PAPER Special Section on Information and Communication System Security

Abnormal Policy Detection and Correction Using Overlapping Transition

Sunghyun KIM[†], *Nonmember and* **Heejo LEE**[†a)], *Member*

SUMMARY Policy in security devices such as firewalls and Network Intrusion Prevention Systems (NIPS) is usually implemented as a sequence of rules. This allows network packets to proceed or to be discarded based on rule's decision. Since attack methods are increasing rapidly, a huge number of security rules are generated and maintained in security devices. Under attack or during heavy traffic, the policy configured wrong creates security holes and prevents the system from deciding quickly whether to allow or deny a packet. Anomalies between the rules occur when there is overlap among the rules. In this paper, we propose a new method to detect anomalies among rules and generate new rules without configuration error in multiple security devices as well as in a single security device. The proposed method cuts the overlap regions among rules into minimum overlap regions and finds the abnormal domain regions of rules' predicates. Classifying rules by the network traffic flow, the proposed method not only reduces computation overhead but blocks unnecessary traffic among distributed devices.

key words: *firewall, security policy, policy anomalies, network security, ACL*

1. Introduction

As new attacks are being created every day, new rules also have been added in security devices. In security devices, the firewall plays crucial role in network traffic management. The basic function of a firewall is to screen network communications to prevent unauthorized access to or from a computer network [1]. The firewall decision to filter out undesired traffic is based on the security policy defined according to a predefined set of rules.

Most firewall's rule sets include a large number of rules and have an order sensitive property. A rule consists of predicates having protocol fields and appropriate action. Based on the rule's action, the firewall allows or denies the packet. Though most firewalls have been improved to handle high speed network traffic and a lot of rules, wrongly configured rules not only downgrade performance, but generate security holes. As Wool observed [2], most firewalls include various types of configuration errors, because rule management is a complicated, complex and error-prone task for network administrators and system managers.

Most rules are order-sensitive, so that the firewall finds the first rule applied to packets among a set of rules. Though there may be no anomaly in the current rules, minor modification of rules' predicates or rules' order may deny normal service or permit attack traffic, which can be the target for a network attack. Also rule redundancy may downgrades performance. Such abnormal problems such as redundancy, shadowing, and spuriousness result from overlap of predicates among rules in a single device or distributed devices.

Based on our observation of network traffic and overlap among rules, we propose a new method to remove abnormal relations among rules and provide new rules without anomaly in distributed firewalls as well as in a single firewall. The proposed method splits overlap regions from rules' predicates and finds abnormal overlap regions among rules. It corrects abnormal relations among rules by removing subsequent rules' overlap regions without changing the meaning of the original rule set. The contribution of this study is as follows:

- The proposed method detects anomalies in overlap regions and generates complete rules without overlap among rules. Since non-overlapping rules have no relation to the other rules, their order is worthless and they can be deleted without influencing others.
- Classifying rules by in-out traffic, we reduced the comparison overhead among rules. In network communication, since incoming traffic is irrelevant to outgoing traffic, it is unnecessary to compare rules for incoming traffic with rules for outgoing traffic.
- The proposed method denies unnecessary network traffic from a source network by detecting and correcting abnormal rules in both sides of traffic flow. Corrected rules make a source network accept traffic as much as a destination network sends and a destination network send traffic as much as a source network accepts.

The remainder of this paper is organized as follows. §2 briefly outlines related work. §3 explains all types of anomalies among rules. §4 describes the proposed method. §5 presents an implemented application and experimental results. In §6, we summarize our experience.

2. Related Work

A firewall is the network equipment that denies or accepts a packet based on rules' policy. As the number of rules has increased, firewall policy can have some abnormal relations among rules in distributed firewalls as well as in a single firewall. There have been many challenges to find

Manuscript received July 9, 2009.
Manuscript revised December 18, 2009.

[†]The authors are with Korea University, Seoul 136–713, South Korea.
a) E-mail: heejo@korea.ac.kr

DOI: 10.1587/transinf.E93.D.1053

such anomalies and to maintain integrity of the security policy configuration.

Lui and Gupda [3]–[5] 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 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.

Hamed et al. [6] provided a taxonomy for conflicts in network security policy. They classified various conflict relations and abnormal types within a single device or among different devices. They implemented the Security Policy Advisor (SPA) tool, which uses Ordered Binary Decision Diagram (OBDD) [7] to present and to manipulate the policy expressions, for automatic discovery of security policy conflicts among IPsec as well as firewalls.

Another useful application for security policy, Firewall Policy Adviser (FPA), was proposed by Al-Shaer et al. [8]–[10]. They presented a set of techniques and algorithms that provide automatic discovery of firewall policy anomalies to reveal rule conflicts and potential problems in legacy firewalls, and anomaly free policy editing for rule insertion, removal, and modification. FPA constructs a policy tree for firewall rules and state diagram for anomaly discovery for rules. Traversing them, FPA searches the misconfigured rules.

Yuan et al. [11] proposed FIREMAN (FIREwall Modeling and ANalysis) implemented by modeling firewall rules using binary decision diagrams (BDDs) that have been used successfully in hardware verification and model checking. FIREMAN performs symbolic model checking of the firewall configurations for all possible IP packets, along all possible data paths. FIREMAN evaluates firewall configuration as an entire set not just limited to relation between two firewall rules.

Alfaro et al. proposed MIRAGE (MIsconfiguRAtion manaGEr) [12], [13] that discovers anomalies in network security policies deployed over firewalls and network intrusion detection systems (NIDSes). In their paper, they consider not only the analysis of relations among rules, but also a complete analysis of the entire set of rules. Pointing out insufficient discovery of abnormal rules in previous researches, they generate completely independent rule set by removing rules' overlap from original rules. They compared all rules in all network path from the network topology.

