Towards Effective Integrated Access Control Lists in Internet Networks

Kamal A. Ahmat Ammar Elnour
Department of Information Technology
City University of New York
New York, NY 11101
kamal.ahmat@live.lagcc.cuny.edu ammar@it.sydney.edu.au

Abstract
Access Control Lists (ACLs) represent a traditional way in filtering packets in routers. In modern complex enterprise networks that provide a vast array of services, there is an ever increasing need for verifying the integrity of ACLs to detect any potential security holes and improve the network performance. This paper concerns the integrity of routers’ ACLs in large enterprise networks. We first investigate the integrity of the ACLs of two routers by describing a bottom-up approach for detecting redundancies in ACLs of two routers. We then extend our study to multiple routers and provide a heuristic algorithm for detecting redundant ACLs in multiple routers. We validate the practicality of our algorithm through real-life and synthetic router ACL groups of large networks. Performance results show that our heuristic algorithm don’t only improve the performance by reducing the number of comparisons overhead, but also helps in discovering potential security holes that cannot be discovered by considering the ACLs of each router individually.

I. Introduction
In today’s complex high performance internetworks, there is increasing need to implement packet forwarding and routing efficiently and effectively by organizations according to their own defined policies. Such implementations do not only need to be in a way that goes beyond traditional issues related to routing protocols, but should also, most importantly, address the security concerns of these organizations. Even though firewalls serve as the first line of defense against cyber attacks, relying solely on the protection provided by firewalls is not sufficient to secure networks’ infrastructures. Consequently, many router products allow a firewall capability, such as Access Control Lists (ACLs) to filter packets and provide additional protection line within the network infrastructure.

The ACLs are mainly created for each router on an individual basis and stored on a file. Then, network administrators usually use TFTP servers to grab the pre-configured ACL file and perhaps modify it to configure each router’s security policy. The ACLs on these policy files are often have different moment, some grant the access for certain applications, others for IPs and protocols, third ones make accessible the certain interfaces, etc. Furthermore, for large-scale complex networks, these files can be modified by multiple administrators over an extended period of time, which makes the ACLs prone to conflicts, redundancy and security holes. The situation even becomes more complex and critical if the integrity of ACLs of all routers in the network infrastructure, which is often ignored, is considered.

Once written, the ACL is experiencing a constant modification, which becomes painstaking with the course of time. The number of ACLs in a group increases constantly which doesn’t only raise serious scalability and performance issues, but also highly increases the probability of anomalies among these ACLs, which in turn raises serious security concerns.

The anomalies have double effect: on the one hand, they can compromise organization for letting confidential information out, on the other hand, they expose the network to external attacks, the consequences of which are hard to predict [1]. Despite the importance of automating the process of discovering ACL integrity at the network level, we are not aware of any commercial platform available on the market today that offer tools for analyzing the integrity or discovering and eliminating anomalies in routers’ ACLs at the network level. The situation becomes even more complex with the presence of extensions to the standard ACLs such as Dynamic, Reflexive, and Time-based ACLs.

Even though firewall policy rules and routing ACL groups may look similar, there are several fundamental differences between them:
Firewall policies filter the network inbound/oubound traffic, while router’s ACLs control all types of traffic including the traffic that is originated from within the network. Thus, routers are used to secure the network infrastructure against insider threats. While firewalls can be also installed within the network, their effect remains limited comparing to the routers.

2) A typical firewall has one incoming and one outgoing interfaces, and the firewall decides to either accept or deny the traffic. However, routers have multiple interfaces and the traffic control can vary based on the source and destination interfaces. Thus, routers are generalization of firewalls.

3) Firewalls don’t deploy any other strategies besides policies to filter the traffic, while routers take the decision based on routing strategies (i.e. protocols) which coordinate the communication between routers.

4) The vast proliferation of virtualization, and consequently virtual networks, resulted in unique set of open issues that are distinctly related to routing and associated policies. For instance, the impact of features such as Router Grafting [8] on the security of network infrastructures remains vague.

This paper represents an initial exploration step in studying this unknown field, posing more questions than it answers. We hope presenting this paper will draw the attention from the research community on this challenging and important problem. In this paper, we address the problems related to the integrity of routers’ ACLs in the network’s infrastructure. First, we describe a bottom-up method for verifying the integrity of two router’s ACL groups and find any potential redundant ACLs. Then, we generalize our approach to multiple routers. After that, we validate the practicality of our algorithms by conducting experiments on multiple large network infrastructures as well as on synthetically generated data. Our results demonstrate that our approach does not only find and eliminate redundant ACLs, but also improves the security level of the network by discovering potential security holes that cannot be discovered by considering the ACLs of each router individually.

