Firewall Rule Set Inconsistency Characterization by Clustering

Sergio Pozo, Rafael Ceballos and Rafael M. Gasca

Department of Computer Languages and Systems, ETS Ingeniería Informática
University of Seville, Avda. Reina Mercedes S/N, 41012 Sevilla, Spain

Abstract. Firewall ACLs could have inconsistencies, allowing traffic that should be denied or vice-versa. In this paper, we analyze the inconsistency characterization problem as a separate problem of the diagnosis one, and propose definitions to characterize one-to-many inconsistencies. We identify the combinatorial part of the problem that causes exponential complexity in combined diagnosis and characterization algorithms proposed by other researchers. The problem is divided in several smaller combinatorial ones, which effectively reduces its complexity. Finally, we propose a heuristic to solve the problem in worst case polynomial time as a proof of concept.

1 Introduction

A firewall is a network element that controls the traversal of packets across different network segments. It is a mechanism to enforce an Access Control Policy, represented as an Access Control List (ACL), or a *rule set.* An ACL is in general a list of linearly ordered (total order) condition/action rules. Let *RS* be a firewall rule set consisting of *n* rules, $RS = \{R_1, ...R_n\}$. Consider $R =< H, Action >, H \in \mathbb{N}^4$ as a rule, where $Action = \{allow, deny\}$ is its action. A selector of a firewall rule R_j is defined as $R_j[k], 1 \le k \le n, k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$. A rule matches a packet when the values of each field of the header of a packet are subsets or equal to the values of its corresponding rule selector.

Firewalls have to face many problems in modern networks. One of the most important ones is rule set consistency [6]. Selectors of rules can partially or totally overlap. There is an inconsistency when two or more rules with different actions overlap. An inconsistent firewall ACL implies a design error in general, and indicates that the firewall is accepting traffic that should be denied or vice-versa.

In this paper we analyze the inconsistency characterization problem in firewall rule sets, and extend the complete formal inconsistency characterization given by Al-Shaer et al. [3], resulting in a complete one-to-many characterization. Then, we identify the combinatorial part of the problem that causes the combinatorial explosion in combined diagnosis and characterization algorithms proposed by other researchers,

Pozo S., Ceballos R. and M. Gasca R. (2008).
Firewall Rule Set Inconsistency Characterization by Clustering.
In *Proceedings of the 6th International Workshop on Security in Information Systems*, pages 138-144
DOI: 10.5220/0001730701380144

and propose a decomposition of this combinatorial part in several smaller ones. The proposed characterization algorithms are based on a polynomial heuristic and on a previous diagnosis process that is worst case $O(n^2)$ time complexity [6]. In this paper, detection is understood as the action of finding the rules that are inconsistent with other rules; identification is the action of finding the rules that cause all the inconsistencies among the detected inconsistent rules (the faulty rules); and characterization is understood as the action of naming the identified inconsistent rules among a pre-established taxonomy of faults.

2 Related Works

Some other researchers have complemented the diagnosis process with a characterization of the faults with an established taxonomy [3]. One of the most important advances was made by Al-Shaer et al. [4], where authors define a complete inconsistency model for firewall rule sets. They give a combined algorithm to diagnose and characterize the inconsistencies between pairs of rules. They used rule decorrelation techniques [2] as a pre-process in order to decompose the rule set in a new, bigger, one with no overlapping rules. This new rule set is different from the initial one, and the user is the responsible of mapping the rules of this rule set to the original one once inconsistencies are given. This model can only diagnose and characterize inconsistencies between pairs of rules. Although the proposed characterization algorithm proposed by Al-Shaer is polynomial, the decorrelation pre-process imposes a worst case exponential time and space complexity for the full process. A modification to their algorithms was provided by García-Alfaro et al. [5], where they integrate the decorrelation and characterization algorithms of Al-Shaer, and generate a decorrelated and consistent rule set. Thus, due to the use of the same decorrelation techniques, this proposal also has worst case exponential complexity. However, García-Alfaro et al. provide a characterization technique with multiple rules. Ordered Binary Decision Diagrams (OBDDs) have been used in Fireman [7], where authors provide a diagnosis and characterization technique with multiple rules. However, the complexity of OBDD algorithms depends on the optimal ordering of its nodes, which is NP [6].