Tongaonkar et al. [14] proposed a technique that aims to infer the high-level security policy from low-level representation. They generate flattened rules, i.e., rules without priorities, which are equivalent to the given firewall rule set. Since removal of priorities from a rule set generates a lot of rules, they used grouping or merging method to reduce the size and the complexity of the rule set. But it was developed for a single device not for multiple devices.

Algorithms to find misconfigured security policy require high complexity. To reduce the complexity of finding firewall's abnormal rules, Pozo et al. [15] proposed Potential Conflicts Graph (PCG) to diagnose the consistency of firewall rule set. PCG isolates all inconsistencies among every pair of rules and identifies the minimum number of conflicting rules. However, it cannot diagnose the redundancy as they stated. Lu et al. [16] proposed and implemented a comparison method for firewall rule tables to update or modify rules. However, they did not provide the method to check the integrity of the rule set itself.

Among previous researches, MIRAGE was the method for both correction of policy anomaly and distributed firewalls. The proposed method is similar to MIRAGE in that it generates complete rules without anomaly and overlap in distributed firewalls as well as in a single firewall. But the proposed method reduced the number of rules to be compared using rule classification. In a single firewall, since the proposed method classifies rule by in-out traffic, we can avoid to compare a rule for incoming traffic with rules for outgoing traffic.

In distributed firewalls, since all rules are classified according to in-out traffic related to each firewall, we compare a rule within corresponding rule group. Besides, the proposed method is a more complete approach in that it blocks unnecessary traffic from its source network. Corrected rules by the proposed method in both sides allow traffic as much as the source network and destination network attempt to exchange. Unlike the previous methods, the proposed method does not need to search any data path or network path in a network topology. It requires only network addresses in rules. Also it is easy to distinguish which parts of rules have what kinds of problems.

3. Rule Anomaly Problem

We define all abnormal relations among rules in view of the predicates' overlap in this section. Each rule in a firewall has the form <*predicates*> → <*decision*>. <*predicates*> are boolean expressions over packets' protocol fields, such as source IP address, destination IP address, source port number, destination port number, and protocol type. <*decision*> can be "deny" or "accept". As rules have increased, it is hard to maintain integrity and consistency among rules. Table 1 and 2 show all possible relations between one rule and others within a single firewall, intra-policy, and among distributed firewalls, inter-policy. There can be five relations between two rules. Based on these relations, we can find inter-policy anomalies across multiple security devices as well as intra-policy anomalies in a single security device.

We denote one of rules by r_x, r_y and assume that r_x has the precedence over r_y. Any correlation of two rules, $r_x \bigcap r_y \neq \emptyset$, can be represented with three subsets such as $r_x - r_y$, $r_y - r_x$ and $r_y \cap r_x$. 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 $r_y \cap r_x$ is a subset of r_x and r_y both, $r_x \bigcap r_y \neq \emptyset$ can be presented by subsets of completely dis-

Table 1 Intra-policy relations and anomalies. ($r_x[order] < r_y[order]$)

Relation	(a) $r_x = r_y$	(b) $r_x \subsetneq r_y$	(c) $r_x \supsetneq r_y$	(d) $r_x \cap r_y \neq \varnothing$	(e) $r_x \cap r_y = \varnothing$
Permit/Permit	Redu.(r_y)	Part.Redu(r_y)	Redu.(r_y)	Part.Redu.(r_y)	Legitimate
Deny/Deny					
Permit/Deny	Shad.(r_y)	Part.Shad(r_y)	Shad.(r_y)	Part.Shad.(r_y)	Legitimate
Deny/Permit					

Table 2 Inter-policy relations and anomalies. (traffic goes from F_s^x to F_d^y)

Relation	(a)$F_s^x = F_d^y$	(b)$F_s^x \subsetneq F_d^y$	(c)$F_s^x \supsetneq F_d^y$	(d)$F_s^x \cap F_d^y \neq \varnothing$	(e)$F_s^x \cap F_d^y = \varnothing$
Permit/Permit	Legitimate	Part.Shad.(F_d^y)	Part.Spur.(F_s^x)	Part. Spur.(F_s^x) Part. Shad.(F_d^y)	Spur.(F_s^x) Shad.(F_d^y)
Deny/Deny	Redu.(F_d^y)	Part.Redu.(F_d^y)	Redu.(F_d^y)	Part. Spur.(F_s^x) Part. Redu.(F_d^y)	Legitimate
Permit/Deny	Spur.(F_s^x)	Spur.(F_s^x)	Part.Spur.(F_s^x)	Part. Spur.(F_s^x) Part. Redu.(F_d^y)	Redu.(F_d^y)
Deny/Permit	Shad.(F_d^y)	Part.Shad.(F_d^y)	Shad.(F_d^y)	Part.Shad.(F_d^y)	Legitimate

joint sebset and exactly matching sebset. For the same reason, inclusive relation, $r_x \supsetneq r_y$ or $r_x \subsetneq r_y$, can be presented with completely disjoint subset and exactly matching subset. Therefore, all relations of two rules can be presented by exactly matched relation and completely disjoint relation. There is little difference in terms of defining conflicts or anomalies [3,4,9,10,13]. Based on previous researches, we redefined policy anomaly simply and clearly in this section.

