Wall-PROV: Revisiting Firewall Rule Misconfigurations with Data Provenance and Verifying the Provenance Graph Properties

Abdullah Al Farooq
Wentworth Institute of Technology
farooqa@wit.edu

Tanvir Rahman Akash
Trine University
tanvir.rahman@my.trine.edu

Manash Sarker
Patuakhali Science and Technology University
manash.sarker@pstu.ac.bd

***Abstract*—Firewall rule misconfigurations is a very-well known challenge in network security management. It often leads to unintended access control behavior, storage misuse, unnecessary management overhead, and performance degradation. Existing approaches primarily rely on static rule analysis and are limited in their ability to explain how misconfigurations manifest during actual firewall execution. In this paper, we propose a provenance-based method for detecting firewall rule misconfigurations by reconstructing causal relationships between network traffic, firewall rules, and filtering decisions using firewall logs. Our methodology enables the systematic detection of well-acknowledged firewall misconfigurations, including shadowing, redundancy, generalization, specialization, and correlation. To ensure completeness and soundness, we formally specify the provenance model and prove key structural properties, including acyclicity, using the F* verification framework.**

We evaluate our approach on an OPNsense firewall with some misconfigured rule sets and demonstrate that it detects all conflicts with negligible runtime and storage overhead. The results show that data provenance provides an effective and viable method for analyzing firewall misconfigurations.

I. Introduction

A Firewall is a fundamental defense component for a network. It is the first line of defense that plays a major role by filtering unwanted network traffic. A typical firewall makes a decision on whether or not it should allow traffic to a network using a few criteria: source IP, destination IP, port, and protocol. Though there have been newer defense systems such as Next Generation Firewalls, Intrusion Prevention Systems, and proxy services which work more intelligently to prevent unwanted traffic to reach a network, the importance of firewall cannot be overlooked. A firewall is still the first gatekeeper of a network. For example, let us assume there is an e-commerce business in USA where it can only deliver items inside the United States only. So, there is no need to accept traffic coming from outside the United States. A Firewall performs this task very efficiently by accepting only the traffic that is coming from USA. Because there are very few fields to check in a network traffic, the timing overhead a firewall adds is negligble. Conceptually, a firewall establishes a controlled interface between an internal network (typically considered secure or trusted) and external networks (it may be untrusted or partially trusted), such as the public Internet. It is very common for any organization to place their only firewall immediately after the border router of their network [1].

However, a firewall's effectiveness critically depends on the correctness of the rule set. Modern enterprise networks deploy hundreds to thousands of rules, often accumulated over years of policy changes, administrator turnover, and ad hoc updates. As a result, rule sets commonly contain subtle and unintentional misconfigurations – including shadowing, redundancy, generalization, specialization, and correlation anomalies [2], [3]. These misconfigurations cause legitimate traffic to be incorrectly blocked, and malicious traffic to be inadvertently allowed. It is also possible that firewall rules that are placed later in the order, become partially or entirely ineffective. Prior research have shown that even small networks exhibit measurable rates of rule misconfigurations which underscores how difficult it is to reason about the dependencies among rules manually. There have been many research works that focus particulary on formal method to detect such firewall rule conflicts. All the firewall rules are inputted into a formal model tool where rule conflicts are defined. These approaches do not require any input data to detect rule misconfigurations. However, firewall rule sets are continuously evolving, as new rules are added and existing rules are modified or removed during normal operation. Consequently, formal verification of misconfigurations is not performed every time the rule set changes. Instead, it is more common for security audit teams to assess the effectiveness and correctness of security controls, such as firewalls or intrusion detection systems, using log records.

A large volume of firewall log data is available for security auditing. Data provenance provides an effective way to parse log records and generate graphs that can be queried for analysis. Such queries may involve verifying a specific security incident or reconstructing an abnormal or attack path. Moreover, data provenance is gaining popularity among security experts for intrusion detection and attack path reconstruction. Therefore, we believe that our method can assist security

Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM) 2026
23 February 2026, San Diego, CA, USA
ISBN 978-1-970672-00-8
https://dx.doi.org/10.14722/prism.2026.23025
www.ndss-symposium.org

audit teams by enabling log-based analysis within a unified provenance-driven platform.

In this work, we propose `Wall-PROV`, a system that generates provenance graphs to detect firewall rule conflicts from firewall log records, even when such conflicts leave no direct evidence in the logs. For example, a shadowed firewall rule does not appear in log records because it is never applied during packet filtering. Our approach addresses this limitation by reconstructing such unseen situations using data provenance.

As this is the first approach to employ provenance graphs for detecting firewall rule misconfigurations, we formally verify key properties of the provenance model, including completeness and soundness, using the F* theorem prover.

II. Background

In this section, we describe firewalls and their rule misconfigurations based on the current state of the art. Although such misconfigurations have been formally defined in prior work, we restate them using F*, as this is the language used to verify properties of our provenance graph. Later in this section, we introduce data provenance and explain why it is well suited for reconstructing firewall rule misconfigurations. Throughout the paper, we use the terms "conflicts" and "misconfigurations" interchangeably.

A. Firewall and its Rule Conflicts

A firewall is a fundamental network security mechanism that enforces access control policies by inspecting traffic based on an ordered set of rules[4]. It operates by inspecting packets by comparing their attributes—such as source and destination IP addresses, port numbers, and protocols—against an ordered set of firewall rules [4], [5]. Each rule specifies an action: to allow or block the traffic. When a packet arrives at the firewall, rules are evaluated sequentially until the first matching rule is found, and the corresponding action is applied [6]. This ordered evaluation model makes firewall behavior highly sensitive to rule placement, and improper ordering can lead to anomalies such as shadowing, redundancy, or conflicts, which may degrade both security and performance.