A. Organization of the Paper

The rest of the paper is organized as follows. The next section describes the system model used in this paper and problem formulation. We subdivide Section III into two subsections. First, in Subsection III-A we present an algorithm to find the set of redundant ACLs in a pair of ACL groups. Second, in Subsection III-B we describe an algorithm for discovering redundant ACLs in multiple groups. Our environment setup and experimental results are described in Section IV. Finally, Section V concludes the paper.

II. System Model and Problem Formulation

In this section, we describe the model we adopted and present formal description of the problem addressed in this paper.

A. Routers and Access Control Lists

A router is a network device that is designed to control data traffic between different computer networks based on what is known as a routing table. It reads every data packet that it receives from a network and directs it through the right connection, so that it reaches the right destination based on what is known as Routing Table[15]. The routing table stored on a router has information about the IP addresses of all computers in a network. It also has a set of priority rules for data transport, which the router follows. Furthermore, routers can filter network traffic by deciding whether packets will be accepted or denied based on what is known as the Access Control List (ACL). An ACL is typically assigned to a specific router interface and it defines the parameters against which each connection is compared, resulting in a decision on what action to take. As soon as a network packet matches an ACL, the corresponding action is applied, and processing stops. If the action is acceptance, then the packet is forwarded to the appropriate destination based on the routing table, otherwise, it is dropped. If no matches are found when the router reaches the end of the list, the traffic is denied. For this reason, each ACL group should have the frequently hit entries at the top of the list. Also, there is an implied deny for traffic that is not permitted. A single-entry ACL with only one deny entry has the effect of denying all traffic. Consequently, there must be at least one permit statement in an ACL group or all traffic is blocked.

The standard ACL is typically a five-tuple filter that has the following format:
(order, src_ip, src_mask, dst_ip, protocol, action)

The first attribute is the order assigned to the ACL and represents the relative order of the ACL within the given ACL group. To improve the performance and scalability of routers, the ACLs that have higher hit probability are typically listed on the top of ACL group so that the number of comparisons is kept minimum.

The remaining attributes represent the source IP address, source IP mask, destination IP address, protocol, and action, respectively. Additional parameters such as the port number, timeout, traffic type (e.g. in or out) are also often used. However, in this paper, we assume the most basic model consisting of the six attributes described above.

To be able to build a useful model for filtering ACLs, we need to determine all the relations that may relate two or more packet filters. In this section we state all the possible relations that may exist between filtering rules. Since router ACLs can be looked at as a superset of firewall filtering policies, all types of relationships between firewall rules are also inherited in a router ACLs. Any two firewall rules can be either (1) disjoint; (2) exactly matched; (3) inclusively matched; (4) partially disjoint; or (5) correlated [3]. Besides the relationships listed above, we also define the sibling relationship for two ACLs on the same router as follows.

**Definition 2.1:** [SIBLING-ACLs]
Two ACLs $a_1$ and $a_2$ are siblings if and only if the following conditions are satisfied:

1) $a_1$ and $a_2$ are completely disjoint; and
2) If $a_1$ succeeds $a_3$ in the ACL group, denoted $a_1 > a_3$, then $a_2 > a_3$.

We use the notion $\prec$ as a function that lists the relationships between two ACLs in a given group $i$. For example, in the ACL group described in Table 1, $a_5 \prec_1 a_6$ means $a_5$ and $a_6$ are siblings since they both are disjoint and that $a_5 > a_3$ and $a_6 > a_3$, while $a_6 \prec_1 a_3$ means $a_6 > a_3$.

Since router ACLs groups inherit the relationship types from the firewall rules, the anomalies are also implicitly inherited. These anomalies include shadowing, correlation and redundancy. Throughout this paper, we assume that the firewall policy anomalies are not present. That is, each ACL group is optimally configured, and we focus on the optimization of ACL groups integrity. This assumption is practical as the research community has addressed practical methods to resolve them [3], [4], [6], [7], [10]. However, the model for router groups can be much more complex. For example, both ACL groups for Router A and Router B contain the ACL $a_4$ which may seem as a redundancy anomaly. But by looking carefully at both routers’ ACL groups it can be observed that a data packet can be treated differently by each router’s ACL group. For example, an IP packet that is originated at IP address 10.1.1.1 will be denied by the group 1 while the same packet will be accepted by the group 2. Thus, the comparison at the ACL level may not give an accurate overview about the anomalies in ACL groups at the network level.