3.1 Intra-Policy Anomaly

Intra-policy anomalies occur among rules in a single security device. Table 1 shows all anomalies in a single firewall. Overlaps among rules make abnormal relations. That is, if there was no overlap among rules, anomalies cannot occur except for irrelevance that is irrelevant with device's traffic. If we present correlation and inclusive relation among rules with exactly matched subset and completely disjoint subset, anomalies occur only in Table 1's (a). When r_x and r_y are exactly matched, such as $r_x = r_y$, anomalies like the followings occur:

Intra-shadowing This 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. For example, when r_x denies a packet but r_y allows that packet, r_y is useless.

Intra-redundancy This 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. For example, when r_x denies a packet and r_y also denies that packet, r_y is useless.

3.2 Inter-Policy Anomaly

Inter-policy anomalies occur among rules in distributed security devices. We define a zone as a network address directly connected to the security device. Let F_s denote the source zone security device and F_s^x denote one rule of F_s. Let F_d denote the destination zone security device and F_d^y denote one rule of F_d. We assume that network traffic flows from F_s to F_d and that each device has the default policy or the device policy. Table 2 shows all anomalies among rules between F_s^x and F_d^y. If we represent correlation and inclusive relation among rules with exactly matched subset and completely disjoint subset, anomalies occur in Table 2's (a) and (e). Unlike intra-policy, overlaps among rules can be a normal or abnormal relations depending on rules' action such as the following:

Inter-shadowing This happens when the source device with F_s^x blocks any packet but the destination device with F_d^y allows the packet. F_d^y is unnecessary.

Inter-redundancy This happens when the source device with F_s^x blocks any packet but the destination device with F_d^y blocks that packet again. F_d^y is unnecessary.

Inter-spuriousness This happens when the source device with F_s^x allows any packet but the destination device with F_d^y blocks that packet. F_s^x is unnecessary.

4. Resolving Policy Anomaly and Overlaps

We propose a new method to solve the policy anomaly based on set theory. As described above, we represent all relations among rules with exactly matched relation or completely disjoint relation. For that purpose, we devised an array data structure, termed RPA (Rule Predicate Array). Each rule within RPA has only exactly matched or unrelated relation one another. RPA is used to find and correct the policy anomaly.

4.1 Rule Predicates Array

In a single firewall, given a set of rules, i.e., $R = \{r_1, r_2, \ldots r_n\}$, let $F = \{f_1, f_2, \ldots, f_m\}$ denote the set of m protocol fields presented in R. Let $P_{f_i} = \{p_{f_i}^1, p_{f_i}^2, p_{f_i}^3, \ldots.\}$ denote the set of the predicates associated with f_i in R. $V_{f_i} = \{v_{f_i}^1, v_{f_i}^2, v_{f_i}^3, \ldots\}$ denotes 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:

$$r_x = p^i_{f_1} \wedge p^j_{f_2} \wedge \ldots . p^k_{f_m} \quad (1)$$

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 predicate(=,≤,≥,etc.). Therefore, Eq. (1) can be described as following:

$$r_x = f_1 \otimes v^i_{f_1} \wedge f_2 \otimes v^j_{f_2} \wedge \ldots f_m \otimes v^k_{f_m} \quad (2)$$

As seen in Eq. (2), 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:

$$\begin{aligned} dom(f_i) = \{&v^1_{f_i} > d^1_{f_i}, d^2_{f_i} = v^1_{f_i}, v^1_{f_i} < d^3_{f_i} < v^2_{f_i}, \\ &d^4_{f_i} = v^2_{f_i}, \ldots, d^{2k}_{f_i} = v^k_{f_i}, v^k_{f_i} < d^{2k+1}_{f_i}\} \end{aligned} \quad (3)$$

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

$$\alpha = s^i_{f_1} \;\&\; s^j_{f_2} \;\&\; \cdots \;\&\; s^k_{f_m} \quad (4)$$

In distributed firewalls, when one firewall has a rule set, R, we can find rules related with R in the other firewalls. We denote them $W = \{w_1, w_2, \ldots w_p\}$. After gathering comparative values from predicates used in R and W, we split domain regions of each protocol field like Eq. (3). In one of f_i's domain regions, we denote the domain bitmap to represent the result of each rule in W by $t^j_{f_i}$, i.e., $t^j_{f_i}$=$x_1 \cdot x_2 \cdot x_3 \ldots \cdot x_p$, where p is the number of rules in W. Let $T_{f_i} = \{t^1_{f_i}, t^2_{f_i}, \ldots, t^{2k+1}_{f_i}\}$ denote the set of W's domain bitmap in each region of $dom(f_i)$. T_{f_i} can be pre-computed by the same way to obtain S_{f_i} in all split regions of $dom(f_i)$. Let β denote one of result bitmaps in W. In a similar way to obtain α like Eq. (4), β can be obtained from each domain bitmap in all protocol fields' domain region as following:

$$\beta = t^i_{f_1} \;\&\; t^j_{f_2} \;\&\; \cdots \;\&\; t^k_{f_m} \quad (5)$$

Rule Predicates Array (RPA) is an array data structure holding the result bitmaps according to divided domain regions of a protocol field f_i. We described the structure of RPA in Fig. 1 and defined RPA in Definition 1.

rif	$d^1_{f_i}$	$d^2_{f_i}$	$d^3_{f_i}$	$d^4_{f_i}$	...	$d^{2k}_{f_i}$	$d^{2k+1}_{f_i}$
zdb	$s^1_{f_i}$	$s^2_{f_i}$	$s^3_{f_i}$	$s^4_{f_i}$	...	$s^{2k}_{f_i}$	$s^{2k+1}_{f_i}$
odb	$t^1_{f_i}$	$t^2_{f_i}$	$t^3_{f_i}$	$t^4_{f_i}$	...	$t^{2k}_{f_i}$	$t^{2k+1}_{f_i}$

Fig. 1 Structure of RPA (Rule Predicate Array).

