule set finds a group of rules and replaces them with a smaller new group with the same meaning.

Research on policy anomalies mainly finds anomalies based on a set theory. The solution to policy anomalies is to separate or disjoint such rules. Since the separation of rules generates many subsequent rules, it requires the aggregation or the merging of rules. The complexity of this task increases substantially in proportion to the number of rules. The proposed method detects anomalies and generates completely disjointed rules without anomalies. It classifies rules by in-out traffic: rules for incoming traffic and rules for outgoing traffic. Each group has opposite addresses in the source address and the destination address. Since it is unnecessary to compare rules in one group with the other group, the proposed method reduces the number of rules to be compared when finding anomalies among the rules in each firewall. Rule-based packet classification by in-out traffic showed good performance for signature matching an Intrusion Detection System [21].

We devised a unique approach to distributed firewalls. It avoids anomalies by replacing rules with one another among firewalls without anomaly detection. It simply replaces the other firewall's rules with the other firewall's rules that an administrator can trust. If there is no anomaly in each firewall, we can have the rules without anomalies in all distributed firewalls. It also allows as much traffic as the source network and the destination network have to exchange. Therefore, it is a more complete approach in that it blocks the unnecessary traffic from its source. Besides, with the observation of the network protocol and traffic classification, we found a new kind of misconfiguration which had not been found in previous research.

3. Problem of Policy Anomaly

We explain the problem of policy anomaly in a single firewall and among distributed firewalls in this chapter. Policy anomalies occur when one rule has overlapping regions with others in a rule set. In an order-sensitive rules set, when multiple rules are applied to a certain packet, others except the most priority one are abnormal rules. Though all rules are completely disjointed with others, firewalls can have rules which are not relevant to their traffic. Such rules are useless and should thus be removed. We define all kinds of policy anomalies and describe a new anomaly which was found in packet classification by in-out traffic. For the sake of simplicity, we only considered a hierarchical network topology.

3.1 Existing Policy Anomalies

It is very similar to define the terms of policy anomaly [3][4][6][9][10][13]. Based on the previous researches, we explain the types of anomalies in view of set relation. Each rule in firewalls has the form, $< predicates > \rightarrow < decision >$. $< predicates >$ consists of Boolean expressions over protocol fields, such as source address, destination address, source port number, destination port number, etc. $< decision >$ can be "deny" or "accept". We denote a set of rules by R, i.e., $R = \{r_1, r_2, ...r_n\}$. Let r_x, r_y denote one of the rules respectively and assume that r_x has the precedence over $r_y (x < y)$. Let $\Re_C$ denote the correlation. We represent the correlation using other terms, such as partial redundancy and partial shadowing because they can be separated into three subsets, as in the following:

$$r_x \Re_C r_y = \{r_x - r_y\} \bigcup \{r_y - r_x\} \bigcup \{r_y \bigcap r_x\} \quad (1)$$

These three subsets do not have the intersection. Since $r_x - r_y$ is a subset of r_x, $r_y - r_x$ is a subset of r_y, and $r_y \bigcap r_x$ is a subset of r_x and r_y both, $\Re_C$ can be presented by completely disjointed subsets and exactly a matching subset. For the same reason, inclusive relation, $r_x \subseteq r_y$ or $r_x \supseteq r_y$, can be presented with completely disjointed subsets and exactly a matching subset. It means that we can represent all relations only through completely disjointed relations and exactly matching relations if we split them. Therefore, we defined three types of anomalies in a single security device as follows:

Intra-shadowing occurs when any packet which matches the preceding rule r_x also matches the subsequent rule r_y, and r_x has a different decision from r_y.

Intra-redundancy occurs when any packet which matches the preceding rule r_x also matches the subsequent rule r_y, and r_x has the same decision with r_y.

Intra-irrelevance occurs when there is a rule which is irrelevant to the traffic of the device.

Policy anomalies for distributed security devices are more complicated than that in a single security device. To find anomalies among distributed security devices, we have to consider data paths and topologies in the network. We define "zones" as network addresses directly connected to the security device. Let F_s denote the security device in source zone and F_s^x denote one rule of F_s. Let F_d denote the security device in destination zone and F_d^y denote one rule of F_d. We assume that network traffic goes from F_s to F_d. Unlike a single device,

the overlap among rules in multiple firewalls can be normal or abnormal relations depending on their policies. We defined three types of anomalies in multiple security devices as follows:

Inter-shadowing occurs when the source device with a rule F_s^x blocks a packet, the destination device with a rule F_d^y allows the packet. Since F_s^x does not send a packet to F_d^y, F_d^y is unnecessary.

Inter-redundancy occurs when the source device with a rule F_s^x blocks a packet, the destination device with a rule F_d^y blocks the packet again. Since F_s^x does not send a packet to F_d^y, F_d^y is unnecessary.

Inter-spuriousness occurs when the source device with a rule F_s^x allows any packet, the destination device with a rule F_d^y blocks that packet. Since F_s^x sends a packet which is denied by F_d^y, F_s^x is unnecessary.

Inter-redundancy anomaly can be intentionally allowed to enforce the security of a network. For example, a conservative administrator may always block some traffic explicitly out of fear that the upstream firewall may fail. Therefore, though our related works refer to it as an anomaly, it is open to debate whether inter-redundancy anomaly is, in fact, even an anomaly.

3.2 Asymmetry Anomaly

The rules in a firewall generally can be classified by two types as traffic is divided by direction. One type involves the rules for incoming traffic, and the other includes the rules for outgoing traffic. We define them as incoming rules and outgoing rules, respectively. This simple idea greatly reduces the complexity of anomaly detection because we do not need to compare one group with the other group. Also, by classifying them, we can find another misconfiguration. Most network communications require the interactions between hosts or networks. Most of all, when TCP protocol creates a network connection, it requires two-way communication. Therefore, if there is an IP address which only receives or sends a packet in a rule set, it cannot be a normal situation. We define it as an intra-asymmetry anomaly.

Let R_{in} denote a set of incoming rules and R_{out} a set of outgoing rules in rule set R. Also, we denote the set of source addresses and the set of destination addresses used in R_{in} by $S_{R_{in}}$ and $D_{R_{in}}$, respectively. In the same way, let us denote $S_{R_{out}}$ and $D_{R_{out}}$ in R_{out}. The asymmetry anomaly occurs in the following situation:

$$(S_{R_{in}} \neq D_{R_{out}}) \quad or \quad (D_{R_{in}} \neq S_{R_{out}}) \tag{2}$$

Intra-asymmetry occurs when there is a rule which only has a network address for outgoing traffic or for incoming traffic under the protocol of two-way communication.