In our model, we state that an ACL is redundant among multiple routers’ ACL groups if and only if all packets are treated similarly by a specific ACL that belongs to all these groups. For instance, ACLs $a_7$ and $a_8$ are redundant in both group 1 and group 2, while, as stated earlier, ACL $a_4$ is not redundant.

In today’s large enterprise networks, the network may comprise tens or even hundreds of routers. In such networks, it is important to look at the global view rather than considering the ACL of each router individually. Furthermore, the situation becomes much more complex when taking into account practical considerations such as the impact resulted in the interaction of the ACLs groups with Border Gateway Protocol (BGP) and other protocols. One major anomaly in distributed ACL
groups is ACL redundancy. Since redundant ACLs may affect the performance of the network infrastructure security management, no redundant rules should be present in routers’ ACL groups in an optimal router group configuration. Consequently, we define a simple optimal configuration as configurations of routers ACLs such that no redundant ACL is present in the given configuration. We use the notion simple optimal configuration to distinguish the problem addressed in this paper from more optimally sophisticated configurations that take other factors into account.

To this end we define a notion of the deletion-safe ACL as the ACL that its deletion from a group doesn’t result in accepting traffic that would have been denied otherwise; or denying traffic that would have been accepted otherwise. For example, the ACL \( a_8 \) is deletion-safe since removing it will not affect the router’s actions while the ACL \( a_1 \) in Table 1 is not deletion-safe since all packets originated at IP address 10.1.1.1 will be accepted otherwise. It is a best practice to deny the packets at the upstream router (see Fig 1) rather than allowing the packet to enter the network and then deleting it at the downstream router. Consequently, the priority is given to eliminating deletion-safe ACLs from upstream routers.

### B. Problem Formulation

Given a collection of ACL group configurations each associated to a router in the network, our goal is to discover and remove all deletion-safe redundant ACLs from the given set of configurations. Formally, the simple optimal configuration problem is defined as follows.

**Definition 2.2:** [SIMPLE-OPTIMAL-CONFIGURATION]

Given a collection \( \mathcal{G} = \{G_1, G_2, \ldots, G_n\} \) of ACL group configurations, construct another collection \( \hat{\mathcal{G}} \subseteq \mathcal{G} \) such that \( \hat{\mathcal{G}} \) doesn’t contain any redundant ACLs and the both collections have identical output on any given input set. □

### III. Redundancy Detection Algorithm

In this section we first describe an algorithm for discovering and eliminating ACL redundancies in two router configurations. Then, we extend our algorithm to multiple routers. After that, we analyze the complexity of our algorithms and provide an example.

#### A. Discovering Redundant ACLs in Two ACL Groups

The Discover-Redundant-ACLs procedure receives as an input two groups of router ACLs. It then eliminates from each group the ACLs that are not present in the other group. At this stage, both ACL groups contain the same set of ACLs but these ACLs could have different relations. The algorithm then finds the relation between each pair of ACLs from the first group and checks whether the same relationship is maintained between this pair in the second group eliminating all ACLs that have different relationship from groups. The algorithm then returns the set \( R \) of redundant ACLs. The formal description of the Discover-Redundant-ACLs procedure is given in Fig. 1.

To derive the time complexity of the algorithm we observe that the first stage of the algorithm, which finds similar ACLs in both configuration, compares each ACL from the first group to each ACL from the second group, resulting of an \( O(n^2) \) time complexity, where \( n \) is the number of ACLs in the smaller group. The second phase finds the type of relationships between each pair ACLs in the same group, then compares the results with the ones obtained from the other group. Consequently, we derive that the time complexity of the algorithm is \( O(n^2) \).