Proper configuration of these rules is essential, as incorrect, redundant, or conflicting rules can undermine both performance and security [2], [7]. Prior research has shown that misconfigurations and rule anomalies are common in real-world firewall policies, often leading to unintended behavior such as blocking legitimate traffic or permitting malicious traffic [7], [8]. Al-Shaer and Hamed developed a systematic classification of firewall rule anomalies and demonstrated efficient techniques for detecting and resolving such conflicts [7], [2]. Subsequent studies have explored automatic anomaly detection, policy visualization, and formal verification of firewall policies to aid administrators in managing increasingly complex rule sets [9], [8].

Blocking is different from rejecting in a way that after rejecting, the source of the traffic is notified while it is not the case for the former. As shown in Fig. 1, incoming traffic is evaluated against an ordered set of firewall rules before being forwarded or blocked.

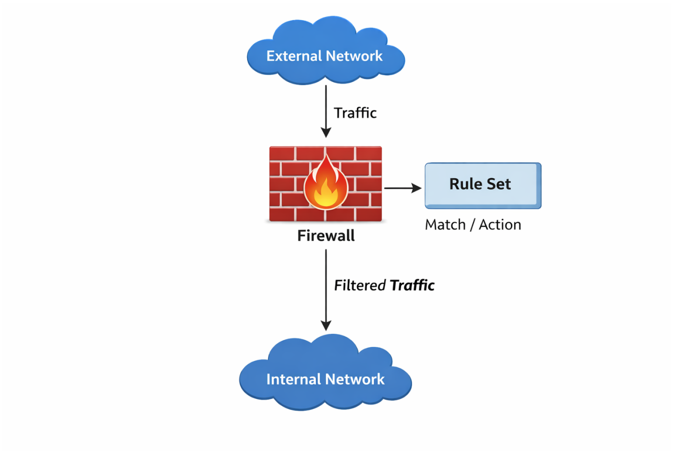

Fig. 1: High-level firewall architecture

Firewall rule anomalies refer to unintended interactions among filtering rules that result in incorrect or ambiguous policy enforcement. Al-Shaer and Hamed classified these anomalies based on how rule conditions overlap and how rule ordering affects packet filtering decisions. The following subsections describe the major types of firewall rule anomalies.

1) Shadowing Anomaly: A shadowing anomaly occurs when a firewall rule is completely overridden by a preceding rule with overlapping conditions and a different action. In this case, all packets that match the shadowed rule are already matched by the earlier rule, preventing the shadowed rule from ever being applied.

2) Redundancy Anomaly: A redundancy anomaly arises when a firewall rule is fully covered by a previous rule that specifies the same action. Because the earlier rule already enforces the same decision for all matching packets, the redundant rule does not contribute to policy enforcement and can be safely removed without affecting firewall behavior. Redundant rules increase policy size and complicate rule management.

3) Generalization Anomaly: A generalization anomaly appears when a more general rule follows a more specific rule but defines a conflicting action. Although the specific rule is applied first, the later general rule may unintentionally override the intended behavior for packets not explicitly covered by the earlier rule.

4) Correlation Anomaly: A correlation anomaly occurs when two rules partially overlap in their matching conditions and specify different actions. In this case, some packets match both rules, while others match only one of them. The resulting behavior depends on rule ordering and packet attributes, making policy enforcement difficult to predict.

B. Data Provenance

Data provenance is defined as the history of data transformed by a system" [10]. The provenance of a piece of data describes what the inputs and outputs of each process are,

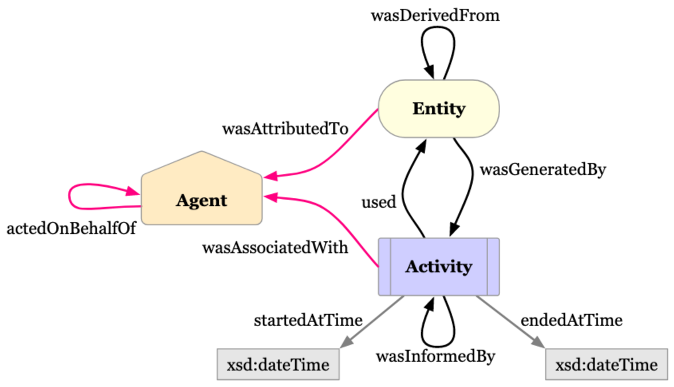

Fig. 2: A basic provenance graph

what processes were executed, and who had control of those processes during execution. Provenance is often represented as a directed acyclic graph (DAG) with nodes representing the data, processes, and controlling entities [11]. The edges represent causal relationships between these nodes.

Provenance was first introduced in databases and computational sciences for tracing and debugging. However, more recently it has been proposed as a primitive for building secure and resilient systems [?] that can "fight through" attacks. In order to provide such capabilities, novel collection, storage, and analysis mechanisms have been proposed to enable near-real-time analysis of provenance to support security and resilience decisions [?]. These mechanisms are being used to provide forensic analysis and intrusion detection capabilities [?].

III. Wall-PROV Architecture

In this section, we provide a high-level overview of our approach. Our tool takes two inputs: (1) a file containing all firewall rules and (2) a log file that records all traffic allowed or blocked by the firewall.

Both files are obtained from the OPNsense hardened BSD filesystem. It is important to note that OPNsense provides full visibility into traffic arriving at an interface, such as LAN or DMZ. That is, the firewall inspects each network packet and records the outcome of the filtering decision, whether the packet is allowed, blocked, or rejected.