There are some restrictions in finding an intra-asymmetry anomaly. If the rules do not use a two-way communication, it is unnecessary to find intra-asymmetry anomaly. Also, if there are rules that keep track of currently- established connections in stateful firewalls, such rules do

not have the intra-asymmetry because they are used for both incoming traffic and outgoing traffic. Therefore, when parsing the rules, if there is any rule having a connection state in the rule set, we simply skip that rule. The intra-asymmetry anomaly is a new type of misconfiguration which could not be found in previous research.

ID	Proto.	SIP	SP	DIP	DP	Act.
r_1	TCP	1.1.1.1-5	*	2.1.1.1-5	*	A
r_2	TCP	1.1.1.1-10	*	2.1.1.1-5	*	A
r_3	TCP	1.1.1.10	*	2.1.1.1-5	*	D
r_4	TCP	1.1.1.10	*	2.1.1.11-20	*	D
r_5	TCP	2.1.1.1-10	*	1.1.1.1-10	*	D
r_6	TCP	2.1.1.10	*	2.1.1.1-10	*	A
r_7	TCP	2.1.1.1-20	*	1.1.1.1-10	*	A

ID	Proto.	SIP	SP	DIP	DP	Act.
r_1	TCP	2.1.1.1-5	*	1.1.1.1-10	*	A
r_2	TCP	2.1.1.6-10	*	1.1.1.1-10	*	A
r_3	TCP	2.1.1.10	*	1.1.1.11-20	*	D
r_4	TCP	1.1.1.1-5	*	2.1.1.1-5	*	D
r_5	TCP	1.1.1.10	*	2.1.1.20	*	D
r_6	TCP	1.1.1.10	*	2.1.1.11-30	*	A
r_7	TCP	1.1.1.30	*	2.1.1.1-5	*	A

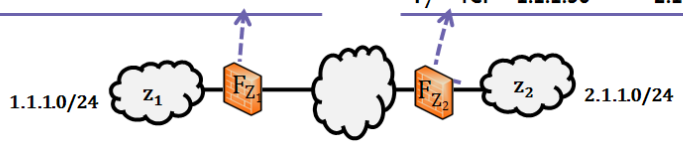


Fig. 1. Example of a network that has deployed two firewalls

3.3 Example of Policy Anomaly

Fig. 1 shows two firewalls deployed in a network. We can find all types of policy anomalies that occur in a single firewall and among multiple firewalls in **Fig. 1**. The anomalies found in **Fig. 1** are listed in **Table 1**.

Table 1. Policy anomalies in **Fig. 1**

Abnormal rule	Anomaly	Associated rule
F_{z_1}'s r_2	Partial intra-redundancy	F_{z_1}'s r_1
F_{z_1}'s r_3	Intra-shadowing	F_{z_1}'s r_2
F_{z_1}'s r_6	Intra-irrelevance	-
F_{z_1}'s r_7	Partial intra-asymmetry	-
F_{z_2}'s r_7	Intra-asymmetry	-
F_{z_2}'s r_2	Inter-spuriousness	F_{z_1}'s r_5
F_{z_1}'s r_4	Inter-redundancy	F_{z_2}'s r_3
F_{z_2}'s r_6	Partial inter-shadowing	F_{z_1}'s r_5

For the purpose of simplicity, port numbers are generalized in this example. In F_{z_1}, r_2 is partially redundant to r_1, while r_3 is completely shadowed with r_1. r_6 in F_{z_1} is irrelevant to F_{z_1}'s traffic. r_7 in F_{z_1} allows receiving packets from IP addresses “2.1.1.1-20”, but there is a rule for sending packets to “2.1.1.1-10”. Therefore, hosts in “2.1.1.11-20” cannot establish the

connection because these rules use TCP. So does r_7 in F_{z_2}. The asymmetry anomaly can occur partially (F_{z_2}'s r_7) or completely (F_{z_1}'s r_7) like the others. Among anomalies between two firewalls, r_2 in F_{z_2} is spurious with r_5 in F_{z_1} because r_2 in F_{z_2} allows traffic blocked by r_5 in F_{z_1}. r_3 in F_{z_1} is completely redundant with r_3 in F_{z_2} because of blocking the same traffic. r_6 in F_{z_2} is partially shadowed with r_5 in F_{z_1} because r_6 in F_{z_2} allows traffic blocked by r_5 in F_{z_1}. A rule with a complete anomaly should be removed from the rule set, while a rule with a partial anomaly requires to be resized to its domain region. We explain how to find all anomalies in the next section.

4. Detecting Anomalies and Rewriting Rules

In this section, we describe a new method not only to find policy anomalies but also to rewrite new rules without anomalies. We classify the rules by in-out traffic to reduce the cost of rule comparisons. If there is a rule not included in two classified groups, the rule has an irrelevance anomaly. To find other anomalies such as shadowing, redundancy, and asymmetry, we devised a bitmap array structure, called the Predicates Bitmap Constructor (PBC). Since correlation and inclusive relation are split by overlapping region and non-overlapping region within a PBC, all the rules have only exactly matching relations or completely disjointed relations. The proposed method uses the PBC to remove all anomalies and rewrite new rules in a firewall.

4.1 PBC (Predicate Bitmap Constructor)

In a firewall, given a set of rules, i.e., $R = \{r_1, r_2, ...r_n\}$, let $F = \{f_1, f_2, ..., f_m\}$ denote the set of m protocol fields presented in R. Let $P_{f_i} = \{p^1_{f_i}, p^2_{f_i}, p^3_{f_i}, ...\}$ denote the set of the predicates associated with f_i in R. Let $V_{f_i} = \{v^1_{f_i}, v^2_{f_i}, v^3_{f_i}, ...\}$ denote the set of distinct comparative values extracted from P_{f_i} used in R in ascending order. We can describe one rule of R, r_x, such as following:

$$r_x = p^i_{f_1} \wedge p^j_{f_2} \wedge .... p^k_{f_m} \tag{3}$$

A predicate used in R can be presented as $p^j_{f_i} = f_i \otimes v^k_{f_i}$, where $\otimes$ is an operator used in the predicates ($=, \leq, \geq$,etc.). Therefore, Eq. (3) can be described as following:

$$r_x = f_1 \otimes v^i_{f_1} \wedge f_2 \otimes v^j_{f_2} \wedge ... f_m \otimes v^k_{f_m} \tag{4}$$

