symmetry

MDPI

Article

Firewall Anomaly Detection Based on Double Decision Tree

Zhiming Lin and Zhiqiang Yao *

College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350100, China
* Correspondence: yzq@fjnu.edu.cn; Tel.: +86-139-5030-5120

Abstract: To solve the problems regarding how to detect anomalous rules with an asymmetric structure, which leads to the firewall not being able to control the packets in and out according to the administrator's idea, and how to carry out an incremental detection efficiently when the new rules are added, anomaly detection algorithms based on an asymmetric double decision tree were considered. We considered the packet filter, the most common and used type of First Matching Rule, for the practical decision space of each rule and the whole policy. We adopted, based on the asymmetric double decision tree detection model, the policy equivalent decision tree and the policy decision tree of anomalies. Therefore, we can separate the policy's effective decision space and the anomalous decision space. Using the separated decision trees can realize the optimization of the original policy and the faster incremental detection when adding new rules and generating a detailed report. The simulation results demonstrate that the proposed algorithms are superior to the other decision tree algorithms in detection speed and can achieve incremental detection. The results demonstrate that our approach can save about 33% of the time for complete detection compared with the other approaches, and the time of incremental anomaly detection compared to complete detection is about 90% of the time saved in a complex policy.

Keywords: firewall; double decision tree; anomaly detection

Citation: Lin, Z.; Yao, Z. Firewall Anomaly Detection Based on Double Decision Tree. *Symmetry* **2022**, *14*, 2668. https://doi.org/10.3390/sym14122668

Academic Editors: Liangmin Wang, Keyang Cheng and Haiqin Wu

Received: 12 October 2022
Accepted: 8 December 2022
Published: 16 December 2022

1. Introduction

As one of the core elements of network and security of information systems, the firewall has been widely used to prevent suspicious traffic and unauthorized access to an enterprise [1]. The firewall is located at the boundary between the local and public networks. All incoming and outgoing data packets are checked according to the security rules to avoid the entry of malicious traffic packets and ensure regular business [2]. However, when the firewall is not configured correctly, it will cause serious security problems [3]. A misconfiguration of the Microsoft Azure cloud service leaked confidential information (including source code) of more than a dozen companies that submitted proposals for cooperation with Microsoft. Due to the firewall configuration error, the personal data of more than 100 million customers were exposed, including sensitive information.

It can be seen that properly designing a firewall policy is not easy [4] because the anomalies occur when the decision space of the rules is overlapped and the rules in the firewall policy are logically entangled [5] due to the anomalies and the result of the sequence sensitivity. Firewall rules have a certain symmetry in structure, but they are asymmetric when administrators write rules. If we manually configure a firewall, the security professional responsible for the configuration must know all possible valid and invalid inputs and outputs to be processed by the applications. In practice situations, it is difficult to change the anomalies manually. Because the number of anomalous rules in the firewall may be vast, the firewall policy may contain thousands of rules, which are usually logically intertwined. Modifying the rules will change the security expectations of the firewall [6,7].

To solve the problem of anomalies, previous work on firewall anomaly detection has been conducted through various techniques, such as those based on a single decision

diagram, based on satisfiability modulo theories, based on machine learning, and based on visualization models. The approach of the decision diagram to detect anomalies is not comprehensive or specific. Some of them only detect anomalies between pairwise rules or only detect anomalies at the rule level. Machine learning is also used to detect anomalies. However, the accuracy rate of the results by machine learning is not 100% [8–24]. The previous work has focused on detecting static firewall anomalies and rule-level anomalies. In contrast, the research on quickly detecting new incremental firewall rules and simplification policies is significantly insufficient, focusing on when new firewall rules are added, how to quickly detect anomalies incrementally, and how to simplify policy making so administrators can more efficiently manage the policy.

To solve the above problems, we present the formal definition of an anomaly and design some algorithms to detect the anomalies of a policy in a single firewall. We use the asymmetric decision tree to build a firewall policy without anomalous rules, and it can detect anomalies when adding new rules.

The key contributions of this paper are as follows:

(1) The asymmetric double decision tree is used to detect anomalies more efficiently and accurately. We construct a new firewall policy decision tree, which is an equivalent simplification of the original policy.

(2) We adopted the mode that separates the simplification of the policy and the detection of anomalies for the feature of the dynamic addition and deletion of the firewall policy. When detecting an incremental anomaly, we only need to compare the asymmetric simplified equivalent decision tree to find anomalies, which significantly improves the efficiency of incremental detection.

(3) We converted the mixed black-and-white list mode of the original policy into an equivalent single black (white) list mode, which will reduce the number of rules and generate a detailed and accurate report of anomalies.

The remainder of the paper is organized as follows. First, we review the related work introduced in Section 2. Second, we present the basic knowledge of a firewall and model the firewall policy in Section 3. We design the algorithms for building two decision trees and optimizing the policy by our decision tree in Section 4. We evaluate our approach in Section 5. We conclude this paper in Section 6.

2. Related Work

Many research efforts have been devoted to detecting anomalies in a firewall as much as possible. Multiple studies have addressed several approaches, including using a single decision diagram, formal methods, machine learning, and a visualization model.

(1) Anomaly detection based on decision tree.

Al-shaer et al. [8] introduced a framework to detect anomalies in single and distributed firewalls. They proposed a new tool called PolicyVis to detect anomalies. However, the tool only can detect anomalies between pairwise rules. Hu et al. [9] proposed a new anomaly management framework (FAME) that promoted the system to detect and solve anomalies in the firewall policy, considering and analyzing the relationship between all rules in the firewall configuration. To solve anomalies, they assigned an action constraint for each conflict segment between rules, which defines the required actions (allow or deny). To generate these action constraints, they used a vulnerability-based network risk assessment to generate "risk levels". They automated the process, but only analyzed the static rules of a given firewall once. A strategy to optimize and clean up the rule set has been summarized by Saâdaoui et al. [10]. They detected and fixed misconfigurations by removing superfluous rules from a simple firewall. Chao et al. [11] employed an enhanced adaptive rule anomaly relation tree and a two-dimensional traffic filtering matrix. In doing so, they considered an anomaly removal mechanism that attains a two-phase rule refinement procedure. These methods only analyzed existing static firewall rules and did not fully consider the characteristics of the firewall, which will dynamically add new rules.

(2) Anomaly detection based on firewall log.

Lu et al. [12] analyzed the order of rules in the firewall and designed a rule order adjustment algorithm that did not destroy the original semantics of the rule table. Additionally, they started from the matching probability of the rules; simple rules were separated from the default rules according to the firewall logs, the relationships between these rules and the original rules were analyzed, and the rules were merged into new rules to evaluate the impact of these rules on the performance of the firewall. A tabulated vector approach was proposed by Gutierrez et al. [13] to create meaningful state vectors from time-oriented blocks. Multivariate and graphical analysis was then used to analyze state vectors in a human–machine collaborative interface.

(3) Anomaly detection based on formal methods.

A solution was investigated to represent every two rules in IPv6 firewall policy in a formal verification format by Yin et al. [14]. They used SMT solver Z3 to verify their inclusion relations and determined their anomalies according to the inclusion relations and actions of rules. In [15], formal language FPL was introduced that enables a high-level human-understandable specification of the desired state of network security. The study demonstrated the instantiation of a compliance process using a verification framework that analyzed the configuration of complex networks and devices.

