A Heuristic Process for Local Inconsistency Diagnosis in Firewall Rule Sets[1]

S. Pozo
Department of Computer Languages and Systems, Computer Engineering High School, University of Seville, Spain
Email: sergiopozo@us.es

R. Ceballos, R.M. Gasca
Department of Computer Languages and Systems, Computer Engineering High School, University of Seville, Spain
Email: {ceball,gasca}@us.es

Abstract— **Writing and managing firewall ACLs are hard and error-prone tasks for a wide range of reasons. During these tasks, inconsistent rules can be introduced. An inconsistent firewall ACL implies in general a design error, and indicates that the firewall is accepting traffic that should be denied or vice versa. However, the administrator is who ultimately decides if an inconsistent rule is a fault or not. Although many algorithms to diagnose inconsistencies in firewall ACLs have been proposed, they have different drawbacks regarding many aspects of the consistency management problem, which can prevent their use in a wide range of real-life situations. The most important one is that they give complete and minimal results, but their algorithmic complexity is too high, making the problem intractable for even reasonably-sized ACLs. In this paper we present an analysis of the consistency diagnosis problem in firewall ACLs. Based on this analysis, we propose to split the process in several parts that can be solved sequentially: inconsistency detection and isolation, inconsistent rules identification, and inconsistency characterization. Our algorithms are the first which can solve the detection, isolation, and identification problems in quadratic time complexity, giving complete but not necessarily minimal results. A theoretical complexity analysis as well as experimental results with real ACLs is given.**

Index Terms— **diagnosis, consistency, conflict, anomaly, firewall, acl, ruleset**

I. INTRODUCTION

Your goal is to simulate the usual appearance of papers in a Journal of the Academy Publisher. We are requesting that you follow these guidelines as closely as possible.

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). Firewalls use obligation policies. Obligation policies can be represented as Event Condition Action Rules (ECA Rules) that must perform certain actions over a subject (in the firewall case, the subject is always traffic) when certain events occur. Thus, a layer 3 Firewall ACL is in general a list of linearly ordered (total order) condition/action rules. The *condition* part of a rule is a set of condition attributes or selectors. The *condition* set is typically composed of five elements, which correspond to five fields of a packet header [14]. Some of these selectors can be expressed as naturals, and others as both naturals and intervals of naturals. In firewalls, the process of matching TCP/IP packets against rules is called filtering. 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. The *action* part of the rule represents the action that should be taken for a matching packet. In firewalls, two actions are possible: *allow* or *deny* a packet. A firewall ACL is commonly denominated a *rule set*.

Writing ACLs is a time-consuming and error-prone task for several reasons [16]. One of the main ones is that networks have different access control requirements (or objectives) which must be translated by a network administrator into firewall ACLs. The gap between the high-level access control requirements and low-level ACLs is too wide. Low-level firewall languages are, in general, difficult to learn, use and understand, and are very different from each other in syntax and semantics. Although many high-level languages have been proposed in order to reduce design complexity and (thus design faults), node of them have been widely adopted by the industry for various reasons [2]. In addition, its use does not guarantee that the resulting ACL is fault-free, thus a method to diagnose design faults must be applied in order to correct them prior to ACL deployment.

[1] This paper is a revised and extended version of a paper published in SECRYPT 2008 [11].

doi:10.4304/jnw.4.8.698-710

Furthermore, complexity of networks is constantly increasing, as changes in requirements, topology, etc. occur with higher frequency and density. According to Taylor [14], the number of rules in a firewall ACL usually ranges between a few ones and five thousand. For this reason, the management of firewall ACLs is also a very hard task. For example, changing from one firewall platform to another often means a complete rewrite of the ACL, and thus new faults can be introduced. In addition, ACL updates can also introduce new design faults [10].

One of the most important and frequent faults during ACL design and management are inconsistencies. A firewall ACL with inconsistent rules implies in general design faults, and indicates that the firewall is accepting traffic that should be denied or vice versa. This can result in severe problems such as unwanted accesses to services, denial of service, overflows, etc. ACL consistency is of extreme importance in several contexts, such as highly sensitive applications (e.g. health care). Thus, algorithms and tools to automatically isolate and characterize inconsistencies must be provided in order to give firewall administrator enough information to correct them and reduce the number of faults in firewall ACLs.