As seen in Eq. (4), one rule consists of conjunctive predicates of protocol fields. Let $dom(f_i)$ denote f_i's domain. When k constant values exist, $dom(f_i)$ can be divided into $(2k+1)$'s interval regions and constant regions at most. According to predicates' comparative values, domain of protocol field f_i can be divided into constant regions and interval regions. That is, $dom(f_i)$ can be divided into $(2k+1)$'s regions, where $k = |V_{f_i}|$, such as following:

$$dom(f_i) = \{v^1_{f_i} > d^1_{f_i}, d^2_{f_i} = v^1_{f_i}, v^1_{f_i} < d^3_{f_i} < v^2_{f_i}, d^4_{f_i} = v^2_{f_i}, ..., d^{2k}_{f_i} = v^k_{f_i}, v^k_{f_i} < d^{2k+1}_{f_i}\} \quad (5)$$

In Eq. (5), $dom(f_i)$ can be divided into two subsets, $\{d^2_{f_i}, d^4_{f_i}, ...d^{2k}_{f_i}\}$ and $\{d^1_{f_i}, d^3_{f_i}, ..., d^{2k+1}_{f_i}\}$. $\{d^2_{f_i}, d^4_{f_i}, ..., d^{2k}_{f_i}\}$ is constant regions and the same to V_{f_i}, which is the set of f_i's comparative values of used in R. $\{d^1_{f_i}, d^3_{f_i}, ..., d^{2k+1}_{f_i}\}$ are interval regions between two constant regions. Based on domain values of f_i, P_{f_i} is determined. Depending on P_{f_i}, we can know that which rules among $r_1, r_2, ...r_n$ are matched. In one of f_i's domain regions, we denote the domain bitmap to represent the result of each rule in R by $s^j_{f_i}$, i.e., $s^j_{f_i} = x_1 \cdot x_2 \cdot x_3 ... \cdot x_n$, where n is the number of rules. We denote the set of R's domain bitmap in each region of $dom(f_i)$ by $S_{f_i} = \{s^1_{f_i}, s^2_{f_i}, ...s^{2k+1}_{f_i}\}$. $\{s^2_{f_i}, s^4_{f_i}, ..., s^{2k}_{f_i}\}$ can be pre-computed by V_{f_i}. Also, $\{s^1_{f_i}, s^3_{f_i}, ..., s^{2k+1}_{f_i}\}$ can be obtained by a random value in each interval domain region. Let α denote one of result bitmaps which shows the result of each rule in R. α can be obtained from each domain bitmap in corresponding protocol fields' domain region as following:

$$\alpha = s^i_{f_1} \,\&\, s^j_{f_2} \,\&\, \cdots \,\&\, s^k_{f_m} \quad (6)$$

Predicates Bitmap Construct (PBC) is an array data structure holding the result bitmaps according to each domain region of a protocol field f_i. We described the structure of the PBC in **Fig. 2** and defined the PBC in **Definition 1**.

Definition 1 (Predicate Bitmap Construct) Given a set of rules $R = \{r_1, r_2, ...r_n\}$, let V_{f_i} denote a set of distinct comparative values extracted from all predicates of a protocol fields f_i in R, i.e. $V_{f_i} = \{v^1_{f_i}, v^2_{f_i}, ..., v^k_{f_i}\}$. Predicate Bitmap Array for f_i, PBC_{f_i} with k distinct constants is defined as an array of $(2k+1)$ regions. Each PBC has following entries:

- **Region identifier (rif)**: An identifier indicates whether its corresponding region is a constant region or an interval region. If $PBC_{f_i}[j]$, the j^{th} of PBC_{f_i}, is constant, it holds one element of V_{f_i}. Otherwise, $PBC_{f_i}[j].rif = null$, implying that $PBC_{f_i}[j-1].rif < PBC_{f_i}[j].rif < PBC_{f_i}[j+1].rif$.
- **Region domain bitmap (*rdb[1..n]*)**: A bitmap stores the pre-computed R's result of its corresponding domain region. In the j^{th} region of PBC_{f_i}, the k^{th} bit of the bitmap is set to *0*, i.e., $PBC_{f_i}[j].rdb[k] = 0$, if r_k's predicate for f_i whose comparative value falls within the region $PBC_{f_i}[j]$ cannot satisfy the predicates for f_i. Otherwise, the k^{th} bit is set to *1*. i.e., $PBC_{f_i}[j].rdb[k] = 1$.

rif	$d_{f_i}^1$	$d_{f_i}^2$	$d_{f_i}^3$	$d_{f_i}^4$	...	$d_{f_i}^{2k}$	$d_{f_i}^{2k+1}$
rdb	$s_{f_i}^1$	$s_{f_i}^2$	$s_{f_i}^3$	$s_{f_i}^4$	...	$s_{f_i}^{2k}$	$s_{f_i}^{2k+1}$

Fig. 2. Predicates Bitmap Constructor for f_i

4.2 Removing Anomalies

We explain anomaly detection and correction in a rule set using PBCs. In a firewall, incoming rules have external addresses in a source address and internal addresses in a destination address, while outgoing rules have opposite source and destination addresses against incoming rules. If there is a rule not included in two rule groups, an irrelevance anomaly is occurred because the rule is irrelevant to traffic of the device. After classifying rules by in-out traffic, we check the policy integrity in each group, R_{out} and R_{in}, respectively. This simple classification reduces the number of rules to be compared in a firewall. Therefore, we create the PBC of each rule group to detect anomalies. For example, there is a rule set which consists of four rules in **Table 2**. If the firewall having this rule set has 1.1.1.0/24 for its zone address, there are three rules for outgoing traffic (r_1, r_2, r_3) and one rule for incoming traffic (r_4).