As part of auditing firewall's efficiency, our proposed `Wall-PROV` will look into the firewall log files to see which firewall rules have filtered which network packet. A typical raw log entry from `/var/log/filter.log` is presented in Listing 1. This comma-separated value (CSV) format is used for internal processing and remote syslog transmission.

```
82,,,1000002665,igb0,match,block,in,4,0x0
    ,,64,28145,0,DF,6,tcp
    ,60,192.168.1.100,45.33.2.1,54321,443,0,S
    ,142345,1024,,
```

Listing 1: Raw OPNsense TCP Log Entry

The fields are parsed sequentially. Table I provides the field-by-field explanation according to standard OPNsense/pfSense raw filter log formatting.

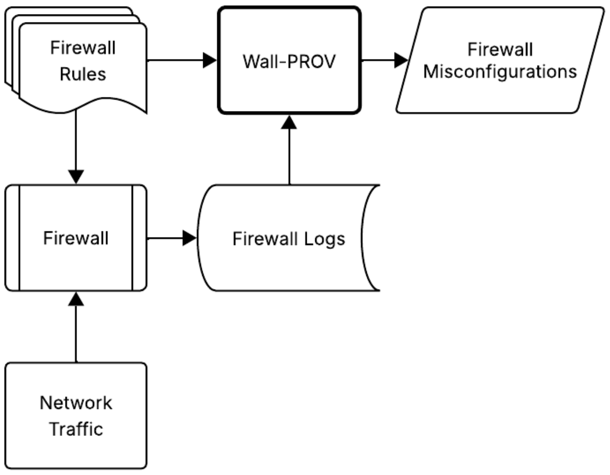

Fig. 3: High-level `Wall-PROV` Architecture

TABLE I: OPNsense Firewall Log Field Descriptions

Pos	Field Name	Example Value
1	Rule Number	82
2-3	Sub-rule/Anchor	(Empty)
4	Tracker ID	1000002665
5	Interface	igb0
6	Reason	match
7	Action	block
8	Direction	in
9	IP Version	4
10-15	IP Header Data	0x0, 64, 28145, etc.
16	Protocol ID	6 (TCP)
17	Protocol Text	tcp
18	Packet Length	60
19	Source IP	192.168.1.100
20	Destination IP	45.33.2.1
21	Source Port	54321
22	Destination Port	443
23	Data Length	0
24	TCP Flags	S (SYN)

`Wall-PROV` takes IPs, ports, and protocols from the log files and then compares them with all firewall rules.

Axiom 1 (All Network Packets are logged) Every network packet recorded in the firewall log has at least one firewall rule applied to it.

```
assume has_applied_rule (p: packet) =
  exists r. applied p r
```

This axiom states that if a network packet is present in the firewall log, then there exists at least one firewall rule that was applied to that packet.

A. *Firewall Rules in OPNsense*

OPNsense is an open-source firewall and routing platform based on the HardenedBSD operating system and the `pf` packet filtering framework. Firewall rules in OPNsense are centrally managed through a configuration database and are persistently stored in an XML-based configuration file. This file is located in the filesystem at `/conf/config.xml` and contains all system settings, including interface definitions, firewall rules, network address translation (NAT) rules, and aliases.

Each firewall rule in OPNsense is represented as an XML element within the `<filter>` section of the configuration file. A rule specification typically includes attributes such as the rule action (e.g., pass or block), the network interface on which the rule is applied, the traffic direction, protocol, source and destination addresses, source and destination ports, and optional metadata such as a description and logging preference. The rules are ordered sequentially in the XML file, and this ordering directly reflects the evaluation order used by the packet filter.

```
<rule>
  <type>pass</type>
  <interface>lan</interface>
  <ipprotocol>inet</ipprotocol>
  <protocol>tcp</protocol>
  <source>
    <address>192.168.1.50</address>
  </source>
  <destination>
    <address>10.0.0.10</address>
    <port>443</port>
  </destination>
  <descr>Allow Workstation to Web Server</
    descr>
</rule>
```

Listing 2: XML representation of an OPNsense firewall rule with explicit IP addresses.

At runtime, OPNsense does not directly enforce rules from the XML configuration file. Instead, the system translates the XML-based rule definitions into `pf` rules, which are then loaded into the kernel-level packet filter. Firewall logs generated by OPNsense reference the internal rule identifiers derived from this translation process, allowing administrators to trace observed traffic decisions back to the corresponding high-level firewall rule definitions stored in the configuration file.

B. Reconstruction of Conflicting Rule Scenarios

Let $\{p_1, p_2, \ldots, p_n\}$ denote the finite set of network packets recorded in the firewall log, where each packet has been filtered by the firewall. Let $\{r_1, r_2, \ldots, r_m\}$ denote the ordered set of firewall rules configured on a given firewall interface. For each logged packet p_i, there exists a firewall rule r_k that was applied to p_i and produced the corresponding log entry.

To identify potential firewall rule conflicts, each packet p_i is evaluated against all other firewall rules r_j, where $j \neq k$, in order to determine whether those rules could also have matched the same packet. Any such rule is considered a potentially conflicting rule for packet p_i, as it would have been applied had rule r_k not been present or enabled.

Our proposed method reconstructs alternative rule-matching scenarios in which a network packet p_x could have been matched by firewall rules other than the rule r_a that was actually applied. Specifically, if rule r_a were absent or disabled, packet p_x might instead match one or more rules $r_b, r_c, \ldots, r_n$. We assume that rules $r_b, r_c, \ldots, r_n$ are ordered sequentially in the firewall configuration, but that rule r_a is positioned before them and therefore takes precedence during rule evaluation.

Intuitively, the provenance graph constructed by our approach captures this conflicting rule scenario for all traffic packets. It not only records that rule r_a was applied to packet p_x, but also reveals that rules $r_b, r_c, \ldots, r_n$ could have been applied under different rule ordering or configuration conditions.