Table 3 Eample of firewall rules.

ID	SIP	SP	DIP	DP	Act.
r_1	1.1.1.4	*	2.1.1.[1-255]	*	A
r_2	*	*	2.1.1.4	*	A
r_3	1.1.1.[1-4]	*	2.1.1.4	80	D
r_4	1.1.1.100	[1-1024]	2.1.1.[1-255]	*	D

Definition 1 (**Rule Predicates Array**): Given a set of rules R=$\{r_1, r_2, \ldots r_n\}$ and W=$\{w_1, w_2, \ldots w_p\}$, let V_{f_i} denote a set of distinct comparative values extracted from all predicates of a protocol fields f_i in R and W, i.e. $V_{f_i} = \{v^1_{f_i}, v^2_{f_i}, \ldots, v^k_{f_i}\}$. Rule Predicates Array for f_i RPA_{f_i} with k distinct constants is defined as an array of $(2k+1)$ regions. Each RPA has following entries:

• **Region identifier(*rif*)**: An identifier indicates whether its corresponding region is a constant region or an interval region. If $RPA_{f_i}[j]$, the j^{th} of RPA_{f_i}, is constant, it holds one element of V_{f_i}. Otherwise, $RPA_{f_i}[j].rif = null$, implying that $RPA_{f_i}[j-1].rif < RPA_{f_i}[j].rif < RPA_{f_i}[j+1].rif$.

• **Zone domain bitmap(*zdb[1..n]*)**: A bitmap stores the pre-computed R's result of its corresponding domain region. In the j^{th} region $RPA_{f_i}[j]$, the k^{th} bit of the bitmap is set to *0*, i.e., $RPA_{f_i}[j].zdb[k] = 0$, if r_k's predicate for f_i whose comparative value falls within the region $RPA_{f_i}[j]$ cannot satisfy the predicates for f_i. Otherwise, the k^{th} bit is set to *1*. i.e., $RPA_{f_i}[j].zdb[k] = 1$.

• **Others domain bitmap(*odb[1..p]*)**: A bitmap stores the pre-computed W's result of its corresponding domain region. It is set by the same way as zone domain bitmap.

4.2 Intra-Policy Detection and Correction

Due to their deployment, the security device can see only two types of network traffic. One is incoming traffic, which has home network in source address field and has external network address in destination address field. The other is outgoing traffic which has the opposite source and destination address against incoming traffic. Therefore, we classified all rules into two groups, rules for incoming traffic and rules for outgoing traffic. We denote them by R_{in} and R_{out}, respectively. If there is any rule not included in R_{in} and R_{out}, we detect it as an irrelevance anomaly, which is caused by the rule that is irrelevant to traffic of the device. After classifying rules, we check the integrity of R_{in} and R_{out}, respectively. That results in reduction of the number of rules to be compared.

We explain only the detection and correction process for outgoing rules because the other is the same process. We

have four outgoing rules in Table 3. Using the rules' predicate, we made four RPAs for each protocol field, such as source address (SIP), destination address (DIP), source port (SP), and destination port (DP). As described Algorithm 1, we extract distinct comparative values from the rules' predicates and divide domain like Eq. (3). Each zone domain bitmap is set by $SetBitmap$ which returns a domain bitmap of R_{out} in certain domain region. In the algorithm, $v^j_{f_i} + 1$ means that interval region between $v^j_{f_i}$ and $v^{j+1}_{f_i}$. We merge the regions having the same bitmap to reduce the array size. Rules share or exclusively possess certain domain regions in RPA. Since RPA cuts the domain of each protocol field into the minimum region, all rules have only two cases, overlap and non-overlap. Therefore, correlation and inclusive relation among rules are removed in RPA.

For example, RPA_{SP} in Fig. 2 is created as follows. First, we gather the predicates' comparative values from rules like 1, 1024, and 65535 because "any" means all domain regions of source port, from 1 to 65535. Therefore, $dom(SP)$ is divided into three regions like 1-1023, 1024, and 1025-65535. Since all rules' predicates are matched in the domain region 1-1023 and 1024, their bitmap have "1111". But since r_4's predicate is not matched in the domain region 1024-65535, its bitmap has "1110". Two regions are merged because the bitmaps in 1-1023 and 1024 are the same.

Algorithm 2 shows the process of the anomaly detection and correction of each rule. From the conjunctive combination of each zone domain bitmap in all RPAs, as Eq. (4) means, we can find which rules apply to which domain regions of the protocol fields in a result bitmap. 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 r_x's result bitmap is not x, the preceding rule is applied in that domain region because of the priority among rules. Otherwise, r_x is the first rule to be applied to that domain region. Finding the first rule to be applied in r_x's domain region, we split r_x into normal rules and abnormal rules. Split rules are merged to reduce the number of rules. We merge two rules when result bitmaps are the same and all protocol field's domain regions except one are the same and the exceptional one is consecutive each other.