Table 2. Example of firewall rules

ID	SIP	SP	DIP	DP	Act.
r_1	1.1.1.4	*	2.1.1.[1-10]	*	A
r_2	*	*	2.1.1.4	*	A
r_3	1.1.1.[1-4]	*	2.1.1.4	80	D
r_4	2.1.1.[1-20]	*	1.1.1.4	*	A

We explain how to create the PBC only for the outgoing rules because the other is the same process. Since there is no predicate of source port (SP), three PBCs for a source address (SIP), a destination address (DIP), and a destination port (DP) are created. **Algorithm 1** describes how to create PBC for R_{out}. $v_{f_i}^j + 1$ is the interval region between $v_{f_i}^j$ and $v_{f_i}^{j+1}$. For PBC for SIP, we extract distinct comparative values from the predicates for source address used in r_1, r_2, and r_3. After sorting them in ascending order, we identify constant regions and interval regions between two constant regions as Eq. (5) describes. There are two distinct comparative values, "1.1.1.1" and "1.1.1.4" in r_1, r_2, and r_3. Since we already know the zone address, the minimum and the maximum addresses of the zone are included. Each rule domain bitmap is set by $SetBitmap$ which returns a bitmap of R_{out} in certain domain regions. In the region of source addresses "1.1.1.1" - "1.1.1.3", since r_2 and r_3 except r_1 are matched, the rule domain bitmap has "011". In order to reduce the array size, the domain regions having the same results are merged. The PBC for DIP and the PBC for DP are created in the same way. Three PBCs in **Table 2** are presented in **Fig. 3**. Since the domain of protocol fields in the PBC are divided by the minimum overlapping region, all rules in the PBC are divided by completely disjointed relations or exactly matching relations without correlations and inclusive relations. We can find which rules apply to which domain regions in domain of the protocol field. When rules

share certain domain regions of protocol fields, anomalies occur in a rule set.

Algorithm 1 CreatePBC(f_i, R_{out})

Require: $R_{out} = \{r_1, r_2, ..r_n\}$

Ensure: PBC_{f_i}

1: Extract $V_{f_i} = \{v_{f_i}^1, v_{f_i}^2, ..., v_{f_i}^k\}$ from R_{out}
2: $k \leftarrow |V_{f_i}|$
3: Create PBC having ($2k+1$)'s size
4: $v_{f_i}^0 \Leftarrow 0$
5: **for** $j = 0$ to k **do**
6: **if** $v_{f_i}^j \neq v_{f_i}^{j-1} + 1$ then
7: $PBC_{f_i}[2j].rif \Leftarrow v_{f_i}^j$
8: $PBC_{f_i}[2j+1].rif \Leftarrow null$
9: $PBC_{f_i}[2j].rdb \Leftarrow$ SetBitmap($v_{f_i}^j, R_{out}$)
10: $PBC_{f_i}[2j+1].rdb \Leftarrow$ SetBitmap($v_{f_i}^j + 1, R_{out}$)
11: **end if**
12: **end for**

Require: $r_x : f_1 \otimes v_{f_1}^i \wedge f_2 \otimes v_{f_2}^j \wedge ... f_m \otimes v_{f_m}^k$

Ensure: NL: linked list of normal rules, AL : linked list of abnormal rules

Algorithm 2 ResolveAnomaly(r_x)

1: Extract $v_{f_1}^i, v_{f_2}^j, ...v_{f_m}^k$ from r_x
2: Create m-element of Array, $Ardb$ and $VT = \{v_{f_1}^i, v_{f_2}^j, ...v_{f_m}^k\}$
3: **for** $t = 1$ to m do
4: $Ardb[t] \Leftarrow PBC_{f_t}[VT[t]].rdb$)
5: $rbmap \Leftarrow rbmap$ & $Ardb[t]$
6: **end for**
7: **if** GetHighestNonzeroPosition($rbmap$) = x **then**
8: InsertList($NL, VT, rbmap$)
9: **else**
10: InsertList($AL, VT, rbmap$)
11: **end if**

Algorithm 2 shows the process of anomaly detection and the correction of each rule. We

search the rule domain bitmap in each PBC with comparative values of corresponding predicates and do & (AND) operation on them as Eq. (6) indicates. In the algorithm, *GetHighestNonzeroPosition* returns the position of the highest non-zero bit in the result bitmap. If the position of the highest non-zero bit in result bitmap of r_x is x, r_x is the first rule to be applied to the domain region. Otherwise, the preceding rule is applied in the domain region because of the priority among the rules. When finding the first rule to be applied to that domain region of r_x, we split r_x into a normal domain region and an abnormal domain region. We merge two domain regions when result bitmaps are the same; all domain regions of protocol fields except one are the same, and the exceptional one is consecutive to the other.

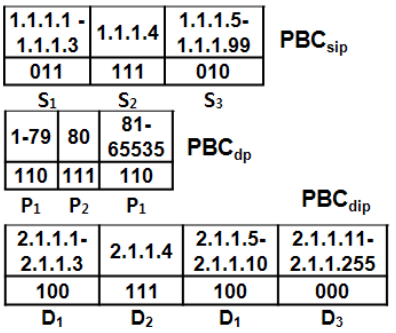
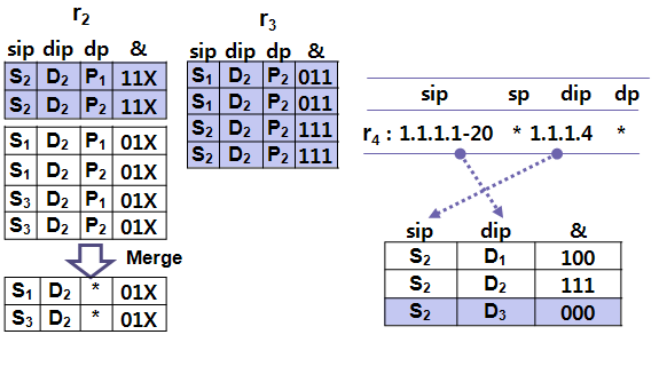


Fig. 3. Example of PBCs



(a) Results of r_2 and r_3 (b) Asymmetry anomaly

Fig. 4. Anomaly detection and rule rewriting using PBCs for rules in Table 2.
("X" means "Don't Care Bit" and colored rows have anomalies)

Fig. 4(a) presents the process of the anomaly detection and rule rewriting for r_2 and r_3 in Table 2. Each row in the table, which was split from the original rule, can be a rule. r_1 Is excluded because it is the highest priority rule. Since the first matched rule is applied, subsequent rules are anomalies. In case of r_2, we can get two split rules without anomalies and overlaps. In case of r_3, r_3 is shadowed by r_2 in the domain region (S_1, D_2, P_2) and redundant with r_1 in the domain region (S_2, D_2, P_2). Therefore, r_3 has a complete anomaly because the preceding rules are first applied in all domain regions of r_3.

