A Firewall Policy Anomaly Detection Framework for Reliable Network Security

Cengiz Togay, Ahmet Kasif, Cagatay Catal, and Bedir Tekinerdogan

***Abstract*—One of the key challenges in computer networks is network security. For securing the network, various solutions have been proposed, including network security protocols and firewalls. In the case of so-called packet-filtering firewalls, policy rules are implemented to monitor changes to the network and preserve the required security level. Due to the dramatic increase of devices, however, and herewith the rapid increase of the size of the policy rules, firewall policy anomalies occur more frequently. This requires careful implementation of the policy rules to ensure cost-efficient solutions for anomaly detection to support network security. In this study, we present an anomaly detection framework for detecting intra-firewall policy anomaly rules. The framework supports the simulation of packets through the firewall ruleset for validating and enhancing the security level of the network. The framework is validated using four different types of firewall policy anomalies. Experimental results demonstrate that the framework is effective and efficient in detecting firewall policy anomalies.**

***Index Terms*—Security, network security, logic programming, packet filtering, anomaly detection, firewall policy.**

I. Introduction

An important challenge for software applications is security, which requires several solutions at different levels, such as device level, network level, and application level. For addressing network-level security, custom network security protocols and different firewalls have been developed. A firewall, which can be hardware, software, or both, is typically used for monitoring the network traffic to allow or block the traffic using a set of rules [28]. Different types of firewalls have emerged [29], such as proxy firewall, packet-filtering firewall, stateful inspection firewall, application-layer firewall, unified threat management (UTM) firewall, next-generation firewall (NGFW), and threat-focused NGFW.

While a packet-filtering firewall does not know whether one packet is related to another packet, a stateful inspection firewall investigates the traffic to determine the context of the packet. Proxy firewall checks the packets at the application layer, and the UTM firewall combines the functionalities of stateful inspection firewalls and intrusion prevention systems (IPS). NGFW firewalls include capabilities of standard firewalls, IPS, and application awareness and control.

In this study, we focus on packet-filtering firewalls, which are frequently used in practice. Although these firewalls are fast, efficient, and inexpensive, they still have several limitations. For instance, they trust the packets about their origin and destination Internet Protocol (IP) addresses. However, by using the IP spoofing technique, hackers can provide fake IP addresses for the network. Moreover, since packet-filtering firewalls are stateless, hackers can exploit this feature and reach the network, which results in frequent network attacks. It is important to detect these anomalies to support the required network security.

In the case of packet-filtering firewalls, policies are created using rulesets. For each incoming and outgoing packet, the action of the first matching rule is triggered. Due to the dramatic increase of devices, however, the size of the rulesets increases, and anomalies regarding the firewall policies occur frequently, and as such, the network is more vulnerable to attacks. In parallel with the increased attack space, the management of the ruleset becomes thus more important and challenging [26].

In this study, a novel Firewall Policy Anomaly Detection (FPAD) model based on logic programming has been developed. In parallel, a corresponding software tool has been developed that can be used to define firewall rulesets, automatically transforms these to logic programming constructs, and discover the intra-firewall anomalies through the use of the FPAD implemented in Prolog [1].

FPAD also allows for importing existing firewall policy configurations and automatically discovering anomalies without human intervention. Further, the framework helps to control firewall policy after manual resolution and to find out if the anomalies are resolved or not. It is also used to find out decaying (unused) rules.

The fields of firewall rules are determined based on the Linux IPTables model [2], and different firewall policies can be imported and transformed into the IPTables model. With the help of this software tool, the administrator can create a firewall policy ruleset either by adding, removing, modifying, swapping rules [3], or importing an existing firewall policy set. These rules are evaluated based on the FPAD model, and corresponding anomalies are discovered.

The contributions of this study are four-fold:

- We proposed a new anomaly detection framework for detecting intra-firewall policy anomaly rules.
- We validated the applicability of the proposed framework using four different types of firewall policy anomalies.

Manuscript received July 25, 2019; revised June 10, 2021. (*Corresponding author: C. Togay.*)

C. Togay is with the Department of Computer Engineering, Uludag University, Bursa, Turkey, e-mail: (ctogay@uludag.edu.tr).

A. Kasif is with the Department of Computer Engineering, Bursa Technical University, Bursa, Turkey, e-mail: (ahmet.kasif@btu.edu.tr).

C. Catal is with the Department of Computer Science and Engineering, Qatar University, Doha, Qatar, e-mail: (ccatal@qu.edu.qa).

B. Tekinerdogan is with the Information Technology Group, Wageningen University & Research, Wageningen, The Netherlands, e-mail: (bedir.tekinerdogan@wur.nl).

- We demonstrated that the logic programming scales better for the implementation of the anomaly detection engines in terms of performance for the large rulesets.
- We proposed an automatic Knowledge Base (KB) generation approach using existing firewall configurations.

The rest of the paper is organized as follows: In Section II, the background and related work are briefly described. In Section III, the adopted methodology is provided. In Section IV, experimental results are explained. In Section V, the conclusion and future work are presented.

II. Background and Related Work

There are numerous approaches to secure a network such as packet-filtering firewalls, stateful firewalls, application layer firewalls, VPNs, dedicated proxy servers, and application proxy gateways [4]. Firewalls have some unique properties including [5]:

- being able to observe all traffic going from the local network to the outside or any packets coming from the outside to the local network
- allowing only authorized connections
- being immune to attacks

As such, they play a critical role in network security and operate in a network for a secure data transfer. Firewalls often include a solution addressing more than one Open Systems Interconnection (OSI) layer or focus on the most critical OSI layer depending on the requirements of the network. Firewalls filter the network traffic by checking several fields such as IP addresses and port numbers. Performance can be affected by the number of rules and different anomalies such as shadowing, redundancy, correlation, generalization, and irrelevancy among rules. In complex networks, including many devices with several use cases (e.g., port openings, IP requests, and corporation policies), the management of the security with the help of firewalls brings several obstacles and challenges. Anomaly-free firewalls can provide better performance in terms of processing speed, and they are less vulnerable to attacks such as Denial of Service (DOS) [6] because they have fewer rules to match for packets passing through the firewall.