A sample provenance graph that `Wall-PROV` generates is given in figure 4. This illustrates the provenance graph constructed from a single entry from OPNSense firewall log record. The graph follows the W3C PROV data model and represents firewall rules, traffic packets, and rule evaluation steps as agents, entities, and activities, respectively. `Wall-PROV` implements the provenance model as given in Table II.

In the figure, the firewall rule (`FirewallRule10`) and the input traffic packet (`IP_Packet`) are modeled as provenance agents. The packet–rule evaluation process is represented by the `Rule_Matching` activity, which captures the action of matching packet attributes against firewall rule conditions. The activity is associated with both agents, indicating that it is performed by the firewall rule and operates on the incoming packet.

Packet and rule attributes, such as source and destination IP addresses, source and destination ports, and protocol fields, are modeled as provenance entities. These entities are linked to the `Rule_Matching` activity through `used` relationships, representing the fact that the matching process consumes these attribute values during evaluation. Attribute entities are further connected to their respective agents via `wasAttributedTo` relations, indicating ownership of the attributes by either the firewall rule or the traffic packet.

The outcome of the rule evaluation is captured by the `FilteredTraffic_TR` entity, which represents the filtered traffic result. This entity is generated by the `Rule_Matching` activity through a `wasGeneratedBy` relationship and records the final decision of the firewall, such as allowing or blocking the packet. Temporal information

TABLE II: Firewall Concepts and Provenance Mapping

Concept	**Description and Example**	**Prov Type**
Firewall rule	Governs whether a packet is allowed or blocked (e.g., Firewall-Rule_10)	Agent
Firewall rule attributes	Fields used for rule evaluation (e.g., source IP, port, protocol)	Entity
Rule matching	Firewall operation that evaluates rules and triggers decisions	Activity
Firewall action	Outcome of rule matching (e.g., allow, block, reject)	Entity
Network packet	IP packet transmitted from a source to a destination	Agent
Packet fields	Structured content of a packet (e.g., IP, port, protocol)	Entity

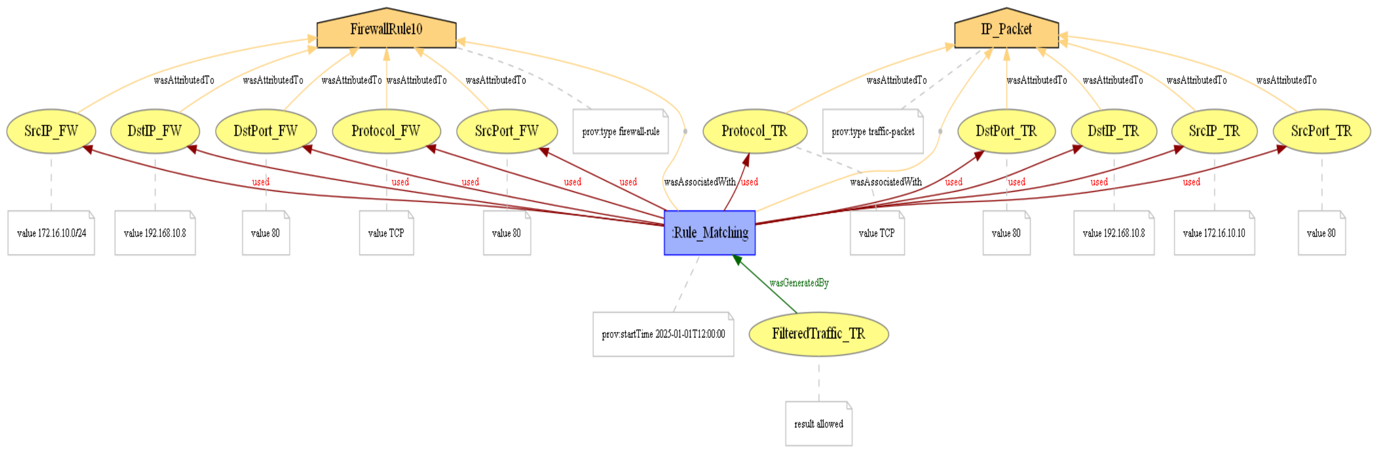

Fig. 4: Provenance graph illustrating firewall rule evaluation for a network packet

associated with the activity, including the evaluation start time, is also captured as provenance metadata.

Overall, the provenance graph provides a structured and auditable representation of firewall rule evaluation, explicitly capturing which rule was applied, which packet attributes were used, and how the final filtering decision was produced. This representation enables post hoc analysis of firewall behavior and serves as a foundation for detecting rule conflicts, misconfigurations, and alternative rule-matching scenarios.

In the best-case scenario, no rule misconfiguration exists, indicating that only rule r_a among all firewall rules matches packet p_x. In contrast, the worst-case scenario arises when every rule configured on the firewall interface could potentially match the same packet. This situation is theoretically possible, though very unlikely in reality.

C. Conflict Detection Algorithm

We illustrate a shadowing conflict using a concrete example based on the provenance entities shown in Fig. 4. In the provenance graph, `srcIP_FW` and `dstIP_FW` represent the source and destination IP address values specified in a firewall rule, while `srcIP_TR` and `dstIP_TR` denote the source and destination IP address values carried by a network traffic packet. All of these values are modeled as provenance entities and are consumed by the firewall's `Rule_Matching` activity during rule evaluation.

Consider two firewall rules configured on the same interface. The first rule, r_a, specifies a source IP range of `192.168.1.0/24` and a destination IP of `10.0.0.5`, and is configured to block matching traffic. The second rule, r_b, specifies a more specific source IP of `192.168.1.10` and the same destination IP `10.0.0.5`, and is configured to allow traffic. Although rule r_b is more specific, it is placed after rule r_a in the firewall rule set.