Algorithm 1 CreateIntraRPA(f_i, R_{out})

Require: $R_{out} = \{r_1, r_2, ..r_n\}$
Ensure: RPA_{f_i}
  Extract $V_{f_i} = \{v^i_{f_1}, v^j_{f_2}, \ldots v^k_{f_m}\}$ from R_{out}
  $k \leftarrow |V_{f_i}|$
  Create $(2k+1)$'s size of RPA for f_i
  $v^0_{f_i} \Leftarrow 0$
  for $j = 0$ to k **do**
    if $v^j_{f_i} \neq v^{j-1}_{f_i} + 1$ **then**
      $RPA_{f_i}[2j].rif \Leftarrow v^j_{f_i}$
      $RPA_{f_i}[2j+1].rif \Leftarrow null$
      $RPA_{f_i}[2j].zdb \Leftarrow$ SetBitmap$(v^j_{f_i}, R_{out})$
      $RPA_{f_i}[2j+1].zdb \Leftarrow$ SetBitmap$(v^j_{f_i} + 1, R_{out})$
    end if
  end for

Algorithm 2 ResolveIntraPolicy(r_x)

Require: $r_x : f_1 \otimes v^i_{f_1} \wedge f_2 \otimes v^j_{f_2} \wedge \ldots f_m \otimes v^k_{f_m}$
Ensure: NL : linked list of normal rules, AL : linked list of abnormal rules
  Extract $v^i_{f_1}, v^j_{f_2}, \ldots v^k_{f_m}$ from r_x
  Create m-element of Array, $Azdb$ and $VT = \{v^i_{f_1}, v^j_{f_2}, \ldots v^k_{f_m}\}$
  for $t = 0$ to m **do**
    $Azdb[t] \Leftarrow RPA_{f_i}[VT[t]].zdb)$
    $rbmap \Leftarrow rbmap$ & $Azdb[t]$
  end for
  if GetHighestNonzeroPosition($rbmap$) = x **then**
    InsertList($NL, VT, rbmap$)
  else
    InsertList($AL, VT, rbmap$)
  end if

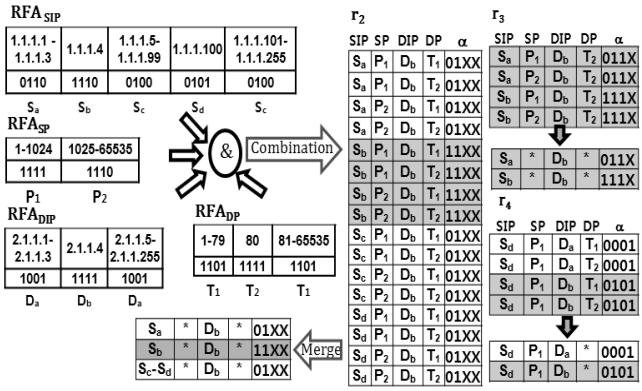

Fig. 2 Anomaly detection and correction of abnormal rules using RPAs for rules in Table 3. ("X" means "Don't Care Bit" and colored rows have anomaly.)

Table 4 Rewritten rules without overlap.

ID	SIP	SP	DIP	DP	Act.
r_1	1.1.1.4	*	21.1.1.[1-255]	*	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	1.1.1.100	[1-1024]	2.1.1.[1-3]	*	D
	1.1.1.100	[1-1024]	2.1.1.[5-255]	*	D

Fig. 2 also shows the process of anomaly detection and correction for rules in Table 3. We exclude r_1 because r_1 is the highest priority rule. In the case of r_4, RPA_{SIP}.zdb is "0101", RPA_{SP}.zdb is "1111", and RPA_{DIP}.zdb is "1001" or "1111". Therefore, we obtain four result bitmaps. In the upper two rules and the lower two rules, three values are the same and one value is consecutive with the other, they can be merged. In merged result, "0001" has not any problem because r_4 is the first rule to be applied in that domain region. But "0101" has a shadowing problem because r_2 and r_4 apply the same domain region with different decisions. Likewise, R_3 has a redundant anomaly in all domain regions. We can rewrite R_2, R_3, and R_4 with rows which have corresponding rule applied for the first time. The new generated rules are corrected from original rules without overlap and anomaly. Final results are presented in Table 4. If there are

Table 5 Final rules without anomaly.

ID	SIP	SP	DIP	DP	Act.
r_1	1.1.1.100	[1-1024]	2.1.1.[1-3]	*	D
r_2	1.1.1.100	[1-1024]	2.1.1.[5-255]	*	D
r_3	1.1.1.[1-255]	*	2.1.1.[1-255]	*	A

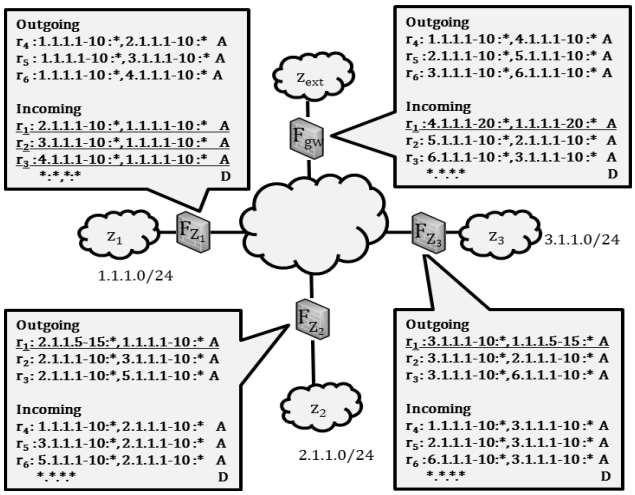

Fig. 3 Example of firewall deployment.

rules with different action, we choose only rules with the same action and add default rule, as in Table 5. The proposed method creates many rules because it splits rules to make non-overlapping states. Our method for distributed security devices is similar to the intra-policy method.

