

Firewall Design: Understandable, Designable and Testable

Yan-Ning Huang

Department of Computer Sciences
Graduate School at Shenzhen, Tsinghua University
The University Town, Shenzhen, China 518055
Email: huangyn99@hotmail.com

Yong Jiang

Department of Computer Sciences
Graduate School at Shenzhen, Tsinghua University
The University Town, Shenzhen, China 518055
Email: jiangy@sz.tsinghua.edu.cn

Abstract—Firewalls are the cornerstones of network security. To make firewalls working effectively, firewall manager must design firewall rules and the rule order correctly. In this paper, we present a firewall management toolkit which makes firewall rules understandable, designable and testable. Understandable means that the rules shown to the manager are easily understood. Designable means that it is no need to design the rule order when modifying the firewall rules. Testable means that firewall rules can be tested without other device. Our method is based on security policy diagram (SPD, for short). We then apply a sequence of algorithms to generate corresponding firewall policy from SPD to be understood, designed and tested. The firewall management toolkit significantly simplifies the management of any generic firewall policy written as filtering rules.

I. INTRODUCTION

Firewalls are the cornerstones of network security. The function of firewall is to provide secure access to and from the private network. Once a firewall is acquired, a security administrator has to configure and manage it to realize a correct security policy for the particular requirements. A firewall security policy is a list of ordered filtering rules; each rule of the form

$$\langle \text{condition} \rangle \rightarrow \langle \text{action} \rangle$$

where the $\langle \text{condition} \rangle$ is composed of set of different filtering fields. The filtering fields of a rule represent the possible values of the corresponding fields in actual network traffic that matches this rule. The $\langle \text{action} \rangle$ is either “a” (for accept), “d” (for discard) or “u” (for undetermined).

It is a crucial task to design correct firewall policy written as filtering rules. While reading or designing the firewall policy, security administrator must consider not only the meaning of each rule but also the order of the rules. The effect of a rule in a firewall policy is relative to that of each previous rule.

Definition 1.1: A firewall policy is a *no-order-policy* if exchanging any rule in the rules with another will not influence the filtering result of the firewall.

Definition 1.2: A firewall policy is an *ordered-policy* if exchanging some rule in the rules with another may influence the filtering result of the firewall.

A firewall policy has three functions. First of all, it is the filtering rules which match the packets to decide the action of them. The second function is to display the filtering result

of the firewall policy in front of the administrator. At last, the firewall policy is required to be edited to realize security object.

Also the management of security policy needs three steps. The first step is to understand the filtering effect of the existing rules. The second step is designing the security rules. In the end, the administrator should review them to ensure the correctness.

Our method is to combine the advantages of *no-order-policy* and *ordered-policy*. we realize the three firewall policy function with three synchronous security policy. The method is based on security policy diagram (SPD, for short) which is a *no-order-policy* maintaining the filtering results. Other policies are generated from SPD. We then apply a sequence of algorithms to this SPD to be understood, designed and tested.

This paper is organized as follows. In section 2, we give a summary of related work. In section 3, we introduce the security policy diagram (SPD). Section 4 covers the understandable firewall rules generated from FDD. In section 5, we discuss how to edit the rules by the toolkit. In section 6, we mention the test of the security policy, and we conclude with Section 7.

II. RELATED WORK

Although there have been many researches on firewall management, most of them was not dedicated to the problem of how to alleviate the difficulty in the management of security policy. The emphasis was mostly on the filtering performance issues [4],[5],[8].

Another research direction in the area of firewall design has been dedicated to using a high-level policy language to define and analyze firewall policies and then map the language to filtering rules. Such as [3],[9].