A. Types of Firewalls

A packet-filtering firewall can be considered as a rule-based system, which consists of an ordered ruleset and operates in the network and transport OSI layers [7]. Rules consist of fields which can be grouped into filtering and action fields. Our firewall rule modelling approach consists of five filtering fields (i.e., a protocol field, a source IP address, a source port number, a destination IP address, and a destination port number) and an action field. We adopted the Linux IPTables firewall model to represent and process the rules. When a packet arrives, it traverses the ruleset one by one until completely matching the filtering fields of a rule. In such a case, the action of the rule is executed. A packet-filtering firewall policy ruleset is presented in Figure 1.

Stateful firewalls are different from packet filtering firewalls since they have a memory of state, a.k.a., cache [8]. When the first packet of a connection arrives, it is stored in the cache until the connection is destroyed. Subsequent packets of the same connection are directly accepted to the cache, and therefore, they do not need to pass from the firewall rules anymore. One of the major limitations of this approach is the cache overflow, which might cause further Denial of Service (DOS) attacks.

Application layer firewalls are specialized solutions and implemented specifically for a particular application type such as web or Session Initiation Protocol (SIP) firewalls. They provide better security mechanisms for the applications compared to the general-purpose firewalls. The application layer firewalls only need to focus on one application, not a network of devices, including many different types of connections. Also, they can easily control the incoming and outgoing traffic due to easier logging mechanisms. Since it takes a huge amount of time to implement a firewall at the application layer specialized for a particular application, they might not be feasible for some cases.

B. Anomalies

In order to reduce the performance loss in the case of firewalls, researchers conducted several studies. Al Shaer and Hazem [9] designed a tool for firewall policy anomaly detection called Firewall Advisor Tool, which determines the firewall anomalies using the comparison of rule tuples based on imperative programming. For the comparison process, they proposed imperative algorithms to decide which anomaly exists among rules. It is also stated that for a resolution strategy, modifying a firewall is often harder than creating a new firewall, and as such, they proposed an algorithm for creating the firewall ruleset from scratch based on the anomaly data.

In [9][10][27][11], four intra-firewall anomalies are discussed, namely shadowing, redundancy, generalization, and correlation. The first two are especially important because they directly affect firewall performance. In the following descriptions, a lower order means higher precedence. For the shadowing anomaly, a lower order rule (R_x) matches all packets, which can be matched by a higher-order rule (R_y). If they imply different actions for the same packets, and the rule matching is done using the order of the rules, R_x will always intercept the packets. This causes R_y to be unnecessary, which means that it has no impact on the firewall policy. The general applied resolution strategy for shadowing anomaly is to delete the shadowed rule if there are no other anomalies present.

A shadowing anomaly occurs if the following conditions are satisfied. For the case in Figure 1, each filtering field of Rule 5 either includes or is equal to filtering fields of Rule 6, and they have different actions. Therefore, it is said that Rule 5 is shadowing Rule 6.

A rule tuple has a redundant rule only if both of the rules match the same packets which apply the same action. Since a higher-order rule would not match any packet, it is said to be the redundant rule. A general resolution strategy for a redundancy anomaly is to remove the redundant rule from the ruleset. A redundancy anomaly occurs if the following conditions are satisfied:

Order	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
1	TCP	192.152.1.*	ANY	128.172.26.*	80	*ALLOW*
2	TCP	192.152.1.72	ANY	128.172.*.*	80	*DENY*
3	TCP	192.152.1.*	ANY	128.172.26.2	80	*ALLOW*
4	TCP	ANY	ANY	ANY	80	*DENY*
5	TCP	151.*.*.*	ANY	108.56.56.1	53	*DENY*
6	TCP	151.126.*.*	ANY	108.56.56.1	53	*ALLOW*
7	TCP	151.51.37.*	ANY	108.56.*.*	53	*ALLOW*
8	UDP	216.22.14.*	ANY	124.24.*.*	53	*ALLOW*
9	UDP	216.22.14.1	ANY	124.24.*.*	53	*DENY*
10	UDP	216.22.14.1	ANY	124.*.*.*	53	*DENY*
11	UDP	25.12.*.*	ANY	124.*.*.*	ANY	*ALLOW*
12	ANY	ANY	ANY	ANY	ANY	*DENY*

Fig. 1. Firewall policy ruleset.

- Order of R_x is smaller than an order of R_y.
- One of the following conditions is satisfied.
  - Each filtering field of R_x is either a superset of or is equal to each filtering field of R_y, and both rules imply the same action,
  - All of the following conditions are satisfied
    - Each filtering field of R_y is either a superset of or is equal to each filtering field R_x.
    - There must not be a third rule R_z which has an order higher than R_x and smaller than R_y. If there is such R_z, its fields must not be a superset of the fields of R_x or actions of R_x and R_z must be different.

An example of a redundancy anomaly is present between Rule 9 and Rule 10. Each filtering field of Rule 10 either includes or is equal to Rule 9 while both imply the same action. If each field in a lower order rule R_x is either subset or equal to the corresponding field in a higher-order rule R_y, and the actions of R_x and R_y are different, it is said that R_y generalizes R_x. If the following conditions are satisfied, there is a generalization anomaly.

- Order of R_x is less than the order of R_y.
- Each filtering field of R_y is either a superset of or is equal to each filtering field of R_x.
- The actions of R_x and R_y are different.

According to this rule, Figure 1 shows that Rule 4 is a generalized version of Rule 1. These rules have different actions, and each filtering field of Rule 4 either includes or is equal to the filtering field of Rule 1. Correlation anomaly occurs if two rules intersect each other and have different actions. If the following conditions are satisfied, the correlation anomaly occurs.

- Filtering fields of two rules intersect which means that both rules have at least one field which includes the other.
- The actions of R_x and R_y are different.

According to Figure 1, there is a correlation anomaly between Rule 1 and Rule 2 because the source IP field of Rule 1 includes Rule 2, the destination IP field of Rule 2 includes Rule 1, and they also have different actions.

C. Related Work

Bandara et al. [12] proposed a logic-based anomaly detection model. Their proposed approach is different than the other studies because the outcome of the analysis is easily explainable, which can be traced back to the logical derivations, and some anomalies which cannot be detected using other conventional techniques can be identified with their approach.