4.3 Inter-Policy Detection and Correction

Anomaly detection and correction for multiple devices is a little more complicated than for a single device. As you can see in Table 2, we have to consider not only rules' decision but default decision or device policy. A "zone" is network address directly connected to the firewall's interface and represent firewall's subnetwork. We assume that one firewall can have multiple zones, but a zone is allocated to one firewall. We classify firewalls into two types by network flow. One is the gateway firewall which is charged with entire network traffic between the external network and the internal network. The other is the zone firewall which is charged with the subnetwork traffic between its own zone and the other zones or the external zone.

We assume that each firewall has non-overlapping rules due to our intra-policy method. All rules have the same decision and are classified into incoming rules and outgoing rules. Figure 3 shows a simple example of a network diagram deploying three zone firewalls and one gateway firewall. For simplicity, we generalize port predicates using "any" type. All rules have "accept" action after applying our intra-policy method. We denote each zone in the network by $z_1, z_2, .., z_n$. Also let z_{int} denote the internal network address and z_{ext} denote the external network address. $F_{z_1}, F_{z_2}, \ldots, F_{z_n}$ denote the set of packets allowed in each zone firewall. We denote the set of packets allowed in gateway firewall by F_{gw}. Let $F_{z_j}^{d=z_i}$ denote the traffic set heading for z_i in F_{z_j}. Likewise, $F_{gw}^{d=z_i}$ denote the traffic set heading for z_i in F_{gw}. $F_{z_j}^{d=z_i}$ is outgoing traffic in F_{z_j}, while $F_{gw}^{d=z_i}$ is internal traffic in F_{gw}. The ideal state is that the other firewalls have to permit as many as packets that are accepted by z_i. That is, the zone firewall z_i's incoming traffic is the same as the other firewalls' outgoing traffic heading for z_i as follows:

$$F_{z_i}^{d=z_i} = \sum_{j=1}^{n} F_{z_j}^{d=z_i} + F_{gw}^{d=z_i} (i \neq j) \tag{6}$$

The gateway firewall ought to have all zone firewall's traffic heading for internal and external addresses as follows:

$$F_{gw}^{d=z_{ext}} = \sum_{j=1}^{n} F_{z_j}^{d=z_{ext}}, F_{gw}^{d=z_{int}} = \sum_{j=1}^{n} F_{z_j}^{d=z_{int}} \tag{7}$$

In a firewall F_{z_i}, let W_{out} denote the set of the other firewalls' rules heading for F_{z_i}, i.e. $W_{out} = \{w_1, w_2, ..w_p\}$. As Algorithm 3 describes, RPAs for F_{z_i} are created with one firewall's incoming rules(R_{in}) and the other firewalls' rules heading for F_{z_i} (W_{out}). Just as in the intra-policy method,

Algorithm 3 CreateInterRPA(R_{in}, W_{out})

Require: $R_{in} = \{r_1, r_2, ..r_n\}$, $W_{out} = \{w_1, w_2, ..w_p\}$
Ensure: RPA_{f_i}
  Extract $V_{f_i} = \{v_{f_1}^i, v_{f_2}^j, \ldots v_{f_m}^k\}$ from R_{in} and W_{out}
  $k \leftarrow |V_{f_i}|$
  Create $(2k+1)$'s size of RPA for f_i
  $v_{f_i}^0 \Leftarrow 0$
  for $j = 0$ to k **do**
    if $v_{f_i}^j \neq v_{f_i}^{j-1} + 1$ **then**
      $RPA_{f_i}[2j].rif \Leftarrow v_{f_i}^j$
      $RPA_{f_i}[2j+1].rif \Leftarrow null$
      $RPA_{f_i}[2j].zdb \Leftarrow \text{SetBitmap}(v_{f_i}^j, R_{in})$
      $RPA_{f_i}[2j+1].zdb \Leftarrow \text{SetBitmap}(v_{f_i}^j + 1, R_{in})$
      $RPA_{f_i}[2j].odb \Leftarrow \text{SetBitmap}(v_{f_i}^j, W_{out})$
      $RPA_{f_i}[2j+1].odb \Leftarrow \text{SetBitmap}(v_{f_i}^j + 1, W_{out})$
    end if
  end for

domain region of each protocol field are sliced into a minimum overlap in RPAs as Eq. (3) describes. However, since RPAs are created by two rule sets, R_{in} and W_{out}, RPAs for F_{z_i} have two domain bitmaps in each domain region, one for F_{z_i} and one for the other firewalls. From Eq. (4) and Eq. (5), we obtain two result bitmaps in each domain region as Algorithm 4 represents. Comparing them, we can find which firewall has anomaly. As a result, F_{z_i}'s incoming rules are compared with the other firewalls' outgoing rules heading for F_{z_i} in each split domain region as Eq. (6) describes. Likewise, RPAs for F_{gw} can be used to detect and correct anomaly among gateway firewall and zone firewalls as Eq. (7) describes.

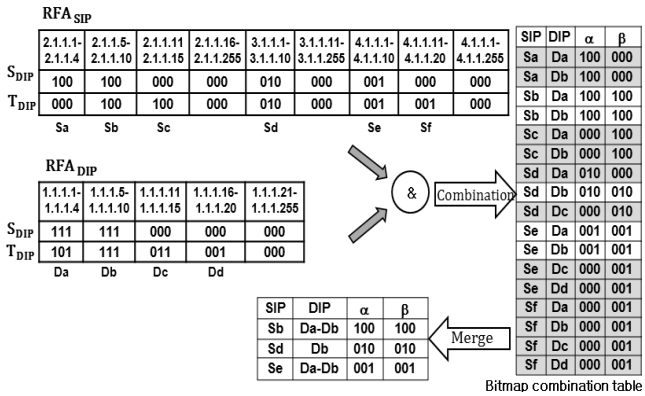

Fig. 4 Detection and correction of abnormal rules among one zone firewall and the other firewalls. (Colored rows have anomaly.)