(4) Anomaly detection based on visualization model.

Kim et al. [16] applied an analysis tool to visualize segment-based firewall rules to facilitate verification of current control conditions. It can see the unnecessary areas in the ruleset. It can check inactive areas in partially matched and inclusively matched rules. A visualization tool designed by Lee et al. [17] showed the status and types of policies applied throughout firewalls to resolve the maintenance of firewall policies.

(5) Anomaly detection based on machine learning.

One study emphasized machine learning models for high-performance computing methods to detect anomalies in the firewall rule repository [18]. It was seen in all algorithms that they achieved the highest learning performance when they reached the training data value, composed of 1,500,000 data tuples; their performance level, however, began to fall after this point. Breier et al. [19] focused on a method to detect anomalies in log files, based on data mining techniques to create dynamic rules. They generate rules dynamically from specific patterns in sample files and can learn new types of attacks. Vartouni et al. [20] developed a method based on the deep neural network as a feature learning method and the isolation forest as a classifier. They applied different activation functions and learning for deep neural networks. Funk et al. [21] presented a firewall based on anomaly detection that aims to detect anomalous HTTP requests using the One-Class SVM classifier. They used expert knowledge about the HTTP request structure to build feature extraction methods that improved detection rates. In this study [22], the authors proposed methods based on deep neural network and parallel feature fusion that featured engineering as an integral part of them and played the most crucial role in their performance.

(6) The other methods.

Togay et al. [23] aimed to analyze firewall rules; they designed an anomaly detection framework for detecting intra-firewall policy anomalistic rules. Valenza et al. [24] proposed a model that was suitable to identify the relationships among the fields of the firewall rules and among the rules used to point out the anomalies.

In their methods, some methods used the state diagram to analyze the relationship between rules. The state diagram allowed for identifying the anomalies between pairwise rules in a single firewall policy. However, we consider all the anomalies between rules, not only the anomalies between pairwise rules. Some methods synthesized new rules by analyzing logs. Still, the semantics of the synthesized new rules cannot be entirely equal to the semantics that the original security administrator wanted to express, which will bring great difficulties for subsequent incremental modification operations. In addition, some approaches said that the rules are visualized through spatial coordinates. However, when the number of anomalous rules is vast, the visual expression will be inefficient, and the

anomalies between rules cannot be visually seen. For the machine learning method, it is a problem whether the detected anomalies are 100% correct. Most people do not trust the results of machine learning. The anomaly detection of firewall rules should be accurate. Whether the machine learning results are reliable or not, it is required for the security administrator to make a careful judgment on the results of machine learning detection, which significantly increases the time.

Previous work on rule analysis focused on analyzing static rules of a given firewall. It is common for a rule to have anomalies with multiple rules. Anomalous rules also need to be accurately represented. It means not only a single indication of which rules have anomalies, but also an indication of which traffic packets will cause the rules to become anomalies. This indication will help to correct rules without generating new anomalies. The approach proposed in this paper can detect the anomalies of any rule. Considering the dynamic modification of the firewall policy, it can achieve rapid and efficient incremental detection of anomalies. All detected irregularities can accurately show which traffic packets will cause the anomalies and point out the action of the firewall when the traffic packets arrive.

3. Formal Definition

3.1. Preliminaries

Firewalls control packet filtering across secure network boundaries according to a specific security policy. A firewall security policy is an ordered list of filtering rules that define the action to be performed on packets that match specific conditions. A firewall rule generally consists of filter fields such as protocol type, source IP address, destination IP address, source port, destination port, and an action (generally accept or deny). The filter fields have a certain symmetry, but the actual decision space of each rule is asymmetric when the firewall is running. The filter field of the rule represents the possible values of the corresponding fields matching the rule in the actual network traffic. Each filter field can be a single value or an interval value. The filtering action is either to accept packets that allow them to enter and leave the secure network, or to deny packets that are discarded. If the fields in a packet match all the filter fields of a rule, the package will be allowed or discarded by the rule. Otherwise, the remaining rules will be checked until a rule is matched or a default rule is performed.

Al-Shaer introduced the concept of anomaly, defined as when "there are two or more filtering rules that may match the same packet, or there is a rule that can never match any packet". They considered the pairwise rules' anomalies and identified four kinds of anomalies, which depend on the priority of the rule and the operation enforced. According to Al-Shaer, the definitions are as follows:

(1) Hidden: Refers to the situation where all traffic packets that a rule wants to reject (accept) are accepted (rejected) by the previous rule.

(2) Generalization: Refers to the case where the previous rule has excluded the subset of traffic packets matching this rule.

(3) Correlation: Refers to the case where the rule intersects with other rules, and different decisions are defined.

(4) Redundancy: Refers to the case for one rule's condition if another rule's situation is the same and both have the same action.

Existing anomaly classification and detection methods regard anomaly as the inconsistency between two rules. This paper states that the firewall policy should always be considered as a whole when identifying the policy anomaly. There should be no overlap space between rules and they should be independent of each other, which is important to solve anomalies effectively.

Therefore, this paper considers that these four definitions of anomalies can simplify and make a more general definition of an anomaly.

Definition 1. *Anomaly: When the decision space of firewall rules overlaps, a traffic packet can match multiple rules.*

3.2. Formalization of Rule

Modeling firewall asymmetric rules is necessary for analyzing firewall policy, designing anomaly detection, and policy editing. This section formally describes our model and explains the firewall policy using an asymmetric decision tree.

Action: The action of the firewall's rules represents whether the firewall will accept or discard the packet. We define A to express the action set. P is the action of permit and D is the action of discard.

$$A = \{P, D\} \tag{1}$$

Conditional filter field: Let f_i denote the conditional filter field, which represents the set of all values that the rule may adopt under this field. For example, if f_i is the filter field value of the source IP, which is 192.168.1.*, the rule matches the IP address range from 192.168.1.0 to 192.168.1.255.

$$f_i = (srcIP/srcPort/sdtIP/dstPort/action) \tag{2}$$

Filter field: Let F_i be the filter field that indicates the set of all values of the field. For example, when F_i is srcIP or dstIP, the value of the filter field F_i is the IP from 0.0.0.0 to 255.255.255.255.

$$F_i = (value), value = 2^{IP/Port/action} \tag{3}$$

Decision space: Let s be the decision space of the rule, which is composed of the number of d conditional filter fields (such as the source IP target and port number). The decision space represents a set that contains all possible values that each filter field of the rule may adopt.

$$s = f_1 \times f_2 \times \cdots \times f_d \tag{4}$$

Rule: We model rules in the form of "decision space $\rightarrow$ action". Rules consist of conditions and actions.

$$r = (s, a), a \in A \tag{5}$$

3.3. Formalization of Decision Tree

This paper uses an asymmetric decision tree to represent firewall policy. The decision tree model provides a simple representation of the firewall rules and can quickly discover the relationships and anomalies of the rules. Each node in the decision tree represents a conditional filter field, and each branch on the node represents a possible value of the related field. Each path of the tree from the root to the end of the leaf represents the valid part of the rule in the policy. The decision tree has the following properties.