Al-Shaer et al. [13] extended their previous work to discover inter-firewall anomalies in distributed firewall systems, which many rules can be matched with packets. In their study, these inter-firewall anomalies are classified into the following categories: shadowing, correlation, spuriousness, and redundancy. The discovery and resolution of irrelevancy anomalies have also been studied extensively.

In [14], a data mining technique that can determine decaying and dominant rules has been proposed. The decaying rules are called irrelevant rules because they have no effect on the firewall. In [15], the solution has been extended with a new data mining algorithm, and therefore, the mining process can be performed continuously as new data arrives through the sliding window filtering method.

Valenza and Cheminod [16] proposed a semi-automatic intra-firewall anomaly detection solution. The solution offers anomaly detection, automatic solution for trivial anomalies such as duplication and irrelevance, but leaves the complex ones for the firewall admin to decide. This resolution strategy reduces the burden on firewall admins as well as offering a precise solution.

Valenza et al. [17] also studied the inter-technology anomalies along with inter and intra-firewall anomalies, thus offering a new taxonomy of firewall anomalies. Proposed taxonomy helps discover new inter-technology anomalies, but the resolution is left to firewall admin.

Abbes et al. [18] suggested the use of tree data structure (FAT) for modeling firewall rules. The solution offers automatic detection and manual resolution of intra-firewall anomalies. The FAT is generated from the policy set and designed to show anomalies in leaf nodes. Firewall admins are expected to manually resolve the anomalies. The FAT automatically re-builds itself to reflect the new situation. Each action taken by firewall admins triggers FAT to automatically update itself, reflecting the new situation.

Bodei et al. [19] presented a firewall analysis tool (FWS) which decompiles various low-level firewall configurations into abstract high-level configurations. These configurations are then used to create a firewall policy. Any change in the configuration that affects policy change is automatically reflected. Also, a rule connectivity module is used to help check if there is any rule that can not be reached.

Jaidi [20] studied intra-firewall policy security with respect to risk-trust levels associated with rules and policies. The approach matches rules with various trust and risk levels and according to those, helps manage the policy. The resolution strategy of the study consists of deactivation, deletion, reordering, or rule changing. An overview of the related studies is presented in Table 1.

III. Methodology

For the packet-filtering firewalls, firewall administrators often update firewalls in case of changes to the network for

TABLE I
HIGHLIGHTS OF REVIEWED ANOMALY DETECTION APPROACHES

Approach	Type	Mode	Resolution Strategy	Highlights	Limitations
Al Shaer et al. [9]	Inter Intra	Offline	Manual	State Diagram Anomaly Tree Pairwise Detection	Low Scalability
Bandara et al. [12]	Intra	Offline	Manual	Argumentation Logic Pairwise Detection	High Performance Cost
Golnabi et al. [14]	Intra	Offline	Automatic	Data Mining Dominant & Decaying Rules	Rules need to be manually adapted
Rao et al. [15]	Intra	Online	Automatic	Data Mining Dominant & Decaying Rules Online Runtime	Low number of target anomalies
Valenza and Cheminot [16]	Intra	Offline	Semi-Automatic	Pairwise Detection	Requires Human Intervention
Abbes et al. [18]	Intra	Offline	Semi-Automatic	Anomaly Tree Automatic resolution of trivial anomalies	Manual Resolution
Bodei et al. [19]	Intra	Offline	Automatic	Abstract Representation Policy Compatible Ruleset Generation	Lacks in-depth performance analysis
Jaidi [20]	Intra	Offline	None	Policy Security Trust-Risk Assessment	Basic resolution strategy
Valenza et al. [25]	Inter Intra Intra Technology	Offline	Semi-Automatic	Intra Technology Anomalies Logical Bindings Graph Representation of Anomalies	Manual Resolution of Inter-Technology Anomalies

preserving the required security level. False configurations that might occur during the updates of firewall ruleset can cause policy anomalies and as such, they might adversely impact the firewall throughput and the processing speed.

In this study, we developed a policy anomaly detection model and implemented it as part of our firewall policy anomaly detection platform focusing on the maintenance of packet-filtering firewalls. This platform has been implemented as a Java Web Application using client-server architecture [21] and is based on Servlet [22] structure. Employment of Java platform has many benefits as the Java community provides several development tools for Web application development and also, the systems developed with these development tools provide high-performance and reliability. As shown in Figure 2, the anomaly detection occurs in the server as the base tool is located there. This enables the concurrent serving of many clients. Firewall admins interaction with the server is through Graphical User Interfaces (GUI). The base tool consists of two modules. The first module is called the FPAD module and includes the basic functionality to discover four anomalies explained earlier. The other module is the packet simulation (PS) module. Both modules are implemented using Logic Programming.

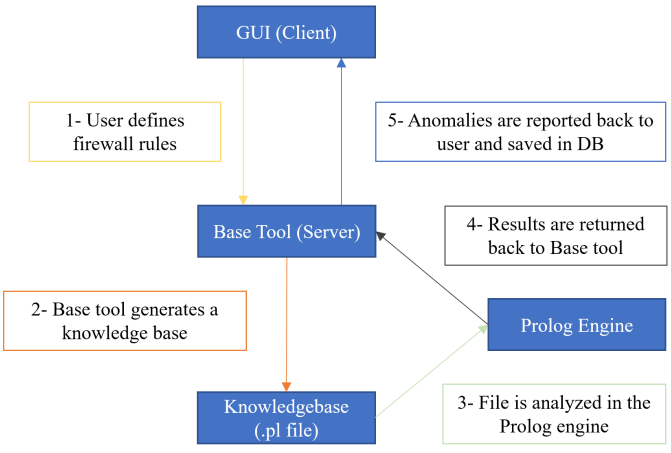

Fig. 2. Workflow of firewall policy anomaly detection platform.

A. Logic Programming

Logic Programming requires some pre-defined 'entities' and 'relationship' information and use this information to discover new relations. The KB proposed in this paper is an example of this .