The PBC is also used to resolve an asymmetry anomaly. The source addresses for incoming rules must be the same as the destination addresses for outgoing rules and vice versa. Therefore, we compare incoming rules and outgoing rules or vice versa after exchanging the source address and destination address of each group. For the detection of the asymmetry anomaly of incoming rules, we search PBCs for outgoing rules with each rule in incoming rules as described in **Algorithm 3**. Fig. 4(b) shows how to find the asymmetry anomaly with PBCs for outgoing rules and one rule for incoming rules. We search the PBC with exchanged IP addresses and obtain a combined result bitmap, such as finding a redundancy anomaly and a shadowing anomaly. The asymmetry anomaly occurs when the result bitmap has all "0" bits like (S_2, D_3). The rewriting process for removing the asymmetry anomaly is similar to other

anomalies. After finding normal domain regions, we rewrite rules from the original ones.

Algorithm 3 FindAsymmetry(r_x)

Require: r_x : a rule in R_{in}

1: CreatePBC(sip, R_{out})
2: CreatePBC(dip, R_{out})
3: $p_{dip} \Leftarrow$ destination address in r_x
4: $p_{sip} \Leftarrow$ source address in r_x
5: $rbmap \Leftarrow PBC_{sip}[p_{dip}].rdb \ \& \ PBC_{dip}[p_{sip}].rdb$
6: **if** $rbmap = 0$ **then**
7: return true
8: **else**
9: return false
10: **end if**

Table 3 shows the final result without anomalies after rule rewriting. Because there is no overlap among the rules, we do not need to keep "accept" rules and "deny" rules at same time. Therefore, we leave only one type of rules having the same decision. Since firewalls have list-base ACLs which have order-sensitive properties, if the blacklisted IP addresses or white-listed IP addresses are in a rule set, these rules are overlapped by subsequent rules. Therefore, subsequent rules have partial redundancy or shadowing with the blacklisted IP addresses or white-listed IP addresses. The proposed method splits them without overlaps, but it generates too many rules. To avoid such a problem, the proposed method allows them. That is, rules having intentional anomalies can be excluded when rules are parsed.

Table 3. Rewritten rules after removing anomalies

ID	SIP	SP	DIP	DP	Act.
r_1	1.1.1.4	*	2.1.1.[1-10]	*	A
r_2	1.1.1.[1-3]	*	2.1.1.4	*	A
	1.1.1.[5-255]	*	2.1.1.4	*	A
r_3	Removed				
r_4	2.1.1.1-10	*	1.1.1.4	*	A

5. Avoiding Anomalies in Distributed Firewalls

Detecting anomalies and rewriting rules for multiple devices is a little more complicated than doing so for a single device, as discussed above. We propose a different approach to solve anomalies in distributed firewalls; namely, we strive to avoid them. The proposed method can be used without removing intra-anomalies. However, for the sake of simplicity, we assumed intra-anomalies of each firewall were removed by adopting our method for intra-anomalies. Therefore, there is no overlap among the rules in each firewall.

We assume that one firewall can have multiple zones, but a zone is allocated to one firewall.

Packets are exchanged between the zones. We classify firewalls into two types. One is a gateway firewall which is charged with the entire network traffic between the external network and the internal network. The other is a zone firewall which is charged with the network traffic between its own zone and the other zones or the external zone. Let us denote $z_1, z_2, .., z_n$ each zone in the zone firewalls. Also, we denote the set of internal network address by z_{int} and the set of external network address in the gateway firewall by z_{ext}. Let $F_{z_1}, F_{z_2}, ..., F_{z_n}$ be the set of packets allowed in each zone firewall and F_{gw} be the set of packets allowed in a gateway firewall. To present the direction of packets in each firewall, we denote the set of packets which has z_j in their destination address field by $F_{z_i}^{d=z_j}$ and the set of packets which have z_j in their source address field by $F_{z_i}^{s=z_j}$.

Network traffic in a zone firewall can be classified by outgoing traffic and incoming traffic, while network traffic in a gateway firewall can be divided by the internal traffic and the external traffic. For example, $F_{z_i}^{d=z_j}$, which is the traffic heading for z_j in F_{z_i}, can be outgoing traffic and the external traffic. $F_{gw}^{d=z_i}$, which is the traffic heading for z_i in F_{gw}, can be internal traffic and incoming traffic. The ideal state is that the other firewalls have to permit as many as packets that are accepted by z_i. That is, incoming traffic of the zone firewall z_i is the same as the other firewalls' traffic heading for z_i as follows:

$$F_{z_i}^{d=z_i} = \sum_{j=1}^{n} F_{z_j}^{d=z_i} + F_{gw}^{d=z_i} (i \neq j) \tag{7}$$

As the same reason, F_{z_i}'s outgoing traffic is the same to the other firewalls' traffic started from z_i as follows:

$$F_{z_i}^{s=z_i} = \sum_{j=1}^{n} F_{z_j}^{s=z_i} + F_{gw}^{s=z_i} (i \neq j) \tag{8}$$

The gateway firewall has all the zone firewall's traffic heading for internal and external networks as follows.

$$F_{z_{gw}} = \sum_{j=1}^{n} F_{z_j}^{d=z_{\text{int}}} + \sum_{j=1}^{n} F_{z_j}^{s=z_{int}} \tag{9}$$

We denote a set of rules in F_{z_i} by R_{z_i}. Let $R_{z_i}^{s=z_i}$ denote a subset of R_{z_i} whose source address is z_i and $R_{z_i}^{d=z_i}$ denote a subset of R_{z_i} whose destination address is z_i. Eq. (7), (8), and (9) can be represented as follows:

$$R_{z_i} = R_{z_i}^{s=z_i} + R_{z_i}^{d=z_i} = \sum_{j=1}^{n} R_{z_j}^{s=z_i} + \sum_{j=1}^{n} R_{z_j}^{d=z_i} (i \neq j) \tag{10}$$

$$R_{gw} = R_{gw}^{s=z_{int}} + R_{gw}^{d=z_{int}} = \sum_{j=1}^{n} R_{z_j}^{s=z_{\text{int}}} + \sum_{j=1}^{n} R_{z_j}^{d=z_{ext}} \tag{11}$$

Algorithm 4 Avoid Anomaly ($R_{z_i}^{s=z_i}$)