(1) There will be a node without an incoming edge, called the root. A node without an edge is called a terminal node. Let 2^T be the set of terminal nodes and 2^N be the nonterminal nodes.

(2) Each node v has a set of filter fields, which is denoted as $F(v)$ for any $F(v)$ having the following properties:

$$\forall i < k, j < k, I(e_i) \cap I(e_j) = \varnothing \tag{6}$$

$$F(v) = \begin{cases} \{I(e_1), I(e_2), \cdots, I(e_k)\}, & v \in 2^N \\ r.name, & v \in 2^T \end{cases} \tag{7}$$

(3) Each edge $e(u \rightarrow v)$ is a set of non-empty sets, which is a subset field of the parent node u, and $I(e)$ is used to represent the range of filtered field values for that edge ($I(e) \subseteq F(u)$).

(4) The directed path from the root node to the terminal node is called the decision path. We stipulate that any two nodes on the decision path will not have the same field.

$$f = f_1 \times f_2 \times \cdots \times f_d \times f_{acton} \rightarrow r.name \tag{8}$$

4. Asymmetric Double Decision Tree-Based Detection

Figure 1 illustrates the structure of our approach. This architecture is ascertained in the local server using construction and optimization layers. The execution flow is explained below, and the details are described in the following subsections.

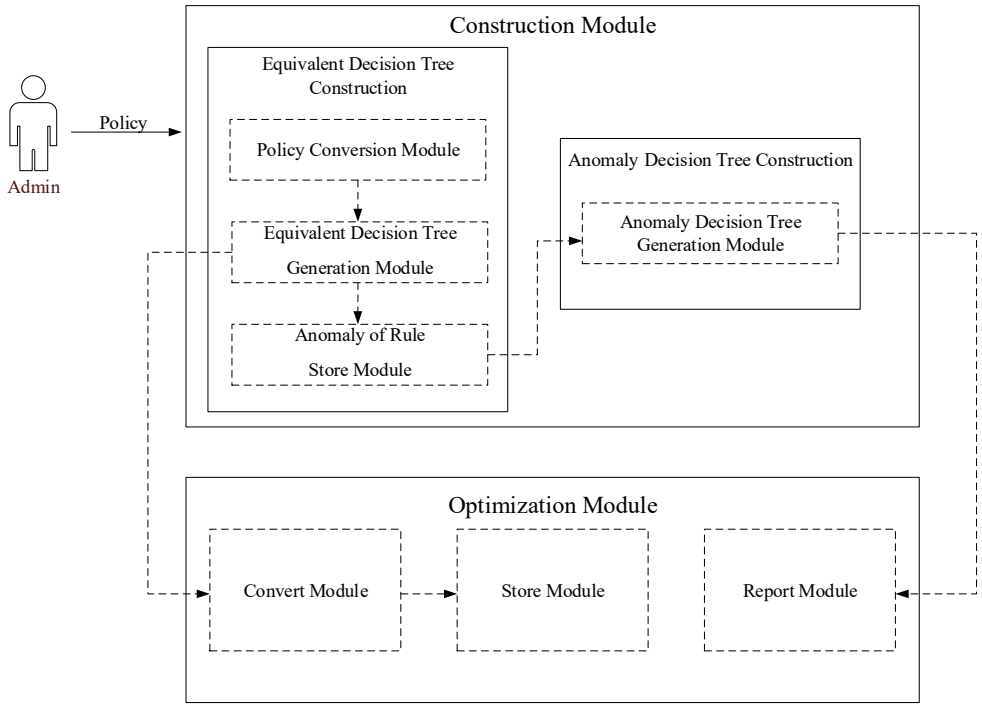

Figure 1. The structure of our approach.

Without loss of generality, in this work, we consider the packet filter, the most common and used type of First Matching Rule (FMR), which selects the action from the first applicable rule in an ordered list of a firewall. We propose an anomaly detection method based on asymmetric double decision trees. The main steps are as follows: (1) Firstly, the original policy is transformed into an equivalent firewall decision tree by an equivalent transformation algorithm, and the overlapping part of the rules is stored while constructing the equivalent decision tree. (2) Generate a decision tree of anomalies by the overlapped part stored in Step 1. (3) The equivalent decision tree is transformed into a single black (white) list mode according to the comparison of the number of nodes' action which is allowed or discarded in the equivalent decision tree. (4) The corresponding anomaly detection report is generated according to the rule's decision tree of anomalies. According to the report, the administrator can determine which traffic packets match multiple rules. The report will indicate which rule will determine whether the traffic packets are allowed or discarded. When the administrator adds a rule, he can know whether the new rule is abnormal compared to the original policy set by matching it with the equivalent decision tree.

4.1. Equivalent Decision Tree Construction

In this phase, we convert the original firewall policy into an equivalent decision tree. Each path of the equivalent tree is independent. The generated decision tree is equal to the

decision of the original policy. By default, any node v has k outgoing edges $e_1, e_2, \cdots, e_k$. Suppose that the added rule is denoted as:

$$f_r = f_1 \times f_2 \times \cdots \times f_d \times f_{acton} \rightarrow r.name \tag{9}$$

The steps are as follows. When the tree is empty, insert the first rule. When the tree is not empty, if the new set of edges is not empty after removing edges $(f_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k)) \neq \varnothing)$, insert a new edge directly. The value of the new edge is the new edge, removing the current edge set $(f_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k)))$ and inserting the subsequent path ($f_2 \times f_3 \times \cdots \times f_d \times f_{acton} \rightarrow r.name$) of the new edge directly into the new edge. In that way, we ensure that the effective part of a rule can be retained. Then, judge the relationship between each edge's f_i which will be added and each outgoing side e_i of the node F_i in the tree:

(1) If the current edge is a subset of the edges $(I(e_i) \cap f_i = I(e_i))$ which will be added, then continue to match the next node.

(2) If the current edge is not a subset of the edges that will be added and the intersection is not empty $((f_i \cap I(e_i) \neq \varnothing) \wedge (f_i \cap I(e_i) \neq I(e_i)))$, first compare the intersection part $(f_i \cap I(e_i))$ and continue judging the relationship between f_{i+1} and F_{i+1}, and then compare the disjoint part $(f_i - f_i \cap I(e_i))$ with each other edge $(e_j, (j \neq i))$. Algorithm 1 shows the pseudocode of the Equivalent Decision Tree Construction.

For example, a simple rule set is given in Table 1.

Table 1. Examples of rules.

Protocol	Source Address	Port	Destination Address	Port	Action
1: tcp	140.192.37.20	any	*	80	deny
2: tcp	140.192.37.*	any	*	80	accept
3: tcp	*	any	161.120.33.40	80	accept

We use "A/B" to express removing the part of set B from set A in the following figures. When we apply Algorithm 1 to create the equivalent decision tree, the tree is first empty. We insert the first rule in Figure 2a. Then, we insert the second rule into the tree in Figure 2b. We recursively check the node and its edges to find whether the intersection of the new rule and the node's edge was null. If not, we obtain a disjoint of the new edge such as (140.192.37.*/140.192.37.20) and insert the effective part into the tree. We recursively check the overlap part to find the anomalies. Because the left nodes' values of rule 2 are the same with each node's edges, the tree did not insert a new edge. We retain the effective part of rule 2 because the overlap of the packet will match rule 1 and never match rule 2. We insert rule 3 in the same way. Finally, we obtain the equivalent decision tree in Figure 3. The tree retains the effectiveness of each rule, and the overlaps that are the anomalies of a paired rule are saved to create the decision tree of anomalies.