Now consider an incoming network packet with a source IP address of `192.168.1.10` and a destination IP address of `10.0.0.5`. The packet's source and destination IP values are represented in the provenance graph as the entities `srcIP_TR` and `dstIP_TR`. These values match the IP address entities `srcIP_FW` and `dstIP_FW` defined by both firewall rules r_a and r_b.

During firewall evaluation, the `Rule_Matching` activity associated with rule r_a is executed first due to the rule ordering. The matching process consumes the packet IP entities and the rule IP entities and generates a firewall action entity indicating that the packet is blocked. Because the firewall follows first-match semantics, evaluation terminates at this point, and the `Rule_Matching` activity for rule r_b is never executed.

From the perspective of the provenance graph, this execution results in a complete causal path from the packet IP entities to the `Rule_Matching` activity of rule r_a and subsequently to the generated blocking action. In contrast, although rule r_b defines IP address values that also match the packet, no provenance edges or action entities are created for this rule. As a result, rule r_b remains semantically applicable but operationally unreachable, which constitutes a shadowing conflict.

The following algorithm finds the firewall rule misconfiguration:

Algorithm 1: Reconstructing Firewall Rule Conflicts

```
for i = 1 to n do
  for j = 1 to m do
    if matches(p_i, r_j)
       and r_j != r_k then
      conflicting_rules <-
        conflicting_rules U {r_j}
```

A walk-through example is given as below explaining shadow conflicts with respect to figure 4.

IV. Wall-PROV Property Verification

In this section, we prove the properties of our provenance graph. In doing so, we first define the conflicts in F* formalism. Note that, we are not re-validating different type of firewall conflicts. Rather, we provide the formalisms as axioms as if they are facts. Later, we use these axioms to prove other properties including soundness and completeness of our tool.

A firewall rule is shadowed when an earlier rule matches the same traffic with a different action.

Axiom 2 (Shadowing Conflict)

```
assume shadowing_axiom :
  forall r1 r2 p.
    r1.order < r2.order /\
    matches p r1 /\
    matches p r2 /\
    r1.act <> r2.act
    ==> Shadowing r1 r2 p
```

A rule is redundant if it matches the same packets as a prior rule and has the same action.

Axiom 3 (Redundancy Conflict)

```
assume redundancy_axiom :
  forall r1 r2 p.
    r1.order < r2.order /\
    matches p r1 /\
    matches p r2 /\
    r1.act = r2.act
    ==> Redundant r2 r1
```

A later rule is more general than an earlier one but has a different action.

Axiom 4 (Generalization Conflict)

```
assume generalization_axiom :
  forall r1 r2 p.
    r1.order < r2.order /\
    matches p r1 /\
    matches p r2 /\
    subset r1 r2 /\
    r1.act <> r2.act
    ==> Generalization r2 r1
```

A later rule is more specific than an earlier rule and has a different action.

Axiom 5 (Specialization Conflict)

```
assume specialization_axiom :
  forall r1 r2 p.
    r1.order < r2.order /\
    matches p r1 /\
    matches p r2 /\
    subset r2 r1 /\
    r1.act <> r2.act
    ==> Specialization r2 r1
```

Two rules overlap partially and have different actions.

Axiom 6 (Correlation Conflict)

```
assume correlation_axiom :
  forall r1 r2 p.
    r1.order < r2.order /\
    overlaps r1 r2 /\
    matches p r1 /\
    matches p r2 /\
    r1.act <> r2.act
    ==> Correlated r1 r2
```

Lemma 1 (Soundness): A Firewall Rule Misconfiguration found by `Wall-PROV`, must be a true Misconfiguration
This is the soundness lemma of the data provenance in detecting firewall rule misconfigurations. With F*, we can formalize misconfiguration as follows.

```
assume Misconfiguration :
  rule -> rule -> packet -> Prop

assume misconfiguration_def :
  forall r1 r2 p.
    Misconfiguration r1 r2 p
    <==>
      Shadowing r1 r2 p
   \/ Redundancy r1 r2 p
   \/ Generalization r1 r2 p
   \/ Specialization r1 r2 p
   \/ Correlation r1 r2 p
   \/ Irrelevance r2
```

The Lemma 1 suggests that if any of the 5 firewall rule conflicts occur, we consider that incident as a misconfiguration. Proving that we use the Axiom 2-6 which define 5 firewall rule conflicts. These axioms have already been well accepted by research community. So, the right hand side of the Lemma is true, which proves the Lemma.

While detecting firewall rule conflicts with data provenance, we store the network packets information (e.g., ip address, port, protocol) in designated entities from the log file. On the other hand, a firewall rule's fields like ip address, port, protocol are stored in entities as well. An example has been given in Figure 4. Then, we followed the algorithms of axioms 2-5 to compare those entities to detect rule conflicts. Therefore, we are only detecting right conflicts.

Lemma 2: If a packet p appears in the firewall log, then at least one firewall rule must have applied to p

```
 lemma packet_in_log_means_rule_applied :
  forall p.
    in_log p
    ==> has_applied_rule p
```

This Lemma uses Axiom 1 and Axiom 7. Firewall logs contain the trace of a packet that is processed by at least one rule. In other words, no packet can pass a firewall without being recorded in the log.

Lemma 3 (Completeness): If there is a firewall rule conflict, `Wall-PROV` will definitely find it

As a provenance graph grows, it may become extremely large and contain many nodes. A query to detect a firewall rule conflict may not reach the appropriate nodes if the graph traversal gets trapped in a loop anywhere in the graph. One fundamental reason a query may not return an answer is the presence of a cycle in the provenance graph, from which graph traversal cannot escape. Therefore, proving Lemma 3 is equivalent to proving another lemma which refers to that the provenance graph generated by `Wall-PROV` has no cycle.