Require: $R_{z_i}^{s=z_i} = \{r_{z_i}^1, r_{z_i}^2, ..., r_{z_i}^m\}$

1: **for** $j = 1$ to n **do**
2: **if** $i \neq j$ **then**
3: **for** $k = 1$ to 1… p **do**
4: **if** $r_{z_j}^k.sip \in GetZoneAddr(F_{z_i})$ **then**
5: delete $r_{z_j}^k$ from $R_{z_j}^{s=z_i}$
6: **end if**
7: **end for**
8: **for** $l = 1$ to m **do**
10: **if** $r_{z_i}^l.sip \in GetZoneAddr(F_{z_j})$ **then**
11: insert $r_{z_i}^l$ into $R_{z_j}^{s=z_i}$
12: **end if**
13: **end for**
14: **end if**
14: **end for**

From Eq. (7), (8), and (10), we have two different approaches. One is to replace each firewall's incoming rules with one firewall's rules heading for the corresponding firewall. Therefore, each firewall receives as much traffic as the corresponding firewall sends. The other is to replace each firewall's outgoing rules with one firewall's rules for the traffic starting from the corresponding firewall. Likewise, each firewall sends as much traffic as the corresponding firewall receives. If an administrator ensures the integrity of F_{z_1}'s incoming rules, the other firewalls' outgoing rules for F_{z_1} can be replaced with F_{z_1}'s incoming rules. Therefore, all firewall rules can have the same integrity for such a traffic flow. When $R_{z_i}^{s=z_i}$ has m rules, i.e. $R_{z_i}^{s=z_i} = \{r_{z_i}^1, r_{z_i}^2, ..., r_{z_i}^m\}$, **Algorithm 4** describes how to replace one firewall's incoming rules with the other firewalls' outgoing rules. In the algorithm, the proposed method deletes all of the other firewalls' rules whose source address is included in the zone address of a firewall F_{z_1} and inserts F_{z_1}'s rules into the corresponding firewalls to replace deleted rules. *GetZoneAddress()* is a function which returns the zone address of parameter. If the algorithm is repeated in all zone firewalls, the proposed method can obtain the consistency among the rules in all zone firewalls because the source zone firewalls send as much traffic as the destination zone firewalls receive. In case of the gateway firewall, from Eq. (9) and (11), it is possible to replace the gateway firewall's rules with each firewall's outgoing rules and incoming rules for the external traffic or vice versa.

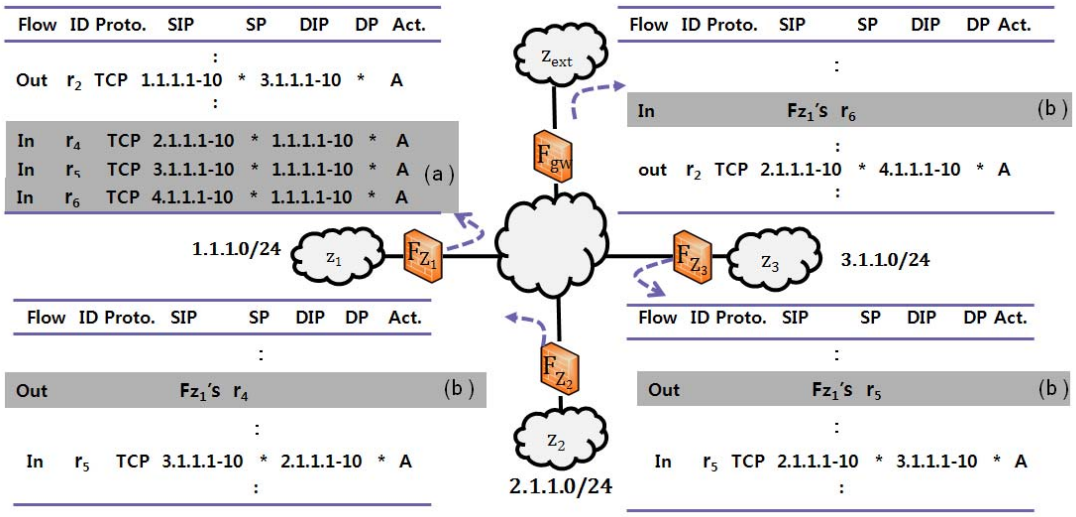


Fig 5. Rule replacement among firewalls.

Fig. 5 shows the simple example of the network diagram deployed with three zone firewalls and one gateway firewall. For the sake of simplicity, we generalized port predicates. All rules have "accept" decisions. If an administrator trusts in F_{z_i}'s rules for incoming traffic, we call F_{z_i} a trusted firewall. As the algorithm describes, the administrator simply replaces outgoing rules of the other firewalls for F_{z_i} with incoming rules in F_{z_i}. In this example, the administrator replaces rules of F_{z2}, F_{z3}, F_{gw} with corresponding rules in F_{z1} (a trusted firewall). For example, if incoming rules in F_{z1} are right, they can be propagated to the other firewalls, (a) is propagated to (b). When the rules in a trusted firewall have different decisions from the rules of the other firewalls, it seems undesirable to replace the "deny" rules of the other firewalls with the "accept" rules of the trusted firewall. However, if all firewalls are in a single administrator's domain and the administrator trusts in the rules of the target firewall, it does not matter to substitute the rules of the trusted firewall for the rules of the other firewalls. By propagating these rules to corresponding firewalls, the other firewalls send or deny packets as the rules of the trusted firewall do.

The proposed method was devised to avoid detection of policy anomalies and comparison of rules and to obtain the consistency among all firewalls. Since the proposed method is based on an administrator's trust in rules, the administrator has to verify the rules before propagating them. Although the administrator trusts the "accept" rules of the target firewall, the administrator may not want to replace the "deny" rules of the other firewall with the "accept" rules of the trusted firewall. We define such a case as an inter-shadowing anomaly because the trusted firewall denies traffic which cannot flow in its zone. Before propagating the rules of target firewall, the administrator has to consider that problem. Trust of the rules includes that there is no policy anomaly in the rules of the trusted firewall. However, if an administrator does not have confidence in the rules of the trusted firewall or if an administrator want to keep the "deny" rules of other firewalls, the administrator can insert the rules of the trusted firewall into other firewall without deleting "deny" rules from other firewalls. In such a case, the rules of other firewalls can deny the traffic which they want to deny but other firewalls may have intra-anomalies, which can be solved by the proposed method for intra-anomalies.