There are several important differences between our works and these ones. The combination of diagnosis and characterization in only one step results in exponential algorithms. However, only a part of the characterization problem is of exponential nature. In a previous work, we proposed to divide consistency management in two sequential processes [6]: detection and identification (diagnosis) of inconsistent rules, and characterization of the diagnosis. This analysis enabled us to identify and isolate the combinatorial part of it and improve the algorithmic complexity of the full process. A heuristic polynomial algorithm is provided as a proof of concept. Results are given over the original, unmodified, rule set.

3 Analysis of the Inconsistency Characterization Problem

Real life rule sets can be decomposed in two different subsets of rules. The first one is a set of consistent rules. The other one is formed by subsets of ICIRs (Definition 2).

Definition 3.1. Inconsistency between two rules $R_x, R_y \in RS$ **.**

$$Inconsistent(R_i, R_j, RS), 1 \le i, j \le n, i \ne j \Leftrightarrow R_x[k] \cap R_y[k] \ne \varnothing \wedge R_x[Action] \ne R_y[Action],$$
$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

Definition 3.2. ICIR. Let $CV = \{R_1, ...R_n\}$ be a set of rules, then

$$ICIR(root, CV) \Leftrightarrow \forall R_i \in CV \bullet Inconsistent(root, R_i) \wedge \forall R_i, R_j \in CV, i \ne j \bullet \neg Inconsistent(R_i, R_j)$$

Definition 3.3. Diagnosis Set, DS.

Let $ICIRS = \{ICIR_1, ..., ICIR_m\}$ be the set of all ICIR of a given RS, then

$$DS = \{ICIR_1(root), ..., ICIR_m(root)\}$$

3.1 Characterization Taxonomy of One to Many Inconsistencies

Our definitions extend Al-Shaer [3] and are capable of a one to many characterization. Let '<' and '>' be operators defined over the priority of the rules, where $R_x < R_y$ implies that then R_x has more priority than R_y and vice-versa. These definitions can be directly extended to support a cluster of rules with the same action in R_x or R_y (but not in both). If R_x is a cluster of rules and R_y is a rule, then R_y is shadowed by R_x, and vice-versa.

- **Shadow and Exact Shadow**

$$\exists R_x, R_y \in RS \bullet R_x > R_y \bullet Shadow(R_y) \Leftrightarrow$$
$$\forall k \bullet R_y[k] \subset R_x[k] \wedge R_x[Action] \ne R_y[Action]$$
$$k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$
Shadow

$$\exists R_x, R_y \in RS \bullet R_x > R_y \bullet ExactShadow(R_y) \Leftrightarrow$$
$$\forall k \bullet R_y[k] = R_x[k] \wedge R_x[Action] \ne R_y[Action]$$
$$k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$
Exact shadow

- **Generalization.** It is the inverse of shadow respect to the priority.

$$\exists R_x, R_y \in RS \bullet R_x > R_y \bullet Generalization(R_y) \Leftrightarrow \forall k \bullet R_y \supset R_x \wedge R_x[Action] \ne R_y[Action]$$
$$k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

- **Correlation**

$$\exists R_x, R_y \in RS \bullet Correlation(R_x R_y) \Leftrightarrow \forall k \bullet R_x[k] \cap R_y[k] \wedge R_x[Action] \ne R_y[Action] \wedge$$
$$\neg(R_x \subseteq R_y) \vee \neg(R_x \supset R_y), k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