4.2. Anomaly Decision Tree Construction

In this phase, we extract all the parts that overlapped in the first step to generate a tree of anomalies to detect the anomalies.

The steps are as follows: Insert the rule directly when the tree is empty. When the tree is not empty, if the new edge is not empty after removing the current edge set $(f_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k)) \neq \varnothing)$, insert a new edge directly. The value of the new edge is derived by removing the current edge set $(f_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k)))$ and inserting the subsequent path ($f_2 \times f_3 \times \cdots \times f_d \times f_{acton} \rightarrow r.name$) of the new edge directly into the new edge. Then, judge the relationship between each edge's f_i which will be added and each outgoing side e_i of the node F_i in the decision tree:

(1) If the current edge is a subset of the edges which will be added $(I(e_i) \cap f_i = I(e_i))$, then continue to match the next node.

(2) If the edge to be added does not intersect the current edge ($f_i \cap I(e_i) = \varnothing$),then continue to match the next edge e_j ($j \neq i$).

(3) If the current edge is not a subset of the edge that will be added and the intersection is not empty $(f_i \cap I(e_i) \neq \varnothing) \wedge (f_i \cap I(e_i) \neq I(e_i))$, first, insert a new edge e_{k+1} in the node. The value of the new edge is the intersection of the edge which will be added and the current edge ($I(e_{k+1}) = f_i \cap I(e_i)$). The subgraph of the new edge is the subtree of the current edge e_i. Replace the value of the current edge with the value of the edge which will be added ($I(e_i) \leftarrow (I(e_i) - f_i)$), then continue to match the subtree of the new edge with the next node f_{i+1} and finally generate a tree of anomalies. Algorithm 2 shows the pseudocode of the Anomaly Decision Tree Construction.

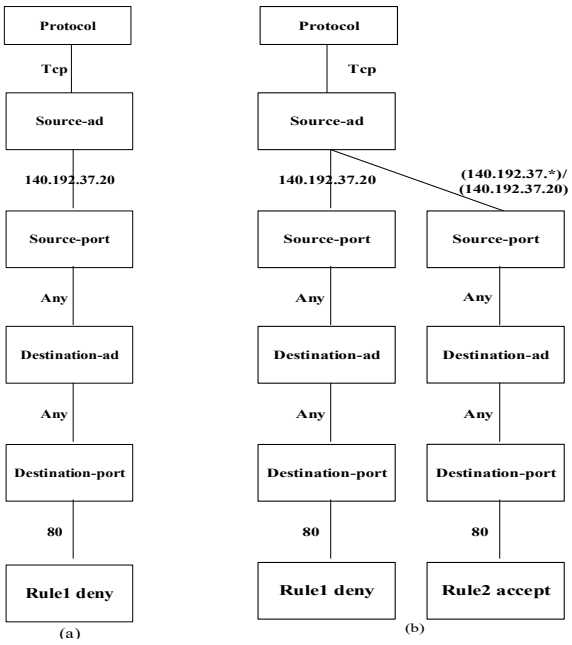

Figure 2. The process of creating an equivalent decision tree. (**a**) The first rule inserts into a tree; (**b**) the second rule takes out the effective part and inserts it into the tree.

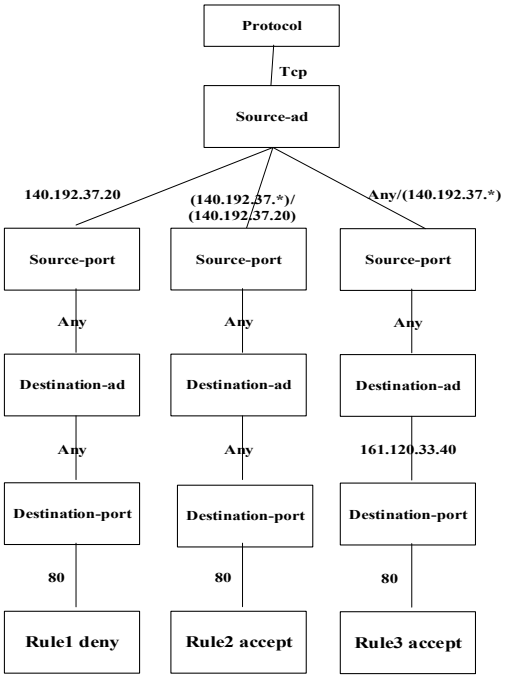

Figure 3. The equivalent decision tree of the example policy.

Algorithm 1: Equivalent Decision Tree Construction (*RuleSet*)

Input: firewall rule set <r_1, r_2, $\cdots$, r_n>
Output: equivalent decision tree f'
Step:
1: $f' = f_1 \times \cdots \times f_d \times f_{action} \rightarrow r_1$ $v \leftarrow f'.root$;
2: new *map* (v, e);
3: **for** i = 2 to n **do**
4: $space = \emptyset$;
5: Append(v, r_i, *space*, *map*);
6: spaces.add(*space*); /*anomalies of paired rules*/
7: anomalyDecisionTreeConstruction (*spaces*);
8: **end for**
9: **return** f'
10: **End**
11: Append (v, $f_m \times \cdots \times f_d \times f_{action} \rightarrow r$, map($v$, e);/*insert into tree*/
12: **if** (f_m-($I(e_1) \bigcup I(e_2) \bigcup \cdots I(e_k)$) $\neq \emptyset$) **then** /*new edge is overlap with node's edge */
13: $I(e) = f_m$-($I(e_1) \bigcup I(e_2) \bigcup \cdots I(e_k)$); /* take out the effective part of new one */
14: insert (v, e, $f_{m+1} \times \cdots \times f_d \times f_{action} \rightarrow r$);
15: *map.add*(v, e);
16: **end if**
17: **if** $m \leq d$ **then**
18: **for** j=1 to k **do**
19: **if** ($I(e_j) \cap f_m \neq \emptyset \wedge m \neq d$) **then** /* none-leaf node*/
20: $space = space \times (I(e_j) \cap f_m)$; /*anomalous path*/
21: Append ($e_j.point$, $f_{m+1} \times \cdots \times f_d \times f_{action} \rightarrow r$, *space*);
22: **else if** ($I(e_j) \cap f_m \neq \emptyset \wedge m = d$)**then**
23: $space = space \times (I(e_j) \cap f_m)$;
24: **end for**
25: **end if**

For the previous example, when rule 2 is inserted into the equivalent decision tree, we obtain the anomaly of rule 1 and rule 2, and then the tree is empty. We insert the anomaly path into the tree in Figure 4a. When rule 3 is inserted into the equivalent decision tree, we recursively check the nodes to find the disjointed part. When rule 3 meets the source-ad, we obtain the effective part (Any/(140.192.37.*)). We recursively check the subsequent nodes by intersecting elements (140.192.37.*).