#### Algorithm Discover-Redundant-ACLs\((G_1, G_2)\)

**Input:** Two sets \( G_1 \) and \( G_2 \) of ACL groups

**Output:** Set \( R \) of redundant ACLs

1. set \( R \leftarrow \phi \);
2. for every \( a_i \in G_1 \)
3. 1. if \( a_i \notin G_2 \)
4. 2. set \( G_1 \leftarrow G_1 \setminus a_i \);
5. 3. set \( G_2 \leftarrow G_2 \setminus a_i \);

(* Compare relationship type in both groups *)

6. for every \( a_i \) and \( a_j \in G_1 \)
7. 1. if \( a_i \bowtie a_j = a_i \bowtie a_j \)
8. 2. if \( a_i \) is deletion safe
9. 3. set \( R \leftarrow R \cup a_i \);
10. 4. if \( a_j \) is deletion safe
11. 5. set \( R \leftarrow R \cup a_j \);
12. return \( R \);

---

**Table III**

<table>
<thead>
<tr>
<th>Id</th>
<th>Source IP</th>
<th>Mask</th>
<th>Dest. IP</th>
<th>Protocol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>192.168.0.0</td>
<td>0.0.255.255</td>
<td>10.0.0.1</td>
<td>IP</td>
<td>allow</td>
</tr>
<tr>
<td>3</td>
<td>10.1.1.0</td>
<td>0.0.255.255</td>
<td>10.0.0.1</td>
<td>TCP</td>
<td>allow</td>
</tr>
<tr>
<td>5</td>
<td>10.1.1.0</td>
<td>0.0.255.255</td>
<td>10.0.0.*</td>
<td>SMTP</td>
<td>allow</td>
</tr>
<tr>
<td>6</td>
<td>10.1.2.0</td>
<td>0.0.255.255</td>
<td>10.0.0.*</td>
<td>FTP</td>
<td>allow</td>
</tr>
<tr>
<td>7</td>
<td>192.168.0.0</td>
<td>0.0.255.255</td>
<td>10.0.0.1</td>
<td>TCP</td>
<td>allow</td>
</tr>
<tr>
<td>8</td>
<td>192.168.0.0</td>
<td>0.0.255.255</td>
<td>10.0.0.1</td>
<td>UDP</td>
<td>deny</td>
</tr>
</tbody>
</table>

---
Algorithm Network-Redundant-ACLs(G)

Input: A collection G of ACL group sets
Output: A collection G of redundant-free ACL group sets

1. find G_i, G_j s.t. G_i ∩ G_j is maximum;
2. set R(i,j) ← Discover-Redundant-ACLs(G_i, G_j);
3. if R_i is upstream router
   set G_j ← G_j \ R(i, j);
4. else set G_i ← G_i \ R(i, j);
5. return G;

Fig. 3. A formal description of the Network-Redundant-ACLs algorithm.

B. Discovering Redundant ACLs for Multiple ACL Groups

Unfortunately, generalizing the Discover-Redundant-ACLs algorithm to multiple groups will result in an algorithm that has the complexity of O(n^5), which is exponential. We suspect that finding redundant ACLs in multiple groups is an NP-hard problem. Consequently, we develop a heuristic for eliminating redundancies from ACL groups even though it doesn’t guarantee finding the optimal collection that doesn’t contain any ACL redundancy.

Our algorithm, which is termed Network-Redundant-ACLs, takes as an input a collection of groups such that each group consists of a list of ACLs. The algorithm follows the greedy approach and starts by finding two groups such that these groups share the maximum number of exact matching ACLs. Then, it applies the Discover-Redundant-ACLs procedure and finds and eliminates the redundant ACLs. These steps are repeated until all redundancies are eliminated. Observe that the result obtained from the Network-Redundant-ACLs algorithm may depend on the order in which the group pairs are processed. For instance, in a collection that contains three groups, processing G_1 and G_2 first may result in a different result than processing G_1 and G_3 first, which supports our hypothesis that the problem is NP-hard.

To derive the time complexity of the Network-Redundant-ACLs algorithm we observe that the Discover-Redundant-ACLs procedure is used to find redundant ACLs between each pair of configurations. The step is repeated k times where k is the number of groups (routers) in the network. Consequently, the total cost of removing all redundant ACLs from the network is O(kn^2), where k is the number of routers in the network and n is the number of ACLs in the smallest group. We strongly believe that more efficient methods for redundancy discovery and elimination can be developed.

Example 3.1: Consider the ACL groups depicted in Table I and Table 2. The algorithm will first eliminate from both G_1 and G_2 all ACLs that are not present in both groups which will result in G_1 = \{a_2, a_5, a_6, a_7, a_8\} and G_1 = \{a_2, a_3, a_5, a_6, a_7, a_8\}. We observe that a_4 = a_6 = a_8 = a_0 (a_1 and a_0 are siblings, and a_2 and a_3 are siblings). Furthermore, a_5 < a_1, a_2 < a_0, and a_3 < a_6. Consequently, all a_3, a_5, and a_6 will be added to R. After that, since a_3 = a_1 ≠ a_4 ≠ a_0, a_4 is not redundant and consequently, it will not be added to R. Also, a_2, a_7, and a_8 will be added to R. Thus, the set R(G_1, G_2) is the group of ACLs depicted in Table III.