We selected Prolog as a logic programming language regarding other logic languages because it is easy to develop

programs that require complex data structures, it is very close to the implementation of logic compared to the other languages, its compiler provides good performance and reliability, it is easy to learn the language features in a short time, and it is well-documented. Due to these features of Prolog, it is one of the most popular logic programming languages. For the implementation platform, we preferred the SWI-Prolog platform because it has many features that simplify the development process and it is widely used by researchers and practitioners.

Predicate methods offered by Prolog are utilized to define this prior information. Predicates can be defined using two types of clauses. These are facts and rule clauses. A clause is either a non-conditional relation (i.e., fact clause) or it depends on a condition (i.e., rule clause). After defining this information, the KB can be queried to discover new relations.

A query either returns all matching information or false, stating that no relation can be discovered. The discovery can also consist of many relations at once and the persistence of these relations poses a problem. This is solved through the use of the variable mechanism proposed in Prolog. Variables are defined using words starting with a capital letter or the underscore character such as X, Y, or _var. The variables starting with a capital letter combine any matching information and make them available through the scope of the predicate. However, there might be unnecessary information that should not be received into memory. By defining variables starting with an underscore character, Prolog can omit those unnecessary fields. A KB example is depicted in Figure 3. It consists of character information where characters are named 'a','b', and 'c'. When this KB is queried with *character(X).*, Prolog combines all the matching values with the variable X, prints them, and returns true. For example, the answer for a query like *character(d).* returns false as there is no such prior information of character with an argument of 'd'.

```
character(a).
character(b).
character(c).
```

Fig. 3. Fact clauses with one argument.

B. The FPAD Module

The FPAD module is used to detect anomalies in the given ruleset through logic programming. Firewall policy rules represent non-conditional information and are transformed into fact clauses. The workflow of the FPAD module is visualized in Figure 2. A client-side GUI is implemented for the firewall administrators to define, update, or import firewall ruleset. Base Tool saves the ruleset in the MySQL database management system for persistent storage. Later, using the provided ruleset, a Prolog KB is generated. Finally, anomalies are detected with the help of the Prolog engine which submits queries to the knowledge base. Detected anomalies are returned to the base tool and then, forwarded to the client-side. Detected anomalies are also saved in the database for further analysis.

As one of our contributions, we propose an automatic KB generation from existing firewall configurations. The proposed method is developed for Linux IPTables firewalls. IPTables firewall configurations output a policy file that contains structured textual information. Thus, we employed string operations to convert these files to the KB. Each policy rule in the access list is separately processed and necessary fields are extracted. Generated *policy rule* predicates are shown in Figure 4. By automating the procedure of embedding fact clauses as *policy rule* predicates in the KB, we present a solution for one of the disadvantages of logic programming stated by Bandara et al. [12].

```
policyrule(1,tcp,source([192,152,1,-1]),-1,destination([128,172,26,-1]),80,allow).
policyrule(2,tcp,source([192,152,1,72]),-1,destination([128,172,-1,-1]),80,deny).
policyrule(3,tcp,source([192,152,1,-1]),-1,destination([128,172,26,2]),80,allow).
policyrule(4,tcp,source([-1,-1,-1,-1]),-1,destination([-1,-1,-1,-1]),80,deny).
policyrule(5,tcp,source([151,-1,-1,-1]),-1,destination([108,56,56,1]),53,deny).
policyrule(6,tcp,source([151,126,-1,-1]),-1,destination([108,56,56,1]),53,allow).
policyrule(7,tcp,source([151,51,37,-1]),-1,destination([108,56,-1,-1]),53,allow).
policyrule(8,udp,source([216,22,14,-1]),-1,destination([124,24,-1,-1]),53,allow).
policyrule(9,udp,source([216,22,14,1]),-1,destination([124,24,-1,-1]),53,deny).
policyrule(10,udp,source([216,22,14,1]),-1,destination([124,-1,-1,-1]),53,deny).
policyrule(11,udp,source([25,12,-1,-1]),-1,destination([124,-1,-1,-1]),-1,allow).
policyrule(12,any,source([-1,-1,-1,-1]),-1,destination([-1,-1,-1,-1]),-1,deny).
```

Fig. 4. Firewall rules transformed into *policy rule* predicates

C. Anomaly Detection Predicates

Anomaly detection algorithms pose conditional relations. Thus, they are implemented using rule clause predicates. There are three different relations that two fields (Protocol, IP, or Port) relate with, unification, superset, or irrelevance. In order to find out these relations, a helper predicate *superset* is implemented. The Prolog implementation of the *superset* predicate is shown in Figure 5. The implementation consists of two rule clauses. Both clauses have two list type arguments. List arguments are divided into the head (i.e., first element) and tail parts (i.e., another list with the removed head). Lists contain integer values ranging between 0 and 255. 256 indicates the *any* which represents the complete inclusion of the full range. In each iteration, elements are compared one by one. If the first field contains the value of '256' while the second does not, there is a superset relation. In a state of equivalence, the algorithm search for the same superset relation recursively. If the tail is empty in this situation, the clause returns true, meaning unification.

```
superset([H1|T1],[H2|T2]):-
        T1\=[],
        T2\=[],
        (H1=256,H1\=H2);
        (H1=H2,superset(T1,T2)).

superset([H1|T1],[H2|T2]):-
        (H1=H2;H1=256),
        T1=[],
        T2=[].
```

Fig. 5. Rule clauses of superset predicate.

For example, for the following two lists ([192, 128, 256, 256], [192, 128, 256, 172]) which are used for IP addresses,

it is clear that the first list is a superset for the latter. The first three fields are equivalent, thus the first *superset* rule clause unifies and the predicate runs recursively. The last fields of each list are different and the first list has a value of 256. This satisfies the second rule clause and indicates a superset relation.

Anomaly detection predicates make field-wise analyzes between rule pairs. The state diagram for both four predicates is depicted in Figure 6. The analysis starts with checking the existence of the *policyrule* fact clauses. Field-wise analyzes are performed only if both of the clauses with the given order numbers exist. Then, the fields of both policy rules are loaded into the variables. The variable names declared in these rule clauses are the same for all the definitions. "P" expresses Protocol, "$SrcIP_1$" and "$SrcIP_2$" shows source IP addresses, "SP_1" and "SP_2" means port addresses, "$DestIP_1$" and "$DestIP_2$" point to destination IP addresses, "DP_1" and "DP_2" are destination port addresses, and "Act_1" and "Act_2" holds actions.