Perhaps the research direction that is closest to the spirit of the current paper is reported in [7],[6],[1]. [7],[6] assist the administrator by finding conflicts from an existing firewall policy. [1] design a firewall decision diagram and apply a sequence of algorithms to generate the filtering rules. However, these methods do not deal with alleviate the management–understanding, designing and testing–of the rules at the same time. The rules in are *ordered-policies* which is difficult to understand. If the structure of the firewall decision diagram

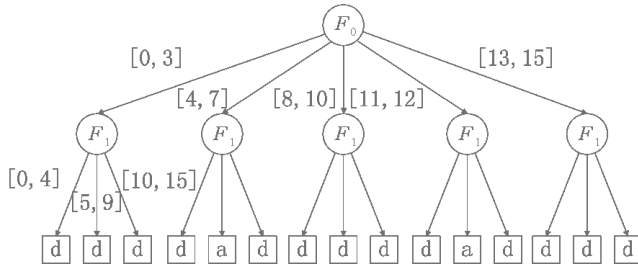


Fig. 1. An SPD

changes, the administrator must design a new firewall policy in [1].

According to the firewall toolkit in the current paper, the rules are based on security policy diagrams. These policy diagrams are similar, but not identical, to other types of decision diagrams such as the Binary Decision Diagrams in [12], the Interval Decision Diagrams in [13] and the Firewall Decision Diagrams in [1].

III. SECURITY POLICY DIAGRAM (SPD)

The $\langle condition \rangle$ of a rule is an n -tuple (p_0, \dots, p_{n-1}) where each p_i is taken from the domain $D(F_i)$ of the corresponding field F_i .

A *security policy diagram* f (or SPD f , for short) over the fields F_0, \dots, F_{n-1} is a tree that satisfies the following four conditions:

1) Each node v in f has a label $F(v)$. If the node is non-terminal, $F(v)$ is taken from the set of fields F_0, \dots, F_{n-1} . If the node is terminal, $F(v)$ is either accept (or “a” for short) or discard (or “d” for short).

2) Each edge e , that is outgoing of a node v in f , is labelled with an integer set $I(e)$, where $I(e)$ is a subset of the domain of field $F(v)$.

3) Let v be any nonterminal node v in f . The set $E(v)$ of all outgoing edges of node v satisfies the following two conditions:

a) *Consistency*: For any distinct e_i and e_j in $E(v)$,

$$I(e_i) \cap I(e_j) = \phi$$

b) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$

4) Let v_1 and v_2 are different nonterminal nodes in f . if $F(v_1) = F(v_2)$, then v_1 and v_2 are *semi-isomorphic*.

Definition 3.1 (Semi-isomorphic Nodes): Let v_1 and v_2 be two different nonterminal nodes in f . The two nodes v_1 and v_2 are *semi-isomorphic* iff they have the same label and there exists a one-to-one mapping σ from the outgoing edges of v_1 to those of v_2 such that for each outgoing edge e of v_1 , $I(e) = I(\sigma(e))$.

For example, each two nodes labelled F_1 in Figure 1 are semi-isomorphic.

Definition 3.2 (Breaking Points): Let v be a node in SPD f . Each outgoing edge e of v is labelled interval of nonnegative integers $I(e)$. It is easy to know that $I(e) \subseteq D(F(v))$. The interval $i = I(e)$ can be presented as $[i.f, i.b]$. If $i.f \neq 0$, the

point $(i.f - 1, i.f)$ is one of the *breaking points* in $F(v)$. If $i.b \neq \max(D(F(v)))$, the point $(i.b, i.b + 1)$ is also one of the *breaking points* in $F(v)$.

Definition 3.3 (Disjoint Rules): Let r_1 and r_2 are two different rules in a firewall policy. r_1 and r_2 are *disjoint* ($r_1 \not\bowtie r_2$) if there is no packet matching both r_1 and r_2 .

Each decision path $(v_0 e_0 \dots v_{k-1} e_{k-1} v_k)$ in an SPD f over the packet fields F_0, \dots, F_{n-1} can be represented as a rule of the form:

$$F_0 \in S_0 \wedge \dots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle action \rangle$$

where $\langle action \rangle$ is the label of the terminal node v_k in the decision path. It is clear that every rule is disjoint to another.