When matching the first decision path, we obtain the anomalous path (protocol.tcp $\times$ source-ad.140.192.37.20 $\times$ source-port.any $\times$ destination-ad.161.120.33.40 $\times$ destination-port.80 $\times$ rule.1/3), and comparing the second path, we obtain the other anomalous path (protocol.tcp $\times$ sourceIP.[(140.192.37.*)/(140.192.37.20)] $\times$ sourcePort.any $\times$ destinationIP.161.120.33.40 $\times$ destinationPort.80 $\times$ rule.2/3). In the next step, we add the anomalous paths into the decision tree of anomalies. Finally, we obtain the complete anomaly decision tree in Figure 4b. Each decision path represents an anomaly.

4.3. Equivalent Decision Tree Optimization

In this phase, we optimize and compress the equivalent decision tree generated in the first step. The steps are as follows: (1) Judge whether the new node or new edge stored in Step 1 has other edges, which means that this edge can be deleted. (2) After removing all the edges that cannot be deleted in the decision tree, judge the number of allowed paths and rejected paths, and then delete the rules whose action is the same with numerous paths. Finally, generate the default allowed or rejected rules at the end of the policy. Algorithm 3 shows the pseudocode of the Equivalent Decision Tree Optimization. When we generate the equivalent tree, every decision path representing the disjoint decision space with the other rule is mutually independent. The intersecting parts of a rule are ineffective because the intersecting decision space will be caught by a high-priority rule ahead of it. Algorithm 1 is ordered to retain the effective decision space of a rule by comparing and retaining the

current rule's decision space that is disjointed with the forward decision space. We divide the policy into two independent spaces: accept space and deny space. Therefore, we can delete any space and add a default rule to catch all packets. The action of the default rule is equivalent to the deleted decision space to keep the tree equivalent to the original policy.

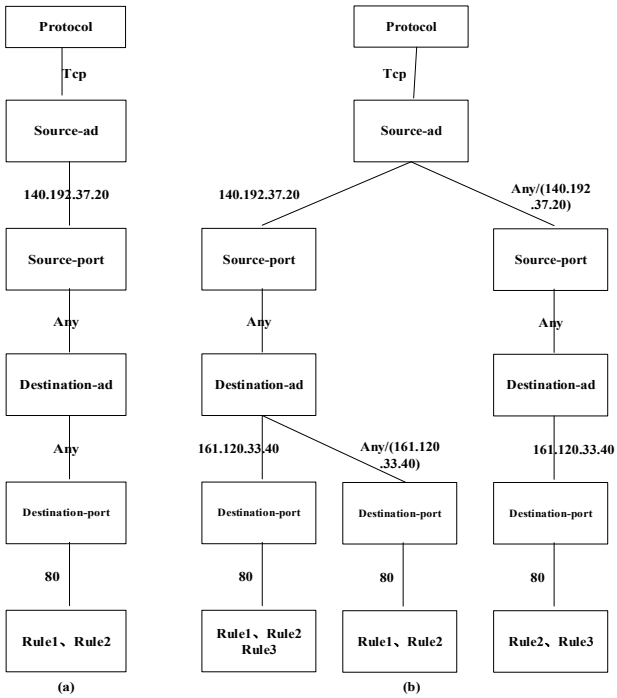

Figure 4. The decision tree of the example policy. (**a**) The first rule inserts into the tree; (**b**) the final decision tree of anomalies.

Algorithm 2: Anomaly Decision Tree Construction (*Space*)

Input: anomaly space $<f_1, f_2, \cdots, fn>$
Output: anomaly decision tree f'
Steps:
1: $f' = f_1 \times \cdots \times f_d \times f_{action} \rightarrow r_1$; $v \leftarrow f'.root$;
2: **for** i = 2 to n **do**
3: append(v, r_i);
4: **end for**
5: **return** f'
6: **End**
7: Append (v, $f_m \times \cdots \times f_d \times f_{action} \rightarrow r$);
8: **if** (f_m-($I(e_1) \bigcup I(e_2) \bigcup \cdots I(e_k)$) $\neq \emptyset$) then/*new edge is overlap with node's edge */
9: $I(e) = f_m$-($I(e_1) \bigcup I(e_2) \bigcup \cdots I(e_k)$);
10: insert (v, e, $f_{m+1} \times \cdots \times f_d \times f_{action} \rightarrow r$);
11: **end if**
12: **if** $m \leq d$ **then**
13: **for** j = 1 to k **do**
14: **if** ($I(e_j) \cap f_m \neq \emptyset \wedge m \neq d$) **then**
15: $I(new\_e) = I(e_j) \cap f_m$; /*overlap became a new edge*/
16: insert ($e_j.father$, new_e, $e_j.f_{m+1} \times \cdots \times e_j.f_d \times e_j.f_{action} \rightarrow r$);
17: Append ($new\_e.point$, $f_{m+1} \times \cdots \times f_d \times f_{action} \rightarrow r$);
18: $I(e_j) = I(e_j) - f_m$;
19: **else if** ($I(e_j) \cap f_m \neq \emptyset \wedge m = d$) **then**/*leaf node*/
20: insert ($e_j.father$, new_e, $e_j.f_{m+1} \times \cdots \times e_j.f_d \times e_j.f_{action} \rightarrow r$);
21: $I(e_j) = I(e_j) - f_m$;
22: **end for**
23: **end if**

For the previous example, the equivalent decision tree is not simplified, and the policy is mixed by blacklist and whitelist. We change the terrible policy into a single black/white list in the next. First, we count the number of the "action" to decide which model to choose. If the number of "deny" actions is less than "accept", we decide to change the policy into a blacklist that will delete the branch of the accept path and add a rule that allows any packet to pass the firewall in the last and vice versa. Finally, we obtain the simplified equivalent decision tree in Figure 5.

Algorithm 3: Tree Optimization (*Root*, *Branch*)

Input: the root of equivalent decision tree *v* and branch of equivalent decision tree *map(v, e)*
Output: Optimized Decision Tree *f'*
Steps:

```
cut (v, map(v, e));
End
Cut(v, map(v, e));
acceptMap = map.getAccept(); /*get "accept" paths*/
denyMap = map.getDeny();
for (v': map.keySet) do
   if (hasBranch(v')) then;/* judge whether there is no sub-branch*/
     map.remove(v');
   end if
 end for
 int accept = getAcceptPathCount(acceptMap);/*count "accept" paths*/
 int deny = getDenyPathCount(denyMap);
 if (accept > deny) then
   v.removeEdge(acceptMap);/* simplification */
   f'.add(f_deny); /*keep equivalent*/
 else then
   v.removeEdge(denyMap);
 f'.add(f_accept);
```

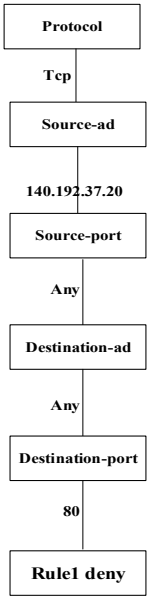

Figure 5. The final simplified equivalent decision tree of the example.