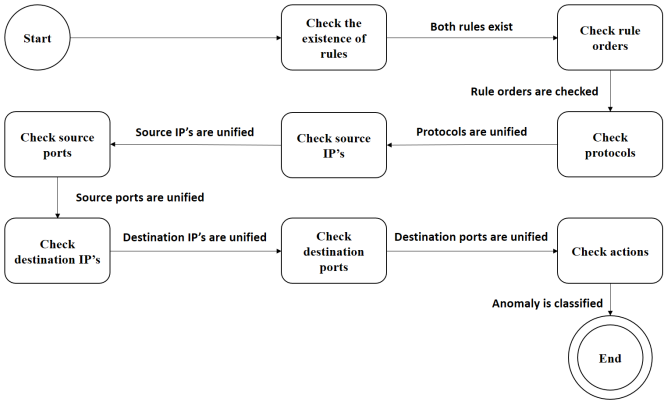

Fig. 6. State diagram for anomaly detection predicates

The Prolog rule clause for the redundancy predicate is represented in Figure 7. The protocol and action values are combined, whereas the other values are loaded into the variables for further analysis. In the following steps, these variables are compared with each other. They are two different conditions for the detection of the redundancy anomaly. If one of them is satisfied, the predicate returns true. In the first condition, each field of R_h must either be a superset of or equal to each field of R_h. In the second condition, each field of R_h must be either a superset of or equal to each field of R_l. Also, in the second condition, there must not be a third rule R_m with an order larger than R_l, and smaller than R_h while also implying the same action with the R_l.

The Prolog rule clause of the generalization predicate is represented in Figure 8. The protocol values are combined and the other values are loaded into the variables for further analysis. In the following steps, these variables are compared against each other to determine whether the relation is a superset relation or not because each field of R_h must be either a superset of or equal to each field of R_l in order to form a generalization anomaly. Also, if one of the fields of R_h is in a subset relation with R_l, there is no generalization anomaly. The equality and superset conditions are evaluated

```
redundancy(O1,O2):-
 policyrule(O1,P,src(SrcIP1),SP1,dest(DestIP1),DP1,Act),
 policyrule(O2,P,src(SrcIP2),SP2,dest(DestIP2),DP2,Act),
 policyrule(O3,P,src(SrcIP3),SP3,dest(DestIP3),DP3,Act3),
 (
  (
   (SrcIP1=SrcIP2;superset(SrcIP1,SrcIP2)),
   (DestIP1=DestIP2;superset(DestIP1,DestIP2)),
   (SP1=SP2;superset(SP1,SP2)),
   (DP1=DP2;superset(DP1,DP2))
  );
  (
   (SrcIP1=SrcIP2;superset(SrcIP2,SrcIP1)),
   (DestIP1=DestIP2;superset(DestIP2,DestIP1)),
   (SP1=SP2;superset(SP2,SP1)),
   (DP1=DP2;superset(DP2,DP1)),
   O1<O3,
   O3<O2,
  )
 ),
 O1<O2,
 Act\=Act3.
```

Fig. 7. Rule clause of redundancy predicate.

using the Prolog built-in equality predicate and our custom superset predicate. Finally, the variables of Act_1 and Act_2 are checked as the actions of the rules should be different.

```
generalization(O1,O2):-
 policyrule(O1,P,src(SrcIP1),SP1,dest(DestIP1),DP1,Act1),
 policyrule(O2,P,src(SrcIP2),SP2,dest(DestIP2),DP2,Act2),
 (SrcIP1=SrcIP2;superset(SrcIP2,SrcIP1)),
 (DestIP1=DestIP2;superset(DestIP2,DestIP1)),
 (SP1=SP2;superset(SP2,SP1)),
 (DP1=DP2;superset(DP2,DP1)),
 Act1\=Act2,
 O1<O2.
```

Fig. 8. Rule clause of generalization predicate.

The Prolog rule clause for shadowing predicate is represented in Figure 9. The protocol values are combined and the other values are loaded into the variables. In the following steps, these variables are compared against each other to determine whether the relation is a superset relation or not because each field of R_l must be either a superset of or equal to each field of R_h in order to form a shadowing anomaly. If one of the fields of R_l is in a subset relation with R_h, there is no shadowing anomaly. The equality and superset checks are evaluated with the help of the Prolog built-in equality predicate and our custom superset predicate. Finally, the variables of Act_1 and Act_2 are checked as the actions of the rules should be different.

The Prolog rule clause of the correlation predicate is shown in Figure 10. The protocol values are combined, and the other values are loaded into variables. Then, it is searched if both rules have at least one field which is in a superset relation with the corresponding field of the other rule. Finally, the variables of Act_1 and Act_2 are checked as the actions of the rules should be different.

```
shadowing(O1,O2):-
  policyrule(O1,P,src(SrcIP1),SP1,dest(DstIP1),DP1,Act1),
  policyrule(O2,P,src(SrcIP2),SP2,dest(DstIP2),DP2,Act2),
  (SrcIP1=SrcIP2;superset(SrcIP1,SrcIP2)),
  (DestIP1=DestIP2;superset(DestIP1,DestIP2)),
  (SP1=SP2;superset(SP1,SP2)),
  (DP1=DP2;superset(DP1,DP2)),
  Act1\=Act2,
  O1<O2.
```

Fig. 9. Rule clause of shadowing predicate.

```
correlation(O1,O2):-
  policyrule(O1,P,src(SrcIP1),SP1,dest(DstIP1),DP1,Act1),
  policyrule(O2,P,src(SrcIP2),SP2,dest(DstIP2),DP2,Act2),
  (
    superset(SrcIP1,SrcIP2);
    superset(DstIP1,DstIP2);
    superset(SP1,SP2);
    superset(DP1,DP2)
  ),
  (
    superset(SrcIP2,SrcIP1);
    superset(DstIP2,DstIP1);
    superset(SP2,SP1);
    superset(DP2,DP1)
  ),
  Act1\=Act2.
```

Fig. 10. Rule clause of correlation predicate.