We formalize the provenance graph following the terminology used in our provenance model. We model provenance *agents* (firewall rule and network packet), *activities* (rule matching), and *entities* (rule attributes, packet fields, and filtering outcomes).

```
module FirewallProvAcyclic

type agent =
  | FirewallRule : string -> agent
  | IP_Packet    : string -> agent

type activity =
  | Rule_Matching : string -> activity

type entity =
  | RuleAttribute   : string -> entity
  | PacketField     : string -> entity
  | FirewallAction  : string -> entity
  | FilteredTraffic : string -> entity

type node =
  | Ag : agent -> node
  | Ac : activity -> node
  | En : entity -> node
```

Next, we declare the core provenance relations used in our graph. These relations correspond to the edges shown in the provenance diagram as Figure 4

```
val used :
  activity -> entity -> Tot bool

val wasGeneratedBy :
  entity -> activity -> Tot bool

val wasAssociatedWith :
  activity -> agent -> Tot bool

val wasAttributedTo :
  entity -> agent -> Tot bool
```

To reason about cycles, we define a directed edge predicate that represents the causal orientation of provenance dependencies.

```
let Edge (u v:node) : Tot bool =
  (exists a.
     u == Ag a /\
     v == Ac (Rule_Matching "")) \/
  (exists e.
     u == En e /\
     v == Ac (Rule_Matching "")) \/
  (exists e.
     u == Ac (Rule_Matching "") /\
     v == En e) \/
  (exists a e.
     u == Ag a /\
     v == En e)
```

We then introduce a simple rank function that assigns each node a level consistent with the expected causality of firewall provenance. Agents and input entities have the lowest rank, the rule matching activity has a higher rank, and output entities have the highest rank.

```
let rank (n:node) : Tot nat =
  match n with
  | Ag _ -> 0
  | En (RuleAttribute _) -> 0
  | En (PacketField _)   -> 0
  | Ac _ -> 1
  | En _ -> 2
```

The key definition for acyclicity is that every directed provenance edge strictly increases rank. This captures the notion that provenance edges represent causal dependencies and therefore must move forward in the rank ordering.

```
assume Edge_increases_rank :
  forall u v. Edge u v ==> rank u < rank v
```

Finally, we define paths and cycles in the directed graph. A cycle exists if there is a non-trivial path from a node back to itself. We state the main lemma that such cycles are not possible under the rank-increase definition.

```
type path : node -> node -> Type =
  | PNil  : forall u.
      path u u
  | PCons : forall u v w.
      Edge u v ->
      path v w ->
      path u w

let cycle (u:node) : Tot bool =
  exists v.
    Edge u v /\
    (exists pf. path v u)

lemma no_cycle :
  forall u.
    not (cycle u)
```

V. Evaluation

In this section, we evaluate `Wall-PROV` in five metrics (1) Effectiveness of firewall misconfiguration reconstruction; (2) Runtime overhead; (3) Storage overhead; and (4) Query performance.

A. Effectiveness

To evaluate whether `Wall-PROV` can capture all the firewall rule misconfigurations, we created some rules intentionally that are responsible for rule conflicts of all types. We use

TABLE III: Effectiveness of `Wall-PROV` in detecting firewall rule misconfigurations

Type	GT	F*	WP	Cov. (%)
Shadowing	12	12	12	100
Redundancy	14	14	14	100
Generalization	12	12	12	100
Specialization	8	8	8	100
Correlation	10	10	10	100

these as our ground truth. We use two Kali Linux tools *hping3* and *trafgen* to generate various types of ICMP, UDP, and TCP traffic that were sent across the LAN network. OPNSense listens to all traffic on the LAN interface.

hping3 can send traffic by spoofing the source IP address. This is commonly used for firewall testing, simulating DDoS attacks, or performing "idle scans" where you hide your real identity. On the other hand, *Trafgen* can spoof source IP addresses, but unlike hping3, it does not use simple command-line flags like –spoof. Instead, you must manually define the spoofed address within a packet configuration file or at the command line using its low-level configuration language.

We created approximately 40 firewall rules on the OPNsense LAN interface. Five virtual machines were connected to this interface, each assigned an IP address from the same address range. The firewall rules were intentionally configured to include multiple misconfigurations. As shown in the *Ground Truth* column of Table III, these misconfigurations were identified in advance.

In this evaluation, we examine whether `Wall-PROV` can detect the predefined misconfigurations. To validate the existence of these conflicts, we additionally apply a formal method–based analysis using F*. Both `Wall-PROV` and the formal method successfully detect all misconfigurations defined in the ground truth. These results demonstrate the completeness of `Wall-PROV` with respect to the firewall rule misconfigurations.

B. Storage Overhead

Figure 6 illustrates the run-time overhead incurred as the cumulative conflict count increases for different conflict ordering sequences. We start only shadowing conflicts at first Then we devise algorithom to detect redundancy conflicts. This is how the cumulative effect of two conflicts on run-time is observed. The x-axis represents the cumulative conflict count, while the y-axis shows the corresponding analysis overhead in milliseconds. Although the run-time overhead increases monotonically for all sequences, the rate of increase varies depending on the ordering of conflict types. Sequences that prioritize specialization or correlation earlier tend to cause lower overhead at the beginning, but later stages we see shadowing and redundancy conflict add extra overhead. The results indicate that ordering of conflict detection algorithm has some affects, though the final run-time with all type of conflict detection remains within a comparable range across all our experimental setup.