4.4. Incremental Detection

When we obtain the optimized equivalent decision tree, if some new rules are added to the policy, we can quickly derive the anomaly between the new rules and the original rules by using Algorithm 1 for the detection of anomalies. The simplified equivalent decision tree

is the premise of incremental detection of anomalies. We conduct much work to simplify and maintain the equivalence to detect anomalies incrementally. Because the simplified equivalent decision tree represents the original policy, we detect the anomalies based on the tree instead of creating the tree from all rules. In that way, we can achieve efficient incremental detection of anomalies.

For the previous example, if we add a new rule, we just compare the simplified tree in Figure 5 instead of creating a new one again.

5. Evaluation and Experimental Results

This paper implements the firewall policy anomalies detection tool in Java. Our policy anomaly analysis mechanism consists of five core components: the policy conversion module, the equivalent decision tree generation module, the decision tree of anomaly generation module, the equivalent tree optimization module, and the equivalent tree storage module. The policy mapping module takes the policy text form as the input and converts the rules of the policy form text form into decision paths, then converts the decision paths into an equivalent tree through the equivalent tree generation module. During the conversion, an anomaly of rules is found, and the anomalous decision space is stored. After that, the anomaly decision tree of all rules is generated through the anomaly decision tree generation module. The equivalent decision tree is converted into a single black (white) list mode in the optimization module. Finally, the equivalent decision tree is stored, which will be used for subsequent incremental detection of anomalies.

We use the synthetic firewall policy to evaluate our detection tool. The experiment in this paper is to run the tool on an Intel dual-core CPU with a reference speed of 2.4 GHz and a memory of 8 GB. We randomly generate 50, 100, 200, 300, 400, 500, and 800 firewall rules for evaluation. Rules in different systems may have various forms, but their essential meanings are the same. Random rules are more complex than realistic policies. If the rules have potential connections, this will improve detection time because the most time-consuming part is comparing the branches of each node. The more relevant the rules are, the more straightforward the tree is in our approach. Because we retain the decision space that is not covered by the previous decision space of a rule, the possibility of overlap by random rules is less than realistic rules, which means that the detection times of random rules are longer than realistic ones.

We evaluate the storage space of the policy decision tree in Figure 6. To quickly detect the incremental anomalies and generate a new policy, the equivalent decision tree will be stored after detection for subsequent incremental rapid detection. It can be seen from the figure that with the increase in rules, the storage space of the equivalent policy decision tree also gradually increases slowly. When the rules gradually increase, the probability of duplication with the current firewall rules gradually increases, and the scheme in this article will directly discard the duplicate parts of the following rules with the existing rules. At the same time, we change the black-and-white list mixed mode to a single black/white list mode, which significantly reduces the number of rules. The space increases exponentially at first, but when the rule scale reaches a certain amount, the space increases slowly. If we fit the data in the latter part, the function curve is close to ln(n). Theoretically, the storage space of our approach's equivalent policy decision tree increases slowly because we only reserve the practical part of every rule. When the rules become very large, the latter policy rules are ineffectual because the front rules' decision space covers the latter. The storage of the equivalent policy decision tree serves to realize the incremental detection. The figure intends to indicate that the expenses of incremental detection are relatively small.

We also evaluate the generation efficiency of the decision tree of anomalies and compare it with the other methods [8,17] in Figure 7. We show the process time of different approaches in Table 2. PolicyVis [8] was based on a single decision tree to detect the anomaly. The branches under a node were not mutually independent in the single tree, so they cannot simplify the tree to incrementally detect anomalies and only detect the anomalies of paired rule. If the rules become larger and larger, the branches of each

node exponentially increase, bringing about the time of detection exponentially growing. HSViz [17] adopted a visualizing policy to show the anomalies intuitively, but the view was too complex to show the entire decision space of rules. Significantly, when the number of rules became tremendous, the view became extremely complex, which was unintuitive to show anomalies. Moreover, it cannot dynamically detect anomalies when new rules are added to the policy. In our approach, only the effective part of the rule is reserved when generating the equivalent decision tree. As the rules grow gradually, the elements that overlap with the current firewall rules are not inserted repeatedly, which greatly reduces overhead. The rule detected only needs to match a decision path in the equivalent decision tree to know if the rule has an exception. If an anomaly occurs, we only need to match the path to the exception decision tree to find the exception. In this way, we can point out the anomalies of all rules. Compared to other methods that require the rule to be detected to be compared with each generated segment, the scheme in this article dramatically reduces the comparison lookup time. The number of anomalous rules positively correlates with the number of original rules, so we can see that the generation time of the decision tree of anomalies increases exponentially with the increase in rules.

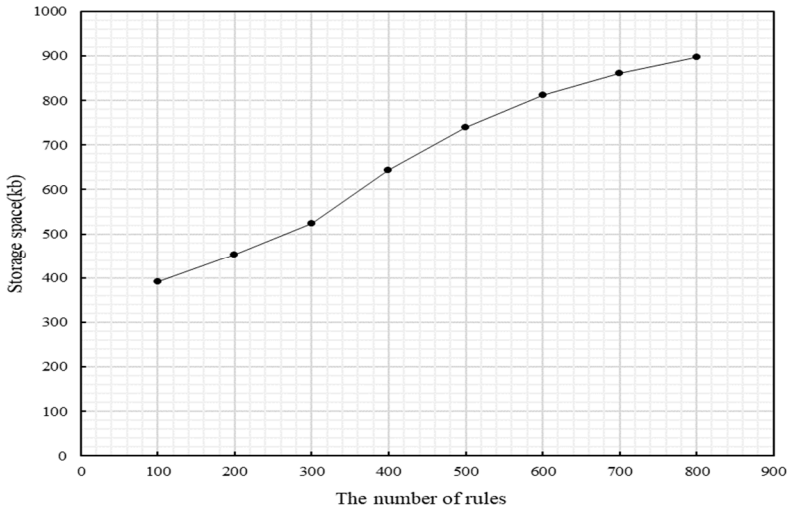

Figure 6. Relationship between the number of rules and the storage space of the equivalent decision tree.

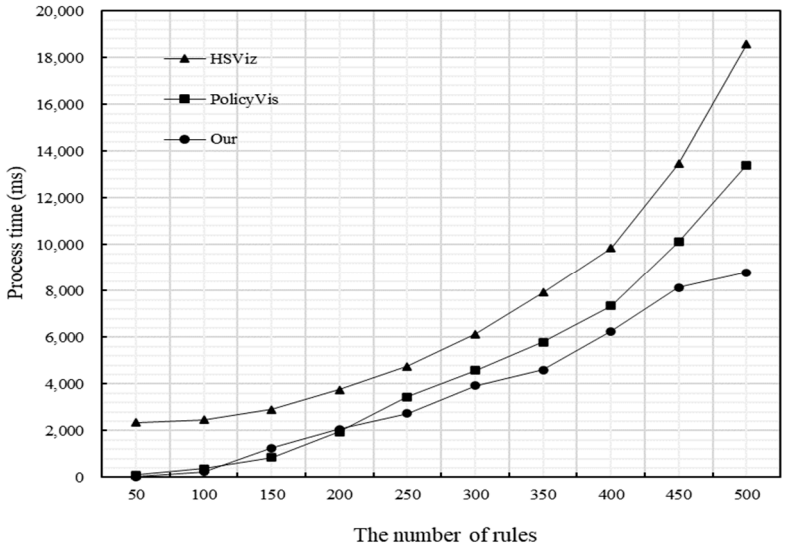

Figure 7. Comparison of the anomaly detection time in different methods.