Since there could be some conflicts among rules between two firewalls, some methods have been proposed to detect such anomalies. For that purpose, we also proposed a method to detect and correct policy anomalies among multiple firewalls in our previous paper [3]. After removing policy anomalies among firewalls using such methods, we can adopt the proposed method which replaces the rules of other firewalls with the rules of the trusted firewall.

6. Implementation and Experiments

The proposed method was implemented in a software prototype called the Policy Anomaly Resolver (PAR). The PAR has been coded by C^{++}. The PAR parses rule sets and creates four PBCs, as described above. The PAR can detect overlaps among the rules and rewrite completely disjointed rules without anomalies and overlaps. The size of each PBC for protocol fields depends on the distinct number of comparative values of corresponding predicates. The size of the PBC does not have a great influence on detecting and rewriting performance. However, the overlapping relation among the rules has a great effect on performance. Therefore, the Rule Overlap Count (ROC) was introduced to present how many rules are overlapped. In $R = \{r_1, r_2, ..., r_n\}$, $ROC(r_i)$ indicates how many rules are overlapped between r_i and $r_1, r_2, ..., r_{i-1}$. For example, $ROC(r_i) = 3$ indicates that there are three rules having overlaps between r_i and r_i's preceding rules.

Table 4. Characteristics of rule set

Type	#Rules	#SIP	#SP	#DIP	#DP	Avg.ROC
ACL1	101	12	9	79	14	1.03
ACL2	278	38	45	95	13	0.95
ACL3	388	191	15	62	21	0.01
ACL4	632	106	23	171	26	2.77
SNT	150	2	12	2	146	0.57

For the experiments, we used two types of rule sets. One involves four different ACL rules from the internal network switches, which were deployed in one Korean online game company. **Table 4** shows the detailed characteristics of three ACL rule sets. "#SIP," "#SP," "#DIP," and "#DP" are the numbers of distinct comparative values used in each predicate of a source address, a source port, a destination address, and a destination port, respectively. "Avg. ROC" is the average of ROC in each rule. "ACL3" has few overlaps among the rules, while "ACL4" has a lot of overlaps among the rules. Since the overlap among the rules makes an anomaly, "ACL3" is much better managed compared to "ACL1," "ACL2," and "ACL4." The other one, "SNT" was made from the VRT Certified Rules for Snort, version 2.7. [22]. We classified Snort rules for TCP and grouped them by source address, destination address, source port, destination port, and in-out traffic. Then, we chose 150 rules for outgoing traffic which have specific port numbers in the destination port.

Fig. 6 shows the processing time of the PAR on each rule set. The PAR was executed on a personal computer with 1Gbyte memory and a Core 2 2.13GHz CPU. The cost of rule rewriting increased exponentially according to the number of rules. The major factor for the processing time is the average ROC, as shown in Figure 15. On average, in "ACL4," one rule

is overlapped with three preceding rules. In addition, “ACL4” has a lot of predicates that have “any” type of predicates in the rule set. That is the reason for the rapid increase of execution time. “ACL1,” “ACL2,” and “ACL3,” which have low ROC values show a linear increase in the cost of rule rewriting. Though “ACL3” has more rules than “ACL2,” “ALC3” shows a better execution time than “ACL2” because of the low ROC value.

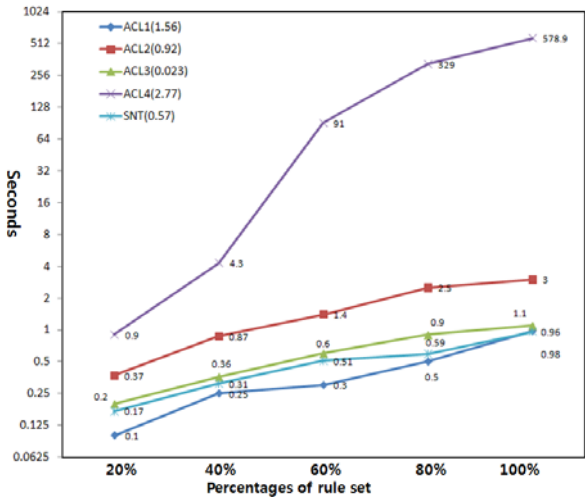


Fig. 6. Execution time for rule each rule set.

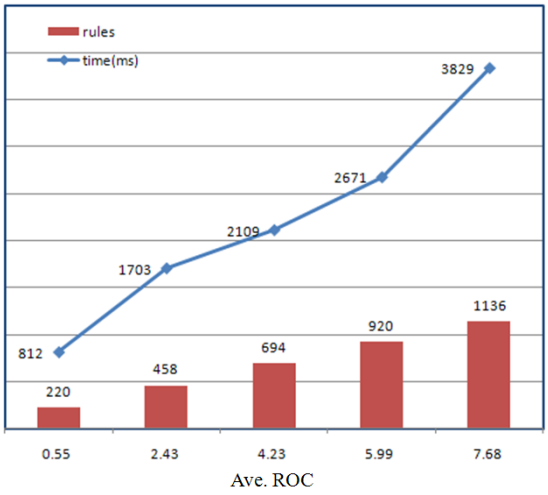


Fig. 7. Execution time and rewritten rules Rewriting in according to average ROC in “SNT” rules.

Also, **Fig. 7** shows the effect of ROC in “SNT” rules. Keeping the number of rules, we changed the average ROC by replacing rules with other rules having “any” destination port in Snort rules. As ROC increases, the execution time and number of rewritten rules are linearly increase. In **Table 5,** after executing the PAR with each rule set, we analyzed the result. The PAR searched policy anomalies in each rule set. For example, the poorly-managed rule set, “ACL4,” with an average ROC of 2.77, has 34% complete redundancy and 65% partial redundancy of total rules, while the well-managed rule set, “ACL3,” with an average ROC of 0.01, has little complete redundancy and partial redundancy.

Table 5. Results of rule rewriting

Type	# Rules	# Complete Redundancy	# Partial Redundancy	# Rewritten Rules	# Max. split rules
ALC1	101	3	7	251	66
ALC2	278	41	70	439	13
ALC3	388	1	8	395	2
ALC4	632	218	413	1277	381
SNT	150	4	41	220	4

The PAR not only finds policy anomalies but also it removes them as **Table 5** shows. Rules with complete redundancy or shadowing are useless in a rule set. Therefore, they have to be

removed from the rule set. Likewise, rules with partial redundancy or shadowing can be rewritten without overlaps. Rewritten rules by the PAR do not have policy anomalies.