4 Inconsistency Characterization Process

The characterization process explained in this section takes as input the inconsistency diagnosis as ICIRs [6]. At the first step, for each ICIR, children are joined in different clusters in order to abbreviate the returned characterization for that conflicting rule (ICIR root). These clusters are formed by rules that are subsets, supersets, equal, or form a continuous space for all selectors. The clusters represent the rules that share the same inconsistency with their ICIR root. Due to the priority dependency of characterization definitions, it is possible to divide ICIR children rules in two lists: rules that come before and rules that go after root. Then, clustering is done independently for each of these two lists. Division process is in *O(c)* with the number of children. In general, clustering is possible if all rule selectors permit multiple values, ranges and/or wildcards in their syntax. Fortunately, firewall languages support it in all selectors, but only continuous ranges [1]. This enables the clustering of rules for all selectors.

Algorithm 1. Initialization.

```
1  Func initialization(in Rule: root, List of Rule: children)
2  Alg
3     if root.DstPort().isRangeOrWildcard() AND
4        children.size()>1 {
5           sortAscendingByDestinationPort(children)
6           if root.Priority()>children.last().Priority() OR
7              root.Priority()<children.first().Priority() {
8                 clusterize(root, children)
9           }
10          else {
11             List before = Rules with priority > root
12             List after = Rules with priority < root
13             clusterize(root, before)
14             clusterize(root, after)
15          }
16    else {
17       doPairwiseCharacterization(root, children)
18    }
19 End Alg
20
21 Func doPairwiseCharacterization(in Rule: root, List of
22 Rule: children)
23 Alg
24    for each i=1..children.size()
25       doClassification(root, children.get(i))
26 End Alg
```

Algorithm 2. Cluster construction.

```
1  Func clusterize(in Rule: root, List of Rule: children)
2  Var
3     Rule cluster
4  Alg
5     cluster = children.first()
6     for each i=2..children.size() {
7        if isClusterizable(union, children.get(i) AND
8           i<children.size()) {
9              cluster.joinWith(children.get(i))
10       }
11       else if isClusterizable(union, children.get(i) AND
12             i==children.size()) {
13                cluster.joinWith(children.get(i))
14                doClassification(root, cluster)
15       }
16       else { // Not clusterizable
17          doClassification(root, cluster)
18          // re-initializes for a new cluster
19          cluster=children.get(i)
20       }
21    }
22 End Alg
23
24
25
26
```

Since ICIRs represent independent clusters of inconsistencies, they can also be characterized independently, effectively reducing the problem complexity: the combinatorial problem have been reduced from the entire rule set to several smaller ICIRs. However, there is still a trade off between optimally solving the problem in exponential time, or using an approximation to the optimum. In this paper we propose a worst case polynomial heuristic. The heuristic is used when clustering ICIR children. It only takes into account one selector for rule clustering, and does not try to check all possible unions between all selectors. For each ICIR, their children are clusterized in several groups by destination port forming a continuous range. This task can be done in linear time if children are ordered by destination port. The first cluster is formed with the first children. Then the next children should be added only if its destination port selector can form a continuous range with the cluster, and if the rest of selectors are equal, subset, superset or wildcard. If it cannot be joined, then the cluster is closed and a new one is formed with that child, and the process begins again until there are no more children.

The first part of the process checks ICIR root structure and prepare children for clustering (Algorithm 1). Then, it identifies the rules that can be joined with others in each ICIR (Algorithm 2). Algorithm 1 takes as input the ICIR root and children, and

first checks if the ICIR has a valid structure for clustering. That is, (1) the considered ICIR must have at least two children; (2) at least one selector of ICIR root must be a range of values or a wildcard. The joined rules in a cluster must form a continuous range (with or without overlapping) and must be subset, superset or equal the corresponding root selector; and (3) for root selectors that do not have multiple values, rules in the cluster must have the same value as root, or at least one of them must be a wildcard. Then it sorts children by destination port in ascending order.