Many algorithms to diagnose inconsistencies in firewall ACLs have been proposed, and in all of them is the firewall administrator who ultimately decides which rules have to be corrected. However, these algorithms have many drawbacks regarding different aspects of the consistency management problem. The most important one is that they pre-process the firewall ACL using different types of non-trivial decompositions in order to use more efficient abstract data types and techniques. However, the proposed decomposition techniques increase the number of rules in the ACL and have worst-case exponential time and space complexity. As a consequence, results of these consistency management algorithms are given over the modified ACL, and have to be interpreted by the firewall administrator. Furthermore, their time and space complexity is very important, since these algorithms are being used in a new range of applications in resource-constrained devices in ubiquitous networks, such as ad-hoc network node real-time ACL updates [10], real-time IDS or IPS rule updates, etc.

In this paper we propose to take a different approach in order to make the problem tractable for real-life, big rule sets. We propose to divide consistency diagnosis in three sequential steps (Fig. 1). This paper focuses in the first two parts of the process (detection and isolation, and identification). In this paper, we propose best case $O(n)$ and worst case $O(n^2)$ time complexity order independent detection and isolation, and identification algorithms with the number of rules of the rule set, n. Algorithms are capable of handling full ranges in rule selectors without doing rule decorrelation, range to prefix conversion, or any other pre-process. Results are returned over the original unmodified ACL. The process does not cope with redundancies, as we consider redundancy diagnosis a different problem because redundancies do not change the ACL semantics, but only affect the performance of the matching algorithm, which is not the focus of this paper.

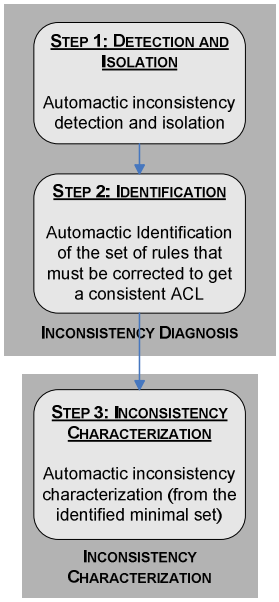

Figure 1. Consistency diagnosis process

This paper is structured as follows. In section II, we analyze the internals of the local consistency management problem in firewall rule sets and formalize it. In section III we propose the consistency-based diagnosis algorithms, give a theoretical complexity analysis and experimental results with real rule sets, which validate our proposal. In section IV we review related works comparing them to our proposal. Finally we give some concluding remarks in section V.

II. ANALYSIS OF THE CONSISTENCY PROBLEM

To understand the problem, it is important to first review the inconsistencies characterized in the bibliography. A complete characterization that includes shadowing, generalization, correlation and redundancy has been given in [6]. Although all of these are inconsistencies, usually not all are considered to be design faults, as they can be used to cause desirable effects. Is the firewall administrator who ultimately decides which rules have to be corrected or removed.