Fig. 4 presents the process of detection and correction of F_{z_1}'s incoming rules and the other firewalls' outgoing rules heading for F_{z_1} in Fig. 3. Since the rules' policy is "accept", the normal case is that both F_{z_1}'s result bitmap(α) and the other firewalls' result bitmap(β) must have the same bitmap state; all zero bits or only one bit set to 1. All zero bits means that since there is no rule in that domain region, the default policy is applied. From the first row of bitmap combination table in Fig. 4, we know that F_{z_1}'s r_1 has shadowing anomaly because a packet denied by other firewall is allowed by F_{z_1}'s r_1 in source addresses [2.1.1.1 − 4] and destination address [1.1.1.1 − 4]. Likewise, if only the other firewalls' result bitmap has a bit set to 1, it means spuriousness anomaly because a packet allowed by other firewall is denied by F_{z_1}. Since the rules' policy is "accept", there is no redundancy among rules as described Table 2.

We distinguish normal rows or abnormal rows from

Algorithm 4 ResolveInterPolicy(*GRPA*)

Require: $GRPA = \{RPA_{f_1}, RPA_{f_2}, \ldots, RPA_{f_m}\}$
Ensure: NL : linked list of normal rules, AL : linked list of abnormal rules
Create m-element of Array, $Asize[1..m] = \{|RPA_{f_1}|, |RPA_{f_2}|, \ldots, |RPA_{f_m}|\}$
for $i = 0$ to $Asize[1]$ **do**
  for $j = 0$ to $Asize[2]$ **do**
    ...
    for $k = 0$ to $Asize[m]$ **do**
      $zbmap = RPA_{f_1}[i].zdb$ & $RPA_{f_2}[j].zdb$ & ... & $RPA_{f_m}[k], zdb$
      $obmap = RPA_{f_1}[i].odb$ & $RPA_{f_2}[j].odb$ & ... & $RPA_{f_m}[k], odb$
      if $zbmap \neq 0$ or $obmap \neq 0$ **then**
        if $zbmap \neq 0$ and $obmap \neq 0$ **then**
          InsertList($NL, RPA_{f_1}[i].rif, RPA_{f_2}[j].rif, \ldots, RPA_{f_m}[k], rif, zbmap, obmap$)
        else
          InsertList($AL, RPA_{f_1}[i].rif, RPA_{f_2}[j].rif, \ldots, RPA_{f_m}[k], rif, zbmap, obmap$)
        end if
      end if
    end for
    ...
  end for
end for

Table 6 Final rules without abnormal relations.

ID	SIP	SP	DIP	DP	Act.
F_{z_1}'s r_1	2.1.1.[5-10]	*	1.1.1.[1-10]	*	A
F_{z_1}'s r_2	3.1.1.[1-10]	*	1.1.1.[5-10]	*	A
F_{z_1}'s r_3	4.1.1.[1-10]	*	1.1.1.[1-10]	*	A
F_{z_2}'s r_1	2.1.1.[5-10]	*	1.1.1.[1-10]	*	A
F_{z_3}'s r_1	3.1.1.[1-10]	*	1.1.1.[5-10]	*	A
F_{gw}'s r_1	4.1.1.[1-10]	*	1.1.1.[1-10]	*	A

combination with each domain region in RPAs. Since this process splits rules, we merge split rules to reduce the size of rules as the intra-policy method does. F_{z_1}'s incoming rules and the other firewalls outgoing rules heading for F_{z_1} are newly rewritten as in Table 6. We can obtain complete and consistent rules if we repeat this process in all firewalls. These rules allow only network traffic as much as the source and destination network attempt to exchange.

5. Implementation and Experiments

We implemented the proposed method in a software prototype called PAR (Policy Anomaly Resolver) to verify in a real network environment. PAR has been developed using C^{++} on a Windows XP machine (1 Gbyte Memory and Core 2 2.13 GHz CPU). PAR parses ACL rules and forms four RPAs as described above. We used two different ACL rule sets from one Korea online game company's two different internal switches for our experiments. Table 7 shows the characteristics of two ACL rule sets in detail. "ACL1" is rules for outgoing traffic while "ACL2" is rules for incoming traffic. "#SIP, #SP, #DIP, and #DP" means that the number of distinct comparative values extracted from the predicates of corresponding protocol fields.

The experiments for intra-policy method are presented in Table 8. Our intention is to show the effectiveness of the proposed method in view of two aspects, detection and correction. First, we found how many anomalies were existed in rule sets. Since overlaps occur the anomaly, "ACL2" is much better managed rule set than "ACL1". In "ACL1" which has 3770 overlaps, 40 rules were removed because they had complete anomaly. But in "ACL2" which has 9 overlaps, no rule had complete anomaly. Next, when overlaps among rules were removed, "ACL1" generated 68% more rules than original rules and one rule was split up to 13 new rules at maximum. On the other hand, "ACL2" generated only 2.3% more rules and one rule was split only 2 at most. Depending on degree of overlaps, the number of new generated rules were various but they were acceptable degree.

The performance of the proposed method was closely related to overlaps among rules. The execution time in Table 8 means the sum of each rule's execution time. In aspect of performance, the size of RPA, which is the number of split domain region of protocol field, does not have great influence on execution time. The major factors for performance are the number of overlaps and the boundary of overlaps among rules. The number of overlaps affects

Table 7 The number of rules and distinct values of protocol fields in two ACL files.