Table 2. Process time of different approaches.

Number of Rules	Approach Name	Process Time (s)
100	HSViz	2470
	PolicyVis	364
	ours	238
200	HSViz	3760
	PolicyVis	1954
	ours	2062
300	HSViz	6147
	PolicyVis	4587
	ours	3935
400	HSViz	9842
	PolicyVis	7340
	ours	6254
500	HSViz	18,578
	PolicyVis	13,413
	ours	8797

In Figure 8, we evaluate the equivalent tree optimization module. The original rules are reduced from 50, 100, 200, and 300 to 10, 27, 31, and 55, respectively. The number of optimized rules drops by about 80% compared to the original rules. The paper's optimization module converts the original policy's black-and-white list mode to the single black (white) list mode according to the number of leaf nodes' actions in the equivalent decision tree. It can be seen from the figure that the number of original rules is increasing, but the number of rules increases slowly. Moreover, compared to the original rules, the number of converted rules is significantly reduced, greatly increasing the administrator's ability to manage policy.

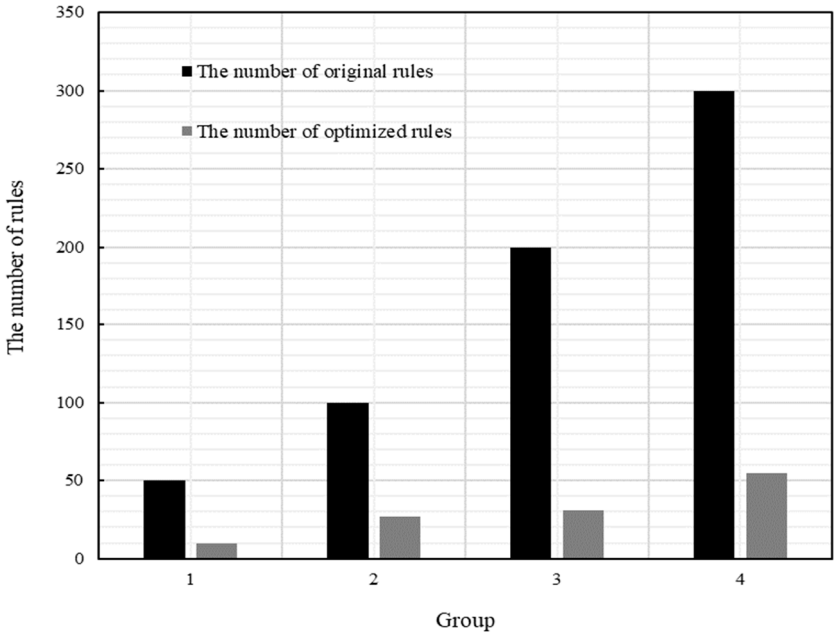

Figure 8. Comparison of rules before and after optimization.

We also evaluate the decision tree generation time and the anomaly detection time after adding new rules that are inserted into a complex policy of 500 rules, as shown in Figure 9. When we perform incremental anomaly detection, we first judge the action of every new rule during incremental detection. The detection of anomalies and the insertion of rules will be performed only when the effect of the new rules is the same as the rules in the equivalent decision tree. As we can see from the figure, the number of new rules

increases, but the time to generate a new policy decision tree to discover anomalies is relatively short. The time to detect incremental anomalies is significantly faster than that to detect complete anomalies.

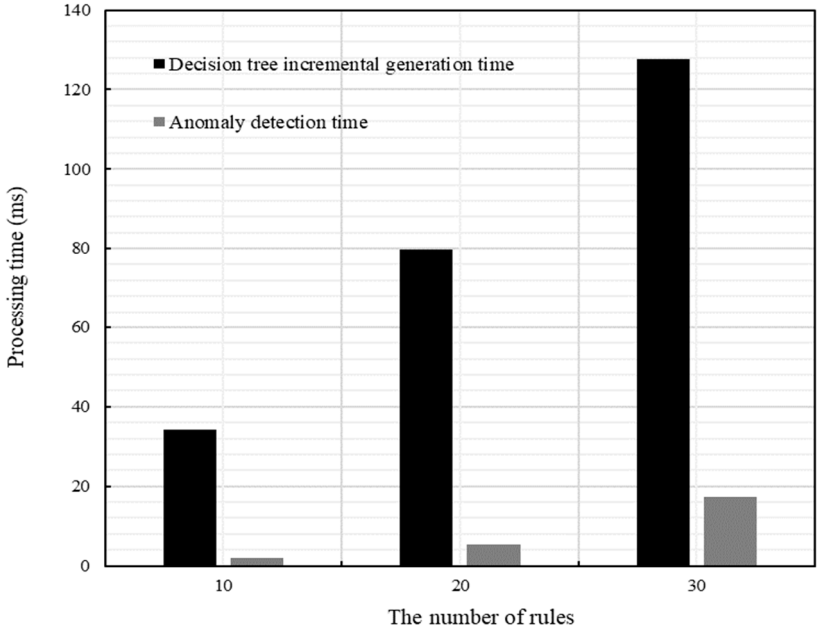

Figure 9. Detection time of adding the different number of rules.

We also verify the equivalence of the decision tree, randomly generate 10, 20, 50, and 100 packets, and then record whether the packets are dropped or passed after being filtered by the original policy. After that, compared with the filtering results of the equivalent decision tree, if the comparison results are all consistent, the decision tree is equal to the original policy. To further illustrate that the simplified equivalent tree is equivalent to the original policy, we construct the rule set from the original rules. Specifically, we compile every rule of the original policy, and according to the original rule, we create rules that can cover all the decision space of the original rule. In this way, we ensure that the test rule set could stand for the original decision space. Our experimental results show that the packet passing results are identical to the original policy. We also test the accuracy of anomaly detection, matching the exception rules in the generated exception report with the rules in the original policy. The experimental results showed that all reported anomalies policies were accurate. Theoretically, when we create the equivalent tree in Algorithm 1, we only retain the effective part, which is not covered by the front decision space of each rule. Therefore, the tree created by Algorithm 1 is equivalent to the original policy, as shown in the experiment.

When using the implementation of our approach, it takes about 8.8 s to analyze the complex policy of 500 rules in a single firewall and takes 3 s to generate a new equivalent policy, which is equal to the original policy and only takes about 150 milliseconds when we added 30 rules to detect the anomalies incrementally. Experiments show that our approach's process time is faster when the number of rules is enormous. The time of incremental detection of anomalies compared to complete detection is about 90% of the time saved in a complex policy of 500 rules. Our algorithms only retain the effective part of the rule, but the other approaches kept the fundamental rule to detect the anomalies. In other techniques, they must deal with the decision space of every rule, whether the forwarding rules cover the rule. Only this way can they detect anomalies. However, we separate anomalies from policy creation and simplification, so we do not need to handle each rule. We improve the approach used by a single decision tree that simply separated the rules that did not consider the effective part of each rule. Our approach separates

policy simplification and anomaly detection, significantly improving detection speed and preparing for incremental detection.