The security policy diagram in the current paper is a special type of the firewall decision diagram in [1]. A firewall decision diagram (FDD) can generate an *ordered-policy* which is consistent, complete and compact.

IV. UNDERSTANDABLE RULES GENERATION

In this section, we present an understandable firewall policy, which has the advantage of *no-order-policy* and *ordered-policy* in representing firewall rules. The security policy is generated from the administrator and the security policy diagram.

Definition 4.1 (Effective Rules): Let r is one of the firewall rules in rule set R . Rule r is an *effective rule* iff the filtering result of all the packets matching rule r is the same as the action of rule r .

Definition 4.2 (Disjoint Policy): Let R be a firewall policy. The firewall policy is a *disjoint policy* iff $\forall r_1, r_2 \in R, r_1 \not\bowtie r_2$.

Theorem 4.1: Every rule r in a *disjoint policy* is an *effective rule*.

Proof: Let rule r be a rule in a disjoint policy R . From the definition of *disjoint policy*, we get that:

$$\forall r' \in R (r' \neq r), r \not\bowtie r'$$

\Rightarrow There is no packet matching both r and r' .

\Rightarrow The filtering result of all the packets matching rule r is the same as the action of rule r .

$\Rightarrow r$ is an *effective rule*. ■

Theorem 4.2: A *disjoint policy* is a *no-order-policy*.

Proof: Let rule r be a rule in a disjoint policy R . From the definition of *disjoint policy*, we get that:

$$\forall r' \in R (r' \neq r), r \not\bowtie r'$$

\Rightarrow There is no packet matching both r and r' .

\Rightarrow Exchanging any rule in the rules with another will not influence the filtering result of the firewall.

\Rightarrow A *disjoint policy* is a *no-order-policy*. ■

As discussed in Section 1, common firewall rules are indigestible because they are in an *ordered-policy*. If there is a *no-order-policy* in the firewall, the rules will be more comprehensive. Our method is to transform a security policy diagram (SPD) into a *no-order-policy* policy. In section 3, we discuss that a security policy diagram can be represented as a *disjoint policy*. From **Theorem 4.2**, we present that a *disjoint policy* is a *no-order-policy*.

From a *no-order-policy*, it is easy to know the filtering result of any packet. However, transforming *disjoint policy* into realistic description is a hard work. Some rules in the

policy may have no realistic meanings because some intervals in condition field can not be represent as practical ones.

Our method is to find all the rules which is subset or equal to the rule which is configured by the administrator from the *disjoint policy*. Firewall filtering results are presented by these rules.

Algorithm 1 (Firewall Generation):

input : the SPD f and the rule r_a in policy R

output: the disjoint rule set R' in which every rule is subset or equal to r_a

steps:

- 1) Generate the *disjoint policy* $R(f)$ from SPD f .
- 2) For each rule $r \in R(f)$ do
if $r \subseteq r_a$ then insert r into R'

For example, the rules of Figure 1 are as follows:

[7,10] [5,9] d
[4,12] [5,9] a
[7,10] [5,9] d

default action is “d”(for discard).

V. FIREWALL POLICY EDITOR

Firewall policies are often written by different network administrators and occasionally updated(inserting, modifying or removing rules) to accommodate new security requirements and network topology changes. In this section, we present a policy editor tool that separates the security policy design from the firewall vendor specifics. This allows a security administrator to focus on designing an appropriate policy without worrying about firewall rule complexity, rule ordering, and other low-level configuration issues. At the beginning, with no rules in the policy, all the terminal nodes of SPD are labelled “u”(for undetermined).

A. Rule Insertion

The process of inserting a new rule r' in the global security policy is performed in two steps. Let R be the previous firewall policy and R' be the new one. In the first step, we first establish the breaking point set of R' on every field. A security policy diagram can be generated from the new breaking point set. All the terminal nodes in the new SPD is labelled ”u”(for undetermined).