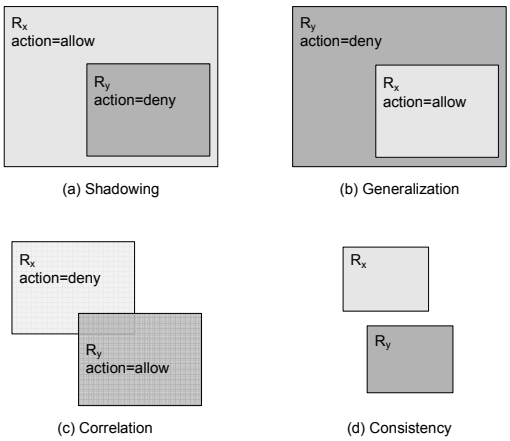

Figure 2. Graphical representation of three inconsistencies

TABLE 1: EXAMPLE OF A FIREWALL RULE SET

Priority/ID	Protocol	Source IP	Src Port	Destination IP	Dst Port	Action
R1	tcp	192.168.1.5/32	any	*.*.*.*/0	80	deny
R2	tcp	192.168.1.*/24	any	*.*.*.*/0	80	allow
R3	tcp	*.*.*.*/0	any	172.0.1.10/32	80	allow
R4	tcp	192.168.1.*/24	any	172.0.1.10/32	80	deny
R5	tcp	192.168.1.60/32	any	*.*.*.*/0	21	deny
R6	tcp	192.168.1.*/24	any	*.*.*.*/0	21	allow
R7	tcp	192.168.1.*/24	any	172.0.1.10/32	21	allow
R8	tcp	*.*.*.*/0	any	*.*.*.*/0	any	deny
R9	udp	192.168.1.*/24	any	172.0.1.10/32	53	allow
R10	udp	*.*.*.*/0	any	172.0.1.10/32	53	allow
R11	udp	192.168.2.*/24	any	172.0.2.*/24	any	allow
R12	udp	*.*.*.*/0	any	*.*.*.*/0	any	deny

These inconsistencies except redundancy are graphically presented in Fig. 2. For the sake of simplicity, only pair wise inconsistencies with one selector are represented. An example of an ACL is presented in Table 1.

In this paper, we propose to divide consistency management in three sequential steps (Fig. 1). At the first step, all rules that cause inconsistencies are detected and isolated, if any. Then, the set of rules that cause the detected inconsistencies should be identified. Their correction or removal guarantees that the resulting rule set is consistent. This set of identified rules must be as small as possible, in order to give useful results in rule sets with a high number of inconsistencies. These two steps are called Inconsistency Diagnosis. Finally, the identified inconsistent rules should be characterized among an established taxonomy of firewall rule set inconsistencies. This paper is focused in these two parts of the consistency diagnosis problem. The third and last problem, minimal inconsistency characterization, is combinatorial [12]. Furthermore, diagnosis is also rule-order independent, contrarily to characterization. The main difference of this work with other ones is that other authors apply brute force algorithms to solve directly the characterization problem, with no previous diagnosis. This yields algorithms that cannot be applied to big rule sets. With the proposed approach, the same characterization algorithms can be applied to several smaller problems, rather than to the full rule set. However, the number of these smaller problems is not minimal with the heuristic algorithms proposed in this paper. In addition, heuristic characterization algorithms [12] can also be used to give approximate results in a reasonable time, even for really big rule sets (with more than 10000 rules).

A. 1..1 and 1..n Consistency in Firewall Rule Sets

First, it is needed to formalize a firewall rule set.

- Let *RS* be a firewall rule set consisting of *n* rules, $RS = \{R_1, ...R_n\}$.
- Let $R =< H, Action >, H \in \mathbb{N}^5$ be a rule, where $Action = \{allow, deny\}$ is its action.
- Let $R_j[k], 1 \le j \le n, k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$ be a selector of a firewall rule R_j.
- 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 its action is going to be taken first, and vice versa.

Attending to Al-Shaer characterization, two rules *(R_x, R_y)* are correlated if they have a relation between all of its selectors, and have different actions. Fig. 2(c) represents a correlation inconsistency between two rules with one selector each. As the figure shows, the relation between the rules is not subset, nor superset, nor equal (rules R_1 and R_3 of Table 1 are correlated). Fig. 2(a) represents a shadowing inconsistency between two rules. The relation is equality or subset of the shadowed rule, R_y, respect to the general rule, R_x, with $R_x > R_y$ (R_4 is shadowed by R_3 in Table 1 example). Fig. 2(b) represents a generalization inconsistency between two rules, which is the inverse of shadowing respect to the priority of the rules. The relation is superset of the general rule respect to the other one (R_2 is a generalization of R_3 in Table 1 example).

Since we are only interested in diagnosis and not in its characterization, let's try to remove names and give a general case of inconsistency based on these inconsistency characterizations (except redundancy). In a closer look at shadowing and generalization inconsistencies in Fig. 2, it can be seen that, in reality, these two inconsistencies are the same one, and the only thing that differentiates them is the priority of the rules. Thus, if priority is ignored, these two inconsistencies are special cases of a correlation. That is, shadowing can be redefined as a correlation where all selectors of one rule (the shadowed one) are subsets or equal of the general rule. As generalization is the inverse with respect to the priority of shadowing, a generalization inconsistency can also be redefined as a correlation where of all selectors of a rule (the general one) are supersets of the other rule. So, the correlation inconsistency can be redefined as the superset of all inconsistencies, representing the most general case. For that reasons, it is possible to define *rule inconsistency* in only one priority independent case that

recognizes all characterized inconsistencies (Axiom 2.1.1). This is a key issue for the proposed diagnosis algorithms.