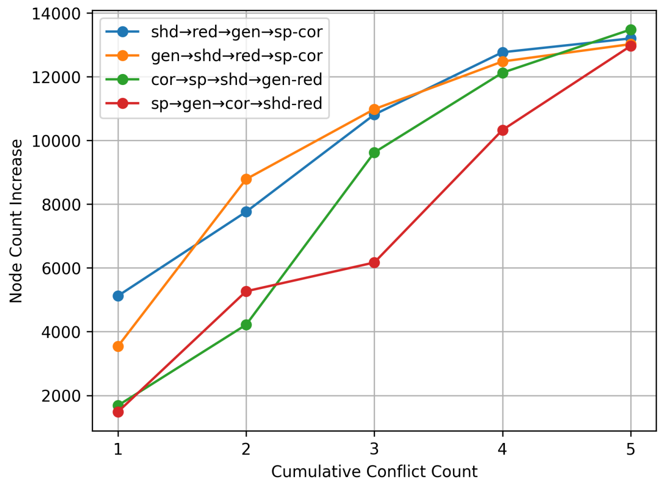

Fig. 5: Node count increase as a function of cumulative conflict count under different firewall rule conflict ordering sequences.

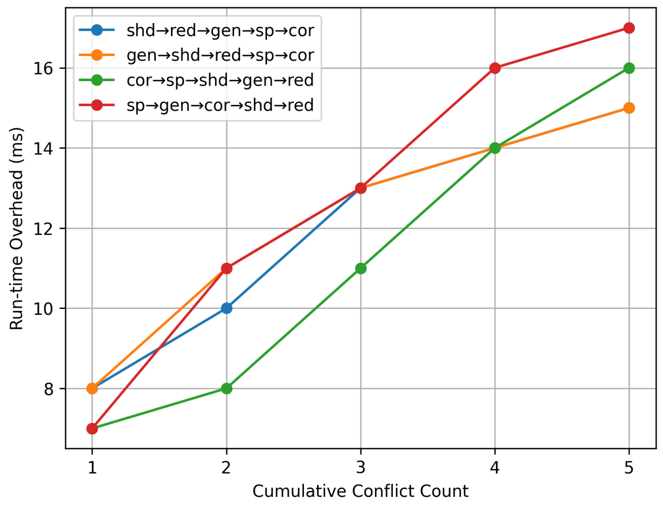

Fig. 6: Run-time overhead as a function of cumulative conflict count under different firewall rule conflict ordering sequences

C. Query Performance

Fig. 7 presents the query time for each type of firewall rule conflict. As can be seen, shadowing and redundancy conflict types exhibit the lowest detection times, at 37 ms and 34 ms respectively. It is explainable, as shadowing and redundancy primarily rely on IP address matching and do not require port and protocol analysis. The query whether or not shadowing or redundancy conflicts are there, do not need to traverse through port and protocol entities of firewall rules and log records.

In contrast, generalization, specialization, and correlation require traversing through the entities of source and destination IP addresses, ports, and protocol. This additional node traversing and analysis is reflected in their higher detection times.

Overall, the results indicate that detection time scales with the number of entity nodes needed to traverse. Nevertheless, all

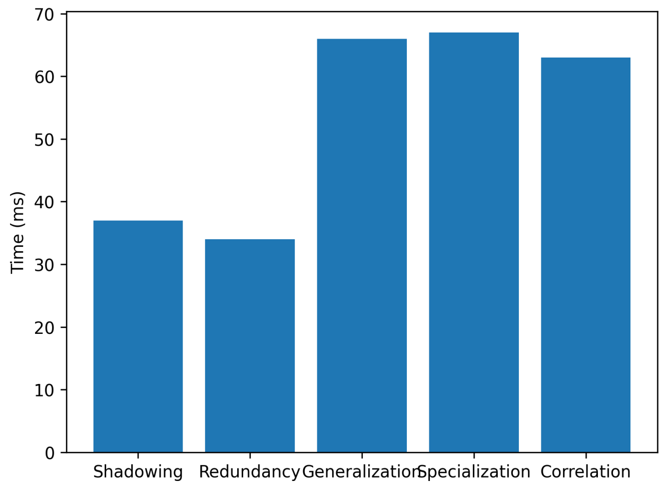

Fig. 7: Query time for different firewall rule conflicts

conflict types are detected within a small time window, demonstrating that `Wall-PROV` has potential for further analysis in manageable time frame.

VI. Related Works

Firewalls remain a common mechanism for enforcing network access control policies by filtering packets according to an ordered list of rules [4], [5]. In many widely deployed systems, rule evaluation follows first-match semantics, where a packet is compared against rules sequentially until the first match determines the action. While conceptually simple, this evaluation model can incur non-trivial latency as rule sets grow, and it makes performance highly sensitive to rule ordering and rule-set structure.

A substantial body of work has studied packet classification, which is closely related to firewall matching because rules typically constrain multiple header fields (e.g., source/destination IP prefixes, transport ports, and protocol). Classic results and systems work include multi-field classification formulations [12], decision-tree approaches such as HiCuts and HyperCuts [13], [14], bit-vector and scalable classification methods [15], [16], and tuple-space search techniques [17]. Benchmarking methodology and synthetic rule-set generation have also been developed to support fair evaluation of classification and filtering approaches [18]. More recently, researchers have explored learned or data-driven methods for packet classification under modern workloads [19].

In parallel, firewall policy management has been recognized as error-prone and costly due to rule interactions, shadowing, redundancy, and other anomalies. A large line of research investigates detecting and diagnosing policy anomalies and misconfigurations [2], [3], representing rule sets in forms amenable to analysis (e.g., BDD-based representations) [20], and designing structured representations that improve correctness and maintainability [21]. Formal verification and semantics-preserving simplification techniques further support auditing and reasoning about real-world rule sets [9], [22]. Related work also studies conflict detection and data-structure techniques for rule-set management and filtering [23].