7. Conclusions

The policy maintenance is a complex and error-prone task. The policy anomaly problem arises from the overlaps among the rules and results in security holes for attackers. In this paper, classifying rules by the network direction, the proposed method reduced the unnecessary cost of rule comparisons to find policy anomalies. Also, we found a new kind of anomaly which can occur under a two-way communication protocol. The proposed method for a single security device not only removes overlapping relations among the rules, but it also rewrites the rules without the anomalies and overlapping relations. In multiple security devices, we proposed a new way to avoid the anomaly, not to find the anomalies. The proposed method can reduce the overhead to compare rules for finding anomaly and block the unnecessary traffic from the source communication node.

We implemented the proposed method into a window application called the Policy Anomaly Resolver (PAR) and tested it with real rule sets. The PAR converts original rules to non-overlapping and anomaly-free rules without change of original policy. Therefore, it makes tasks for rule management simple and clear. The PAR has disadvantages in the processing overhead for a large scale rules like other methods. Therefore, we are trying to find more efficient method to perform this process.

References

[1] Strasberg,Gondek and Rollies, “The Complete Reference Firewalls,” *MacGrawHill*, 2002.
[2] Avishai Wool, “A quantitative study of firewall configuration errors,” *IEEE Computer*, vol.37, no.6, pp.62-67, Jun. 2004.
[3] Sunghyun Kim and Heejo Lee, “Abnormal policy detection and correction using overlapping transition,” *IEICE Transactions on Information and Systems*, vol.E93-D, no.5, pp.1053-1061, 2010.
[4] Ehab S. Al-Shaer and H. Hamed, “Modeling and management of firewall policies,” *IEEE eTransactions on Network and Service Management*, vol.44, no.3, pp.134-141, Apr. 2004.
[5] E. S. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE Journal on Selected Areas in Communications*, vol.23, no.10, pp.2069-2084, Oct. 2005.
[6] E. S. Al-Shaer and H. Hamed, “Discovery of policy anomalies in distributed firewalls,” in *proc. of IEEE INFOCOM*, pp. 2605-2616, Mar. 2004.
[7] H. Hamed and E. Al-Shaer, “Taxonomy of conflicts in network security policies,” *IEEE Communications Magazine*, vol.44, pp.134-141, 2006.
[8] R. Bryant, “Graph-Based algorithms for Boolean function manipulation,” *IEEE Transactions on Computers*, vol.35, no.8, pp.677-691, Aug. 1986.
[9] M. G. Gouda and A. X. Liu, “Firewall design: consistency, completeness, and compactness,” in *proc. of 24th International Conf. on Distributed Computing Systems (ICDCS)*, 2004.
[10] M. G. Gouda and A. X. Liu, “Structured firewall design,” *Computer Networks Journal*, vol.51, no.4, pp.1106-1120, 2007.
[11] A. X. Liu, and M. G. Gouda, “Diverse firewall design,” *IEEE Transactions on Parallel and Distributed Systems*, vol.19, no.6, pp.1237-1251, 2008.
[12] L. Lu, R. Safavi-Naini, J. Horton and W. Susilo, “Comparing and debugging firewall rule tables,” *International Journal of Information Security*, vol.1, no.4, pp.143-151, 2007.
[13] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, “FIREMAN: A Toolkit for

FIREwall Modeling and Analysis," *IEEE Symposium on Security and Privacy*, pp.199-213, 2006.
[14] J. G. Alfaro, N. Cuppens-Boulahia, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *International Journal of Information Security*, vol.7, no.5, pp.103-122, 2008.
[15] F. Cuppens, N. Cuppens-Boulahia, and J.G. Alfaro, "Detection and removal of firewall misconfiguration," in *proc. of 2005 IASTED International Conf. on Communication, Network and Information Security*, pp.154-162, 2005.
[16] J.G. Alfaro, F. Cuppens, and N. Cuppens-Boulahia, "Aggregating and deploying network access control policies," in *proc. of Third International Conf. on Availability, Reliability and Security*, 2007
[17] S. Pozo, R. Ceballos, and R. M. Gasca, "Fast algorithms for consistency-based diagnosis of firewall Rule Sets," in *proc. of Second International Conf. on Availability, Reliability and Security*, 2006.
[18] S. Pozo, R. Ceballos, and R. M. Gasca, "CSP-based firewall rule set diagnosis using security policies," in *proc. of Third International Conf. on Availability, Reliability and Security*, 2007.
[19] M. Abedin, S. Nessa, L. Khan, and B. Thuraisingham, "Detection and resolution of anomalies in firewall policy rules," in *proc. of 20th Annual IFIP WG 11.3 Working Conf. on Data and Applications Security (DBSec)*, 2006.
[20] M. Yoon, S. Chen, and Z. Zhang, "Reducing the size of rule set in a Firewall," in *Proc. of IEEE International Conf. on Communications*, 2007.
[21] Sunghyun Kim and Heejo Lee, "Reducing payload inspection cost using rule classification for fast attack signature matching," *IEICE Transactions on Information and Systems*, Vol.E92-D, no.10, pp.1971-1978, 2009.
[22] Snort: Open source Network Intrusion Detection System, http://www.snort.org.

Sunghyun Kim received the B.S. degree in Computer Science from Pukyung University, Korea, in 1994, and the M.S. degree in Computer Science from Yonsei University, Korea, in 2006. Currently, he is a Ph.D. candidate in Computer Science and Engineering, Korea University. His research interest includes network security and security policy.

Heejo Lee is an Associate Professor at the Division of Computer and Communication Engineering, Korea University, Seoul, Korea. Before joining Korea University, he was at AhnLab, Inc. as a CTO from 2001 to 2003. From 2000 to 2001, he was a Postdoctorate Researcher at the Department of Computer Science and the security center CERIAS at Purdue University. Dr. Lee received his B.S., M.S., Ph.D. degree in Computer Science and Engineering from POSTECH, Pohang, Korea. Dr. Lee serves as an editor of the Journal of Communications and Networks. He worked on constructing the National CERT in the Philippines (2006) and consulted for the CERTs in Uzbekistan (2007) and Vietnam (2009). He is a visiting professor at CyLab/CMU until December 2010.