ID	rules	#SIP	#SP	#DIP	#DP	overlap
ACL1	277	38	45	95	13	3770
ACL2	388	191	15	62	21	9

Table 8 Results of detection and correction for intra-policy.

ID	exec. (sec.)	# rewritten rules	# removed rules	# max. split rules
ACL1	3	439	40	13
ACL2	1.1	395	0	2

the number of split rules because the subsequent rules are split to avoid overlap. Likewise, the boundary of overlaps affects both the number of combined results and the number of split rules. For example, if all predicates of a rule have only a constant IP address and port number, the combined result has only one. But if a predicate of a rule has wide IP addresses or port numbers like "any", which covers all the domain regions of the corresponding protocol field, the combined results are generated as many as the number of split domain regions. "ACL1" includes 723 rules having one "any" predicate, 95 rules having two "any" predicates, and 6 rules having three "any" predicates. "ACL2" includes 193 rules having one "any" predicate, 30 rules having two "any" predicates, and 3 rules having three "any" predicates. As a result, "ACL1", which has 40% less rules than "ACL2", needed about 3 times process time because of overlaps.

6. Conclusions

Security policy has a critical role for network protection. Policy maintenance is a complex and error-prone task, as recognized by many people. The policy anomaly problem is caused from overlap among rules. We proposed a new method not only to detect rule anomaly but also to rewrite new rules. The proposed method removes all anomalies among rules in multiple devices as well as in a single device. It uses a special array data structure which replaces inclusive overlap and correlated overlap with exactly matched overlap or non-overlap.

We implemented the proposed method into application called PAR (Policy Anomaly Resolver) and tested real rule sets. Since PAR can produce new rules without order dependency and anomaly, deletion or order change of certain rule does not effect the other rules. In case of rule addition or rule modification, we can check whether it causes any problem with the other rules in advance. Since the proposed method resize rules as much as source and destination need, the proposed method blocks unnecessary traffic and allows traffic as much as source and destination attempt to exchange. The proposed method supports consistency and completeness but not compactness in firewall design principals. Besides, it has disadvantages in processing time and memory requirement for a large scale rules due to the combination of divided domain regions. Therefore, we are studying ways to reduce the complexity and the size of rules to be rewritten.

Acknowledgments

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 National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development and Experts Education. Additionally supported by the IT R&D program of MKE/KEIT (2008-S-026-02, The Development of Active Detection and Response Technology against Botnet).

References

[1] R. Strasberg and Gondek, The Compelete Reference Firewalls, MacGrawHill, Nov. 2002.
[2] A. Wool, "A quantitative study of firewall configuration errors," Computer, vol.37, no.6, pp.62–67, June 2004.
[3] M.G. Gouda and X.Y.A. Liu, "Firewall design: Consistency, completeness, and compactness," 24th Int. Conf. Distributed Computing Systems (ICDCS), 2004.
[4] A.X.L. Mohamed and G. Gouda, "Structured firewall design," Comput. Netw., vol.51, no.4, pp.1106–1120, 2007.
[5] A.X. Liu and M.G. Gouda, "Diverse firewall design," IEEE Trans. Parallel Distrib. Syst., vol.19, no.9, pp.1237–1251, Sept 2008.
[6] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," IEEE Commun. Mag., vol.44, no.3, pp.134–141, 2006.
[7] R. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.35, no.8, pp.677–691, Aug. 1986.
[8] E.S. Al-Shaer and H.H. Hamed, "Modeling and management of firewall policies," IEEE eTrans. Netw. Serv. Manage., vol.1-1, April 2004.
[9] R.B. Ehab Al-Shaer, H. Hamed, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," IEEE J. Sel. Areas Commun., vol.23, no.10, pp.2069–2084, 2005.
[10] E.S. Al-Shaer and H.H. Hamed, "Discovery of policy anomalies in distributed firewalls," IEEE INFOCOM, pp.2605–2616, March 2004.
[11] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "Fireman: A toolkit for firewall modeling and analysis," IEEE Symp. Security and Privacy, 2006.
[12] F.C.J.G. Alfaro and N. Boulahia-Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," Int. J. Infomation Security, pp.103–122, Oct. 2008.
[13] J.A.F. Cuppens and N. Cuppens-Boulahia, "Detection and removal of firewall misconfiguration," Network and Information Security, pp.154–162, 2005.
[14] N.I. Alok Tongaonkar and R. Sekar, "Inferring higher level policies from firewall rules," Proc. 21st conference on Large Installation System Administration Conference, 2007.
[15] R.M.G.S. Pozo and R. Ceballos, "Fast algorithms for consistency-based diagnosis of firewall rule sets," Third International Conference on Availability, Reliability and Security, 2006.
[16] J.H.L. Lu, R. Safavi-Naini, and W. Susilo, "Comparing and debugging firewall rule tables," Int. J. Infomation Security, pp.143–151, Dec. 2007.

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 at the Department of Computer Sciences and the security center CERIAS, Purdue University. Dr. Lee received his B.S., M.S., Ph.D. degrees in Computer Science and Engineering from POSTECH, Pohang, Korea. Dr. Lee serves as an editor of the Journal of Communications and Networks. He has been an advisory member of Korea Information Security Agency and Korea Supreme Prosecutor's Office. With the support of Korean government, he worked on constructing the National CERT in the Philippines (2006) and consultation on cyber security in Uzbekistan (2007) and Vietnam (2009). More information is available at http://ccs.korea.ac.kr