IV. Implementation and Experimental Results

We implemented the algorithms described in this paper in the context of Verizon Internet Security Suite’s (Beta) system using platform-independent C++. In this section, we present our evaluation study of the scalability and performance of our method. To evaluate the performance of our algorithms, we conducted two sets of experiential tests. In the first set, we conducted further stress tests on synthetic routers’ configurations that have very large number of ACLs. Since the goal of this paper is to discover network-level ACL anomalies rather than router-level anomalies, the ACLs of each router were first ordered and conflicts were removed using the methods described in [3], [7].

To test the performance of our methods on real-life networks, we deployed our implementation component in six distinct networks that consisted of 11, 24, 52, 107, 131, and 158 routers, respectively. For the stress test, we ran six test sets on very large ACL groups. For each test set, we generated random network topologies using BRITE brite and then generated the ACLs of each router using a modified version of the method described in [13]. Then, we submitted the ACL group files to our component and calculated the average for each test set. The performance results are shown in Tables IV and V, respectively. One of our main observation is that network administrators have the tendency to often add, remove, and edit ACLs that control active traffic in the network, which implies that the majority of redundant ACLs among multiple routers are mostly related to the active traffic in the network. Thus, de-
tecting and removing these redundant ACLs result in considerable performance improvement. Obviously, the average comparison numbers depend on the traffic type and the complexity of ACLs which we didn’t consider at this stage. We observed that the performance ratio of our method is not affected by the increasing number of routers and the numbers of ACLs, which validates the high scalability and efficiency of our approaches.

In general, we have discovered that our algorithm is sufficiently fast and highly scalable for all practical purposes. As can be seen in our results, the performance improvement ratio was quite high and our method helped in discovering many redundancies as well as potential security holes that were undiscovered by network administrators in real-life configurations.

V. Conclusion

Can more sophisticated network-level techniques provide significant improvement on security and performance gains over the conventional device-level methods used currently in large networks? Even though the initial results we obtained are very encouraging, we believe more research needs to be conducted to find the precise answer to this question. This paper addresses a new unexplored direction in networking security by considering integrating security configurations for all routers in the network. We provide initial exploration of the problem and the motivations behind it. Then, we study the problem of discovering and eliminating redundant ACLs from multiple routers’ configurations and describe efficient methods for removing such redundancies. We have implemented our methods and validated its practicality on both real-life as well as synthetically generated access control lists on several large-size networks. The experimental results support our theoretical analysis and show that our proposed method improves the performance of firewall performance significantly and can also discover potential security holes in network infrastructures.

References


<table>
<thead>
<tr>
<th>Network No. of Routers</th>
<th>Avg. No. of ACLs</th>
<th>Avg. No. of Redundant ACLs</th>
<th>Imp. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>35</td>
<td>3.7</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>34</td>
<td>2.6</td>
</tr>
<tr>
<td>3</td>
<td>52</td>
<td>61</td>
<td>7.1</td>
</tr>
<tr>
<td>4</td>
<td>107</td>
<td>42</td>
<td>9.8</td>
</tr>
<tr>
<td>5</td>
<td>131</td>
<td>66</td>
<td>17.8</td>
</tr>
<tr>
<td>6</td>
<td>158</td>
<td>71</td>
<td>12.4</td>
</tr>
</tbody>
</table>

TABLE IV PERFORMANCE RESULTS FOR REAL-LIFE ACL GROUPS.

<table>
<thead>
<tr>
<th>Network No. of Routers</th>
<th>Avg. No. of ACLs</th>
<th>Avg. No. of Redundant ACLs</th>
<th>Imp. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>500</td>
<td>85.4</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>700</td>
<td>121.7</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>1000</td>
<td>64.2</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>1500</td>
<td>213.2</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>2000</td>
<td>262.1</td>
</tr>
<tr>
<td>6</td>
<td>2000</td>
<td>2500</td>
<td>394.5</td>
</tr>
</tbody>
</table>

TABLE V PERFORMANCE RESULTS FOR SYNTHETIC ACL GROUPS.