Algorithm 1 (Firewall Generation):

input : an breaking point set B and a new rule r'

output: a new firewall policy

steps:

- 1) Insert the breaking points of r' into B .
- 2) Generate a security policy diagram from B and set all terminal nodes labelled “u”(for undetermined).
- 3) Generate a *disjoint-policy* R' from the SPD.

In the second step, we determine the action of new rules. At first, we set the action from the original policy.

Algorithm 2 (Action setting from the original policy):

input : the original policy R and the new policy R'

output: the new policy R' which has the same filtering result as R .

steps:

For each rule r' in R' do

1) find a rule $r \supseteq r'$ in R .

2) $r'.action = r.action$

However, the action of the new rule r' may be conflict with some rules in R' . The toolkit will submit these rules to the firewall administrator to get the proper action of them.

Definition 5.1: Rules r_1 and r_2 are *correlated* if every field in r_1 is a subset or a superset or equal to the corresponding field in r_2 . Formally, $r_1 \bowtie r_2$ iff

$$\forall i : r_1[i] \bowtie r_2[i]$$

where $\bowtie \in \{ \subset, \supset, = \}, i \in [0, n - 1]$

Definition 5.2: Tow rules are *conflictive* if they are *correlated* and have different filtering actions.

Algorithm 3 (Action Setting from the new rule):

input : the new policy R' and the new rule r'

output: the new policy R'

steps:

For each rule $r(r \subseteq r'$ and $r \in R')$ do

if $r.action \neq r'.action$ then

1) submit r to the firewall administrator for the proper action A .

2) $r.action = A$.

For example, if we insert a rule $r = [8, 10] [2, 11] a$ into the security policy corresponding to Figure 1, the algorithm will submit $[8, 10] [5, 9]$ to us for the correct action of the rule.

If we choose “a” for accept, the rules shown will be:

[7,10] [5,9] d
[7,10] [5,9] a
[4,12] [5,9] a
[8,10] [2,11] a

default action is “d”(for discard).

B. Rule Removal and Modification

The process of removing a rule r from the global security policy is also performed in two steps. Let R be the previous firewall policy and R' be the new one. In the first step, we first establish the breaking point set of R' on every field. Every breaking point of R' comes from R , but some breaking points of the rule to be removed may not appear in the breaking point set of R' .

Algorithm 1 (Firewall Generation):

input : an breaking point set B , a rule r_a to be removed and the original policy R

output: a new firewall policy

steps:

1) For each breaking point b of r

a) If b is the breaking point of $r'(r' \in R, r' \neq r)$ then reserve this point.

b) If there is no $r'(r' \in R, r' \neq r)$ which has the breaking point b , then remove b from B .

2) Generate a security policy diagram from B and set all terminal nodes with label “u”(for undetermined).

3) Generate a *disjoint-policy* R' from the SPD.

In the second step, we determine the action of the new rules.

Algorithm 2 (Action setting from the original policy):

input : the rule r_a to be removed, the original policy R and the new policy R'

output: the new firewall policy R'

steps:

For each rule $r'(r' \in R')$ do

1) If r' is not correlated with r_a then

$r'.action=r.action$

where $r \in D(R), r \supseteq r'$.

2) If r' is superset of r_a then

$r'.action=r.action$

where $r \in D(R), r \subset r'$.

3) If r' is equal to r_a then find all the rules $r(r \in R, r \supseteq r')$.

If every one of these rules is not conflict with another, then let the action of these rules A

$r'.action=A$

If there is conflict in these rules, then

$r'.action=r.action$ where $r \in D(R), r = r'$.

For example, if we remove the rule $r = [8, 10] [2, 11] a$ from the result of rule insertion, the algorithm will automatically generate the policy as follows:

[7,10] [5,9] d

[4,12] [5,9] a

[7,10] [5,9] d

default action is "d"(for discard).

which is the original rules.

Modifying a rule in a firewall policy is a crucial operation. However, this editing action can be easily managed as rule removal and insertion as described before.