D. PS Module

The PS module is the second module that behaves like a real-life firewall. The module accepts packet information and simulates this packet through the previously defined rules of the firewall. After the simulation, a detailed log of the process with the matching rule is returned to the client and the results are rendered at the client GUI. Every rule that would match is contained in the logs. This feature is used for two functionalities. It helps to find decaying rules and also if the anomalies are resolved after manual resolution.

Packet simulation module uses two predicates, evaluate and test, given in Figure 11. Evaluate predicate takes 6 parameters, consisting of information about the packet and the RuleId, which will be returned back to the user. The predicate checks if there is a rule where every field of that rule is in a superset relation with the values from the packet. When a rule is unified, it's RuleId becomes accessible from the caller predicate. While this predicate is sufficient enough to find the first matching rule, a secondary predicate 'test' is implemented to find out all rules which can possibly intercept the simulated packet. This, while enabling us to see if the packets match with the expected rules, also gives us the possibility to find and analyze 'decaying' rules.

IV. Experimental Results

A comparative runtime analysis between the proposed approach and the work of Al-Shaer and Hamed [9] is conducted. We compared our contribution with this study because their techniques such as the use of tree data structure and the pairwise detection are still valid, provide insights for further research, and although it has been published in 2004, applied anomaly detection principles are still valid for the benchmark.

```
evaluate(RuleId,Protocol,PktSrcIP,PktSP,PktDestIP,PktDP):-
  policyrule(
    RuleId,
    Protocol,
    source(RuleSrcIP),
    RuleSP,
    destination(RuleDestIP),
    RuleDP,

    _
  ),
  superset(RuleSrcIP,PktSrcIP),
  superset(RuleDestIP,PktDestIP),
  superset(RuleSP,PktSP),
  superset(RuleDP,PktDP).

test(L,Protocol,PktSrcIP,PktSP,PktDestIP,PktDP):-
  findall(X,evaluate(X,Protocol,PktSrcIP,PktSP,PktDestIP,PktDP),L).
```

Fig. 11. Rule clauses of evaluate and test predicates.

A Java application is implemented to procedurally generate firewall configurations for large-scale analysis. The application does not follow a certain firewall policy. It rather generates a specified number of firewall rules and modifies some user-defined percentage of them to include anomalies. Ruleset sizes are selected between 500-5000 rules. Experiments were conducted on a desktop computer, which has an Intel Core i5-3470 3.2GHz processor and 8 GB RAM.

The algorithm proposed in the work of Al-Shaer and Hamed [9] has been implemented in Java using the Object-Oriented Programming paradigm. The runtime performance comparison between this work [9] and the proposed FPAD approach is given in Figure 12. While the x-axis shows the number of rules, the y-axis depicts the required time to find anomalies. These results demonstrate that, while in small configurations both algorithms remain competent, our Prolog-based logic programming model excels at bigger configurations (i.e., 3000 rules). For complex networks, where many devices are employed in a single network to operate together, the performance of our model is more preferable compared to [9]. The proposed algorithm also has the advantage over the algorithm proposed in [12] by having the automatic KB generation, thus neglecting the error-prone human intervention on the creation of KB.

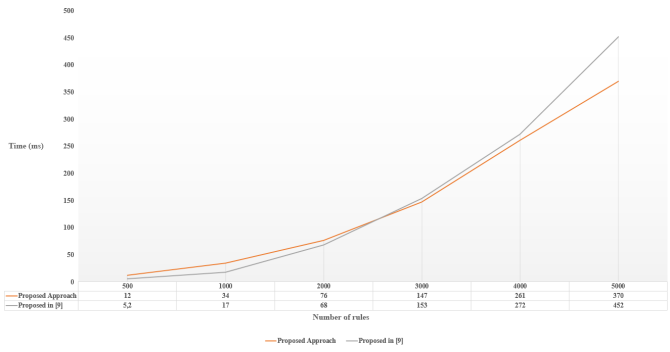

Fig. 12. Runtime comparison between proposed FPAD algorithm and the algorithm proposed in [9].

We also analyzed the standalone runtime performance of each Prolog anomaly detection predicate. The results are presented in Figure 13. The x-axis of this figure shows the rule size and the y-axis represents the required time based on the selected rule size. As shown in this figure, when the rule size increases, the required time to process the dataset increases, as expected. For each anomaly, the required time is different, which means that the detection of some anomalies is more costly or cost-efficient depending on the required time. Figure 13 shows that the detection of the shadowing anomaly is the most cost-efficient one, the redundancy anomaly is the second cheapest predicate, the generalization anomaly is the third one, and the correlation predicate is the most costly anomaly. In addition, we experimented with a different number of anomalies, however, we find out that an increased number (or percentage) of anomalies in a firewall configuration does not have a major impact on the runtime performance of these algorithms.

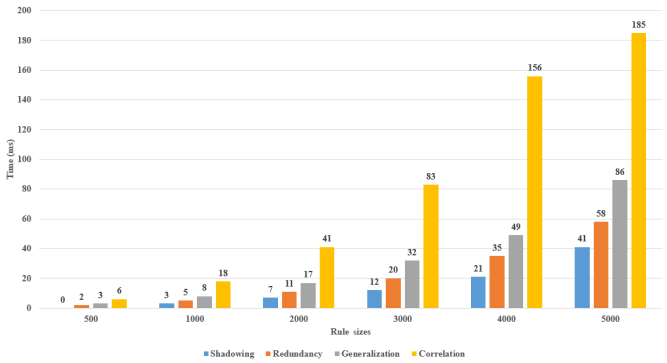

Fig. 13. Anomaly detection time distribution per anomaly for the proposed algorithm.

In terms of time complexity, both algorithms perform an exhaustive approach as they are both pairwise algorithms. Comparisons between rule fields do not change the complexity class, as the number of fields is finite. For this reason, the time complexity of both algorithms is in O(n^2).