Rule ordering is particularly important for list-based firewalls because it directly affects the average number of comparisons per packet. Prior work has proposed optimization strategies for firewall rule sets that aim to improve evaluation efficiency while preserving policy semantics, including techniques for analyzing rule interactions and optimizing rule-list structure [20], [24]. Additional systems research considers accelerating list-based classifiers by improving cache behavior, prefetching, or skipping mechanisms [24]. Recent surveys synthesize trends in automation for network security configuration and rule management [25], [26].

Within this context, the FROG approach proposes a traffic-independent strategy for reordering firewall rules while respecting policy constraints, targeting firewalls that can benefit from scanning optimizations such as skipping groups of rules during evaluation [27]. Such work complements correctness-oriented analysis (e.g., anomaly detection and verification) by addressing the efficiency bottleneck of rule evaluation in large deployments.

VII. Conclusion

This paper presented a provenance-based approach for detecting firewall rule misconfigurations. By modeling firewall rules, packet attributes, and rule evaluation as agents, entities, and activities, we constructed provenance graphs that capture the causal relationships between network traffic and firewall decisions. Unlike traditional static analysis techniques to solve the same problem, our method leverages firewall logs to reconstruct alternative rule-matching scenarios, enabling the identification of conflicts that are not directly observable from firewall logs alone. As Security auditing heavily relies on log records, our approach contributes significantly by leveraging from the firewall log records.

Through the provenance graph, common firewall misconfigurations such as shadowing, redundancy, generalization, specialization, and correlation can be systematically detected by reconstructing missing or alternative paths. We further provided a formal specification in F* to define conflict conditions and prove key properties of the provenance graph, providing assurance of soundness and completeness.

Our experimental evaluation on an OPNsense firewall with misconfigured rules showed that the proposed approach accurately detects all conflicts, achieving full coverage with negligible runtime and storage overhead. These results indicate that data provenance offers a practical and explanatory foundation for firewall misconfiguration analysis. Future work will explore a provenance-based analysis to rewrite firewall rules based on the misconfigurations `Wall-PROV` provides.

References

[1] K. Scarfone and P. Hoffman, “Guidelines on firewalls and firewall policy, nist special publication 800-41 revision 1,” National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2009. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-41r1

[2] E. S. Al-Shaer and H. Hamed, “Firewall policy advisor for anomaly discovery and rule editing,” in *Proceedings of the IEEE/IFIP Integrated Management Conference*, 2003.

[3] E. Al-Shaer and H. Hamed, “Discovery of policy anomalies in distributed firewalls,” in *Proceedings of the IEEE INFOCOM Conference*, 2005.

[4] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker*, 2nd ed. Addison-Wesley, 2003.

[5] K. Scarfone and P. Hoffman, “Guidelines on firewalls and firewall policy,” National Institute of Standards and Technology, Tech. Rep. SP 800-41 Revision 1, 2009.

[6] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2021.

[7] E. S. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005.

[8] M. Alicea and I. Alsmadi, “Misconfiguration in firewalls and network access controls: Literature review,” *Future Internet*, vol. 13, no. 11, p. 283, 2021.

[9] A. X. Liu, “Firewall policy verification and troubleshooting,” *Computer Networks*, vol. 53, no. 16, pp. 2800–2809, 2009.

[10] World Wide Web Consortium, “PROV-Overview: An Overview of the PROV Family of Documents,” https://www.w3.org/TR/prov-overview/, 2013.

[11] H. E. Sorotos, “Why provenance is the key to ai success: Knowledge graph ontology design,” https://odsc.medium.com/why-provenance-is-the-key-to-ai-success-knowledge-graph-ontology-design-8baa83554ccb, 2022, medium article.

[12] P. Gupta and N. McKeown, “Packet classification on multiple fields,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 924–934, 1999.

[13] ——, “Packet classification using hierarchical intelligent cuttings,” in *Proceedings of the IEEE Workshop on High Performance Switching and Routing*, 1999.

[14] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet classification using multidimensional cutting,” in *Proceedings of the ACM SIGCOMM Conference*, 2003.

[15] F. Baboescu and G. Varghese, “Scalable packet classification,” in *Proceedings of the ACM SIGCOMM Conference*, 2001.

[16] ——, “Scalable packet classification,” *IEEE/ACM Transactions on Networking*, 2005.

[17] V. Srinivasan, S. Suri, and G. Varghese, “Packet classification using tuple space search,” in *Proceedings of the ACM SIGCOMM Conference*, 1999.

[18] D. E. Taylor and J. S. Turner, “Classbench: A packet classification benchmark,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499–511, 2007.

[19] E. Liang and P. Bailis, “Neural packet classification,” in *Proceedings of the ACM SIGCOMM Conference*, 2019.

[20] S. Hazelhurst, A. Attar, and R. Sinnappan, “Algorithms for improving the dependability of firewall and filter rule lists,” in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, 2000.

[21] M. G. Gouda and A. X. Liu, “Structured firewall design,” *Computer Networks*, 2004.

[22] C. Diekmann, L. Hupel, and G. Carle, “Semantics-preserving simplification of real-world firewall rule sets,” *arXiv preprint arXiv:1604.00206*, 2016.

[23] D. Eppstein and S. Muthukrishnan, “Internet packet filter management and rectangle geometry,” in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[24] S. Hager, A. Kaufmann, and S. Müller, “Accelerating list-based packet classification through caching and prefetching,” in *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies*, 2014.

[25] A. Behl and S. Bagchi, “A survey of network security policy management and automation,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1124–1152, 2021.

[26] I. Kovačević, B. Štengl, and S. Groš, “A systematic review of automatic translation of high-level security policies into firewall rules,” *ACM Computing Surveys*, vol. 55, no. 3, pp. 1–36, 2022.

[27] A. Coscia, A. Maci, and N. Tamma, “Frog: A firewall rule order generator for faster packet filtering,” *Computer Networks*, vol. 257, p. 110962, 2025.