VI. FIREWALL POLICY TEST

Currently, special device and environment are needed for firewall policy testing. A great deal of test cases need be designed to test a firewall. In this section, we present a method for testing firewall rules without testing device and testing cases. Our attention is focused on the design correctness. The test function in our toolkit is to assist the firewall design by displaying the effect of the security rules in certain condition.

Our approach to rule testing is based on the given $\langle condition \rangle$ over the fields F_0, \dots, F_{n-1} . The testing will gives firewall administrators filtering results of the packets which satisfy the $\langle condition \rangle$.

Let the $\langle condition \rangle$ be d_0, \dots, d_{n-1} over the fields F_0, \dots, F_{n-1} , each d_i is the condition interval on corresponding field F_i . If we find all the rules(each mapping a path in FDD) whose condition is correlated to the $\langle condition \rangle$, the rules found present the filtering result of the packets satisfying the condition.

The process of generating these rules is performed in three steps. In the first step, we first establish the breaking point set of R' on every field. Every breaking point of R is also that of R' , and the $\langle condition \rangle$ may import new breaking points. If a new breaking point is found, insert it into the breaking point set of R' . The algorithm is something like the first step of rule insertion.

In the second step, we determine the action of new rules from the original rules. The algorithm is something like the second step of rule removal.

In the third step, we display the rules in new security rule set whose condition is subset or equal to test condition.

For example, if we want to know the filtering result in the condition $[0, 15] [5, 9]$ from the security policy corresponding to Figure 1, the result will be presented as follows:

[0,3] [5,9] d

[4,7] [5,9] a

[8,10] [5,9] d

[11,12] [5,9] a

[13,15] [5,9] d

VII. CONCLUSION

Firewalls require proper management in order to realize proper filtering effect. The firewall administrator must understand the existing rules before editing them. After configuring the policy, a tool is required to approve that the rules in the firewall is correct.

In this paper, we present a firewall toolkit which allows a security administrator to focus on designing an appropriate policy without worrying about firewall rule complexity and rule ordering. When reading, editing or reviewing the firewall policy, what the administrator faces is only one rule. We consider the toolkit in current paper will improve the management of security policy.

REFERENCES

- [1] M. G. Gouda and X.-Y. A. Liu. Firewall design: consistency, completeness and compactness. In Proc. of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS04), March 2004.
- [2] M. G. Gouda and X.-Y. A. Liu. Diverse firewall design. In Dependable Systems and Networks, 2004 International Conference, June 2004
- [3] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In Proc. of IEEE Symp. on Security and Privacy, pages 17C31, 1999.
- [4] F. Baboescu and G. Varghese. Scalable Packet Classification. In IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 13, NO. 1, FEBRUARY 2005
- [5] V. Srinivasan. A Packet Classification and Filter Management System. In Proc. of IEEE Infocom, 2001
- [6] E.S. Al-Shaer and H.H. Hamed. Discovery of policy anomalies in distributed firewalls. In Proc. of IEEE Infocom, 2004
- [7] A. Hari, S. Sui and G. Parulkar. Detecting and Resolving Packet Filter Conflicts. In Proc. of IEEE Infocom, March 2000
- [8] S. Hiurichs. Policy-based Management: Bridging the Gap. In Proc. of 15th Annual Computer Security Applications Conference(ACSAC'99), December 1999
- [9] A. Mayer, A. Wool and E. Ziskind. Fang: a firewall analysis engine. In Proc. of Security and Privacy. May 2000
- [10] A. Wool. A quantitative study of firewall configuration errors. In Computer. IEEE. June 2004
- [11] Y. Du and D. Hoffman. PBit - a pattern-based testing framework for iptables. In Proc. of Communication Networks and Services Research. May 2004.
- [12] R. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Trans. on Computers, 35(8):677.691, 1986
- [13] K. Strehl and L. Thiele. Interval diagrams for efficient symbolic verification of process networks. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 19(8):939.956, 2000