From the perspective of accuracy, extensive test through our logic programming is conducted with firewall rulesets. The rulesets differ in sizes and configurations and the anomalies are known prior. In Figure 14, we show the experimental results of this extensive test. Although the rule size is increased 10 times from 500 rules to 5000 rules, the performance change per anomaly is not affected too much. This indicates that the detection approach is applicable in real-world scenarios that include too many rules. The best performance is achieved for shadowing anomaly (i.e., average 92%) and the least performance is achieved for generalization anomaly (i.e., average 78%). As shown in Figure 14, the overall accuracy for detected anomalies is 84%, which shows that the proposed approach is able to detect different kinds of anomalies at an acceptable performance. The method does not produce any false-positives.

We have also performed experiments for the packet simulation module, which is queried with packets to find out decaying rules. Results show that the algorithm runs in linear

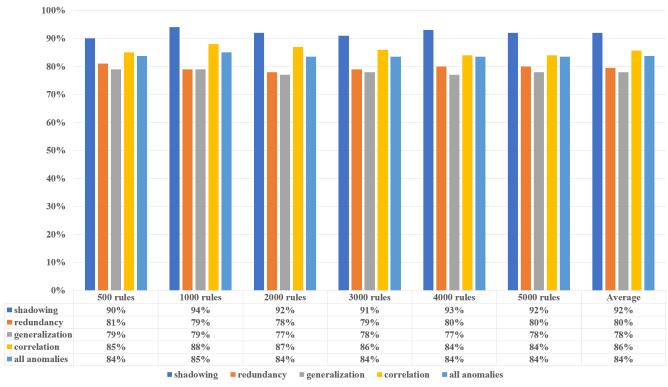

Fig. 14. FPAD anomaly detection accuracy

time to find all the intercepting rules. Figure 15 shows the runtime performance of the Packet Simulation module and indicates that time complexity stays linear. Even at rule sizes of 5000, analysis completes in less than 8ms.

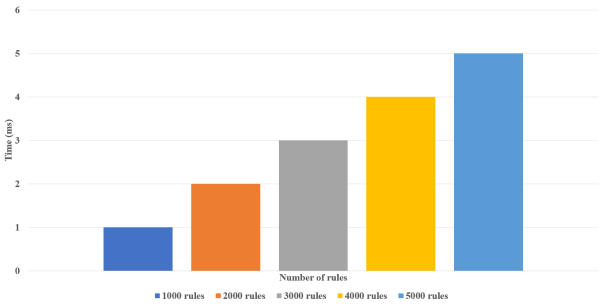

Fig. 15. The runtime performance of the PS module

Using the FPAD module of the presented tool we were able to effectively detect firewall policy anomalies for a large number of rules. In addition, the proposed FPAD module provides a similar accuracy compared to the algorithm proposed in [9] in case of anomaly detection. The resulting anomaly table for the ruleset given in Figure 1 is presented in Figure 16.

#	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}
R_1	-	C	R	G	-	-	-	G	-	-	-	-
R_2	-	-	C	R	-	-	-	-	-	-	-	R
R_3	-	-	-	G	-	-	-	-	-	-	-	G
R_4	-	-	-	-	-	-	-	-	-	-	-	R
R_5	-	-	-	-	-	S	C	-	-	-	-	R
R_6	-	-	-	-	-	-	-	-	-	-	-	G
R_7	-	-	-	-	-	-	-	-	-	-	-	G
R_8	-	-	-	-	-	-	-	-	S	C	-	G
R_9	-	-	-	-	-	-	-	-	-	R	-	R
R_{10}	-	-	-	-	-	-	-	-	-	-	-	R
R_{11}	-	-	-	-	-	-	-	-	-	-	-	G
R_{12}	-	-	-	-	-	-	-	-	-	-	-	-

S: Shadowing, R: Redundancy, G: Generalization, C: Correlation, - : No Anomaly

Fig. 16. Anomaly detection precision table for proposed FPAD algorithm.

V. Conclusion

In a large organization, thousands of rules might be defined by different firewall managers having various levels of experience. In this study, we proposed a firewall policy anomaly

detection model to identify different kinds of anomalies by using logic programming. We implemented a packet-filtering firewall anomaly detection tool including two sub-modules, the FPAD module, and a packet simulation module. Also, we presented the approach for automated generation of firewall rules (i.e., configurations) as a Prolog file in order to evaluate these against anomalies. We implemented our tool using the Prolog logic programming language and observed that logic programming scales better for the implementation of the anomaly detection engines in terms of the performance for the large rulesets (i.e., 10000 or more). Also, since the logic programming code is written based on logical derivations, it makes the code much more readable and easier to maintain. The ability to use the solution as a cloud service was also verified by creating a prototype, which serves many clients concurrently [23], [24]. As future work, we plan to extend our work by implementing an anomaly resolution module [20] and developing support for the analysis and resolution of irrelevancy anomalies [21] which are not currently supported.

REFERENCES

[1] W. F. Clocksin and C. S. Mellish, *Programming in Prolog: Using the ISO standard*: Springer Science & Business Media, 2012.

[2] G. N. Purdy, *Linux iptables Pocket Reference: Firewalls, NAT & Accounting*: O'Reilly Media, Inc., 2004.

[3] A. X. Liu, *Firewall policy change-impact analysis*: ACM Transactions on Internet Technology, Vol. 11, No. 4, 15, 2012. ACM.

[4] S. Karen, and H. Paul, *Guidelines on firewalls and firewall policy*: NIST Recommendations, SP, 800–41, 2008.

[5] S. M. Bellovin, M. Steven and W. R. Cheswick, *Network firewalls*: IEEE communications magazine, Vol. 32, No. 9, 50–57, 1994.

[6] C. L. Schuba, I. V. Krsul and M. G. Khun, E. H. Spafford, H. Eugene, A. Sundaram, D. Zamboni *Analysis of a denial of service attack on TCP*:IEEE Symposium on Security and Privacy 208–223, 1997.

[7] H. Zimmermann, *OSI reference model–The ISO model of architecture for open systems interconnection*, IEEE Transactions on communications, Vol. 28, No. 4, 425–432, 1980.

[8] A. Wool, *Packet filtering and stateful firewalls*, Handbook of Information Security, Vol. 3, 526–536, 2006.