Algorithm 3. Inconsistency Characterization.

```
Func doClassification(in Rule: root, Rule: cluster; out
String: conflictType)
Alg
// Root is last rule
   if (root.getPriority()>cluster.getLastRulePriority()) {
      if (cluster == root)
         conflictType = "Root is exact shadowed by union"
      else if (superset(cluster, root))
         conflictType = "Root is shadowed by cluster"
      else if (subset(cluster, root))
         conflictType = "Root is generalization of cluster"
      else
         conflictType = "Root and cluster are correlated"
   }
else { // Root is first rule
      if (superset(cluster, root)
         conflictType = "Cluster is generalization of root"
      else if (subset(cluster, root)
         conflictType = "Cluster is shadowed by root"
      else
         conflictType = "Root and cluster are correlated"
   }
   return conflictType
End Alg
```

Next, the algorithm checks if root is the last or first rule or is in between. If root is in between it divides children in two lists: rules that come before and rules that go after root, as also was explained before. Finally, if clustering is possible, it calls Algorithm 2, and if not, it calls directly the inconsistency characterization (Algorithm 3). Algorithm 2 also takes as input ICIR root and children. This algorithm implement the heuristic as it has been described in the previous section. Characterization algorithm (Algorithm 3) follows directly the extended definitions proposed in an earlier section. Algorithm 3 takes as input ICIR root and the clusters of that ICIR. Then, it checks each type of inconsistency using the equality, subset and superset operations. Note that characterization is different depending on the relative priority of the ICIR root (if it is the first or last rule). Algorithm 3 is in *O(c)*. Result is returned as a text string. The combined worst case complexity of the three algorithms is in *O(clogc)*. As these algorithms must be run for each ICIR, the final time complexity is in *O(h*clogc),* where *h* is the cardinality of the diagnosis set (or the number of ICIRs) and *c* is the number of children of each ICIR. Note that the combinatorial part of the inconsistency characterization problem is only the clusterization (where the heuristic has been used), and not the characterization itself.

5 Conclusions and Future Works

In this paper, we have analyzed the inconsistency characterization problem in firewall rule sets. We have proposed a complete and formal inconsistency characterization for clusters of rules in order to obtain a one-to-many characterization. The analysis of the characterization problem enabled us to identify and isolate the combinatorial part of

it. We showed that the combinatorial problems to be solved are very small due to the decomposition made in the diagnosis process, which has effectively reduced a worst case $O(2^n)$ problem in several $O(2^c)$ ones, with $n>>c$. As a proof of concept we have proposed a heuristic and algorithms that solve the characterization problem in worst case polynomial time. Algorithms are capable of handling full ranges in rule selectors without doing rule decorrelation. Results are given over the original ACL. In the future, we are going to design optimal algorithms, and compare its performance with other proposals.

References

1. S. Pozo, R. Ceballos, R. M. Gasca. “Model Based Development of Firewall Rule Sets: Detecting and Diagnosing Errors.” Information and Software Technology Journal, Elsevier, Spring 2008. Accepted, to appear.
2. S. Luis, M. Condell. "Security policy protocol." IETF Internet Draft IPSPSPP-01, 2002.
3. H. Hamed, E. Al-Shaer. "Taxonomy of Conflicts in Network Security Policies." IEEE Communications Magazine Vol.44, No.3, 2006.
4. E. Al-Shaer, Hazem H. Hamed. Modeling and Management of Firewall Policies". IEEE eTransactions on Network and Service Management (eTNSM) Vol.1, No.1, 2004.
5. J. García-Alfaro, N. Boulahia-Cuppens, F. Cuppens, Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies, Springer-Verlag International Journal of Information Security (Online) (2007) 1615-5262.
6. S. Pozo, R. Ceballos, R. M. Gasca, “Fast Algorithms for Consistency-Based Diagnosis of Firewalls Rule Sets.” International Conference on Availability, Reliability and Security (ARES), Barcelona, Spain. IEEE Computer Society Press, March 2008.
7. L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, P. Mohapatra. FIREMAN: A Toolkit for FIREwall Modelling and ANalysis. IEEE Symposium on Security and Privacy (S&P’06). Oakland, CA, USA. May 2006.