6. Conclusions and Future Work

Like any other technology, firewalls require appropriate management to provide appropriate security services. However, network vulnerabilities caused by the complexity of firewall rules and anomalies between rules will make the network unsafe. The firewall anomalies detection tool introduced in this paper provides a technique to optimize and detect anomalies in a firewall policy. Attempting to explain the asymmetry of firewall rules, we constructed an asymmetric double decision tree model to detect the anomalies of a firewall. The administrator can use the firewall policy anomalies detection tool to detect and simplify the firewall policy and perform incremental anomaly detection for subsequent policy updates. In this paper, we define the policy anomalies in a single firewall. Then, we propose algorithms to detect rule anomalies in a single firewall. We convert the original input policy into a single (black/white list) mode that is more concise and has the same effect as the original policy. We implement the tool in Java.

Regarding availability, the firewall policy anomalies detection tool can find all anomalies written by expert network administrators. In terms of performance, the time of policy anomaly analysis is related to the number of rules in the firewall policy. However, our experiments show that the average processing time for the detection of anomalies in the firewall is reasonable for a realistic environment.

Future work includes practically implementing the intra-firewall management tool in a real-time environment. The ability to handle inconsistencies in distributed firewalls is also left as a part of future work.

Author Contributions: Writing—original draft preparation, Z.L.; supervision, Z.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (grant 61872090), Fujian Provincial Science and Technology Guidance Project (grant 2019H0010), and the Open Fund of Fujian Provincial University Engineering Research Center (grant FJ-ICH201901).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Daly, J.; Bruschi, V.; Linguaglossa, L.; Pontarelli, S.; Rossi, D.; Tollet, J.; Torng, E.; Yourtchenko, A. Tuplemerge: Fast software packet processing for online packet classification. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1417–1431. [CrossRef]
2. Liu, A.X.; Khakpour, A.R.; Hulst, J.W.; Ge, Z.; Pei, D.; Wang, J. Firewall fingerprinting and denial of firewalling attacks. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1699–1712. [CrossRef]
3. Jartelius, M. The 2020 Data Breach Investigations Report—A CSO's perspective. *Netw. Secur.* **2020**, *2020*, 9–12. [CrossRef]
4. Clincy, V.; Shahriar, H. Web Application Firewall: Network Security Models and Configuration. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 1, pp. 835–836.
5. Kaur Chahal, J.; Bhandari, A.; Behal, S. Distributed denial of service attacks: A threat or challenge. *New Rev. Inf. Netw.* **2019**, *24*, 31–103. [CrossRef]
6. Xu, X. Cultural communication in double-layer coupling social network based on association rules in big data. *Pers. Ubiquitous Comput.* **2020**, *24*, 57–74. [CrossRef]
7. Hande, Y.; Muddana, A. A Survey on Intrusion Detection System for Software Defined Networks (SDN). In *Research Anthology on Artificial Intelligence Applications in Security*; IGI Global: Hersey, PA, USA, 2021; pp. 467–489.
8. Al-Shaer, E.; Hamed, H.; Boutaba, R.; Hasan, M. Conflict classification and analysis of distributed firewall policies. *IEEE J. Sel. Areas Commun.* **2005**, *23*, 2069–2084. [CrossRef]
9. Hu, H.; Ahn, G.J.; Kulkarni, K. Detecting and resolving firewall policy anomalies. *IEEE Trans. Dependable Secur. Comput.* **2012**, *9*, 318–331. [CrossRef]

10. Saâdaoui, A.; Ben Youssef Ben Souayeh, N.; Bouhoula, A. FARE: FDD-based firewall anomalies resolution tool. *J. Comput. Sci.* **2017**, *23*, 181–191. [CrossRef]
11. Chao, C.S.; Yang, S.J.H. A Novel Mechanism for Anomaly Removal of Firewall Filtering Rules. *J. Internet Technol.* **2020**, *21*, 949–957.
12. Lu, N.; Yang, Y. Application of evolutionary algorithm in performance optimization of embedded network firewall. *Microprocess. Microsyst.* **2020**, *76*, 103087. [CrossRef]
13. Gutierrez, R.J.; Bauer, K.W.; Boehmke, B.C.; Saie, C.M.; Bihl, T.J. Cyber anomaly detection: Using tabulated vectors and embedded analytics for efficient data mining. *J. Algorithms Comput. Technol.* **2018**, *12*, 293–310. [CrossRef]
14. Yin, Y.; Tateiwa, Y.; Wang, Y.; Zhang, G.; Takahashi, N.; Zhang, C. An Analysis Method for IPv6 Firewall Policy. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1757–1762.
15. Lorenz, C.; Clemens, V.; Schrotter, M.; Schnor, B. Continuous Verification of Network Security Compliance. *IEEE Trans. Netw. Serv. Manag.* **2021**, *19*, 1729–1745. [CrossRef]
16. Kim, H.; Ko, S.; Kim, D.S.; Kim, H.K. Firewall Ruleset Visualization Analysis Tool Based on Segmentation. In Proceedings of the 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), Phoenix, AZ, USA, 2 October 2017; pp. 1–8.
17. Lee, H.; Lee, S.; Kim, K.; Kim, H.K. HSViz: Hierarchy Simplified Visualizations for Firewall Policy Analysis. *IEEE Access* **2021**, *9*, 71737–71753. [CrossRef]
18. Ucar, E.; Ozhan, E. The analysis of firewall policy through machine learning and data mining. *Wirel. Pers. Commun.* **2017**, *96*, 2891–2909. [CrossRef]
19. Breier, J.; Branišová, J. A dynamic rule creation based anomaly detection method for identifying security breaches in log records. *Wirel. Pers. Commun.* **2017**, *94*, 497–511. [CrossRef]
20. Vartouni, A.M.; Kashi, S.S.; Teshnehlab, M. An anomaly detection method to detect web attacks using Stacked Auto-Encoder. In Proceedings of the 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems, CFIS, Kerman, Iran, 28 February–2 March 2018; pp. 131–134.
21. Funk, R.; Epp, N.; Cappo, C. Anomaly-based Web Application Firewall using HTTP-specific features and One-Class SVM. *Rev. Eletrônica Argent.-Bras. Tecnol. Inf. Comun.* **2018**. [CrossRef]
22. Moradi Vartouni, A.; Teshnehlab, M.; Sedighian Kashi, S. Leveraging deep neural networks for anomaly-based web application firewall. *IET Inf. Secur.* **2019**, *13*, 352–361. [CrossRef]
23. Togay, C.; Kasif, A.; Catal, C.; Tekinerdogan, B. A Firewall Policy Anomaly Detection Framework for Reliable Network Security. *IEEE Trans. Reliab.* **2022**, *71*, 339–347. [CrossRef]
24. Valenza, F.; Cheminod, M. An Optimized Firewall Anomaly Resolution. *J. Internet Serv. Inf. Secur.* **2020**, *10*, 22–37.