[9] E. S. Al-Shaer, H. H. Hamed, *Modeling and management of firewall policies*, IEEE Transactions on Network and Service Management, Vol. 1, No. 1, 2–10, 2004.

[10] R. Yadav, H. Kaur and A. Saurabh, *Design of Tool to Detect Anomalies in Firewall Rules*, Indian Journal of Science and Technology, Vol. 9, No. 47, 2016.

[11] E. S. Al-Shaer, H. H. Hamed, *Discovery of policy anomalies in distributed firewalls*:Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies 2605–2616, 2004.

[12] A. K. Bandara, A. C. Kakas, E. C. Lupu, A. Russo, *Using argumentation logic for firewall configuration management*: IFIP/IEEE International Symposium on Integrated Network Management 180–187, 2009.

[13] E. Al-Shaer, H. Hamed, R. Boutaba, M. Hasan, *Conflict classification and analysis of distributed firewall policies*: IEEE journal on selected areas in communications, Vol. 23, No. 10, 2069–2084, 2005.

[14] K. Golnabi, R. K. Min, L. Khan, E. Al-Shaer, *Analysis of firewall policy rules using data mining techniques*:Network Operations and Management Symposium 305–315, 2006.

[15] S. Rao, B. R. Rama, and K. N. Mani, *Firewall Policy Management Through Sliding Window Filtering Method Using Data Mining Techniques*, International Journal of Computer Science and Engineering Survey, Vol. 2, No. 2, 39–55, 2011. Academy & Industry Research Collaboration Center(AIRCC).

[16] F. Valenza and M. Cheminod, *An Optimized Firewall Anomaly Resolution*, Journal of Internet Services and Information Security (JISIS), Vol. 10, No. 1, 318-323, 2020.

[17] F. Valenza, C. Basile, D. Canavese and A. Lioy, *Classification and analysis of communication protection policy anomalies*, IEEE/ACM Transactions on Networking, 25(5), 2601-2614, 2017.

[18] T. Abbes, A. Bouhoula and M. Rusinowitch, *Detection of firewall configuration errors with updatable tree*, International Journal of Information Security, 15(3), 301-317 (2016)

[19] C. Bodei, P. Degano, R. Focardi, L. Gavlletta, M. Tempesta and L. Veronese, *Firewall Management With FireWall Synthesizer*, CEUR WORKSHOP PROCEEDINGS, vol. 2058, no. 16, 2018.

[20] F. Jaidi, *A Quantified Trust-Risk Assessment Approach for Enhancing Firewalls-Filtering Services*, Journal of Information Assurance & Security, 14(2), 2019.

[21] A. Berson, *Client-server architecture*, McGraw-Hill, 1992.

[22] J. Hunter and W. Crawford, *Java Servlet Programming: Help for Server Side Java Developers*, O'Reilly Media, Inc., 2001.

[23] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, I. Ali, and M. Guizani, *A survey of machine and deep learning methods for internet of things (IoT) security*, IEEE Communications Surveys & Tutorials, 2020.

[24] M. M. Othman and A. El-Mousa, *Internet of Things & Cloud Computing Internet of Things as a Service Approach*: 11th International Conference on Information and Communication Systems (ICICS) 318–323, 2020.

[25] E. Karafili, F. Valenza, Y. Chen, and E. Lupu, *Towards a framework for automatic firewalls configuration via argumentation reasoning*, 2020.

[26] A. Saâdaoui, N. B.Y. B. Souayeh, and A. Bouhoula, *FARE: FDD-based firewall anomalies resolution tool*. Journal of computational science, 23, 181-191, 2017.

[27] S. Alsehibani and S. Almuhammadi, *Anomaly detection: Firewalls capabilities and limitations*. 2018 International Conference on Computing Sciences and Engineering (ICCSE), 1–5, 2018.

[28] F. Jaidi, *FW-TR: Towards a Novel Generation of Firewalls Based on Trust-Risk Assessment of Filtering Rules and Policies*. 15th International Wireless Communications & Mobile Computing Conference (IWCMC), 1043–1048, 2019.

[29] L. W. Thwin and Z. M. Aye, *Classification and Discovery on Intra-Firewall Policy Anomalies*. National Journal of Parallel and Soft Computing, 1(1), 235–242, 2019

[30] M. Rezvani, A. Ignjatovic, M. Pagnucco and S. Jha, *Anomaly-free policy composition in software-defined networks*. In 2016 IFIP Networking Conference (IFIP Networking) and Workshops, 28–36, 2016.

Cengiz Togay is an assistant professor at the Computer Engineering department in Bursa Uludag University. He received PhD degree in Computer Engineering from the Middle East Technical University, Ankara, Turkey, in 2008. His research interests include component-oriented software engineering, WebRTC based secure communications, smart cards, privacy, and the Internet of Things. He has served in the program committees of various international conferences.

Ahmet Kasif Ahmet Kasif is a research assistant at the department of Computer Engineering in Bursa Technical University. He received his MSc at Computer Engineering from Bursa Uludag University in 2020. He is currently pursuing his PhD degree in Computer Engineering at Istanbul University Cerrahpasa. His research interests include information security, IoT, machine learning, and computer music.

Cagatay Catal is a faculty member at the department of computer science and engineering in Qatar University. He worked as a full professor at the department of computer engineering in Bahcesehir University between 2020 and 2021. Before joining Bahcesehir University, he worked two years at Wageningen University & Research in the Netherlands and previously, he worked six years at the Department of Computer Engineering in Istanbul Kultur University as Associate Professor and the Head of the Department. Prior to join the academia, he worked 8 years at the Scientific and Technological Research Council of Turkey (TUBITAK), Information Technologies Institute as Senior Researcher & Project Manager. His research interests include reliability, software quality, artificial intelligence, machine learning, deep learning, and software engineering.

Bedir Tekinerdogan is a full professor and the chairholder of the Information Technology group at Wageningen University & Research in the Netherlands. He has more than 25 years of experience in information technology and software engineering. He is the author of more than 300 peer reviewed scientific papers. He has been active in dozens of national and international research and consultancy projects with various large software companies whereby he has worked as a principal researcher and leading software/system architect. His research interests include software and systems engineering, data science, and empirical software engineering.