Axiom 2.1.1. Rule inconsistency. Two rules $R_i, R_j \in RS$ are *inconsistent* if and only if the intersection of *each of all* of its selectors *R[k]* is not empty, and they have different actions, *independently of their priorities*. The inconsistency between two rules expresses the possibility of an undesirable effect in the *semantics* of the rule set. The inconsistency is considered to be a fault if an administrator identifies the behaviour of the executed ACL as being causing undesirable effects (or having errors). The semantics of the rule set changes if an inconsistent rule is corrected or removed.

$$Inconsistent(R_i, RS), 1 \le i \le n \Leftrightarrow \exists R_j \in RS, 1 \le j \le n, j \ne i \bullet$$

$$R_i[k] \cap R_j[k] \ne \varnothing \wedge R_i[Action] \ne R_j[Action]$$

$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

Inconsistency of one rule in a RS

$$Inconsistent(R_i, R_j, RS), 1 \le i, j \le n, i \ne j \Leftrightarrow$$

$$R_i[k] \cap R_j[k] \ne \varnothing \wedge R_i[Action] \ne R_j[Action]$$

$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

Inconsistency between two rules in a RS

Attending to Axiom 2.1.1, all cases represented in Fig. 2 are of the same kind, and are called *inconsistencies* without any particular characterization. As it has been demonstrated in the proof, rule priorities are not required to detect inconsistencies.

Axiom 2.1.1 can also be used for more than two rules, since the case of *one to n* rule inconsistency can be decomposed in several independent pair wise inconsistencies (Lemma 2.1.1).

Lemma 2.1.1. Axiom 2.1.1 can be extended to capture inconsistencies between one and more than one rules (that is *1..n* inconsistencies), because a *1..n* inconsistency can always be decomposed in *n 1..1* inconsistencies.

Proof. Let us reason by contradiction. Suppose there is a *1..n* inconsistency between a rule R_z and a set of rules $R_1..R_n$, that is *Inconsistent*(R_z, $R_1..R_n$, *RS*)=*true*. Suppose that this *1..n* inconsistency cannot be decomposed in *n 1..1* inconsistencies. If that inconsistency exists, then the rules $R_1..R_n$ are consistent between them (they must have the same action, or in other case R_z could not be inconsistent with the whole set of rules). For that reason, all selectors of each rule of the $R_1..R_n$ set, must necessarily intersect with R_z selectors. Thus, R_z must necessarily be inconsistent with all of them in an independent manner. Note that if the rules in the $R_1..R_n$ set overlap or not between them is not important, since this could indicate a partial or total redundancy between one or more rules in the set.

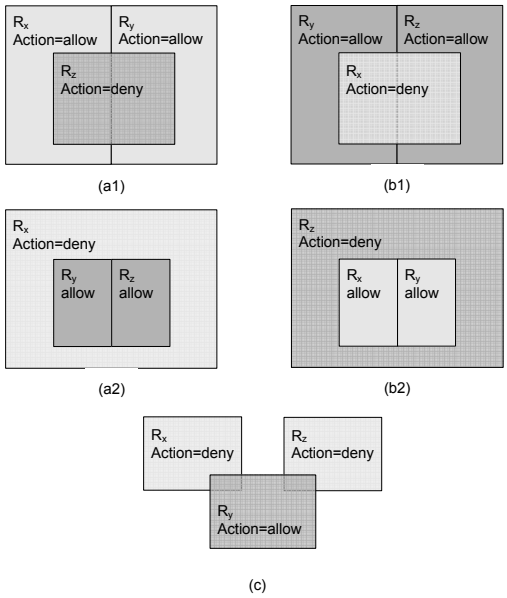

Figure 3. Graphical representation of inconsistencies between three rules

For example, all base situations are presented in Fig. 3, which is an extension to Fig. 2. This figure is a simplification to three inconsistent rules, but can easily be extended to more rules that can be composed in several ways.

Fig. 3(a1) represents an inconsistency where the union of two independent rules *(R_x, R_y)* overlap with another one, R_z (Fig. 4(a) taken from [5] exemplifies this situation). As R_x is inconsistent with R_z, and R_y is also inconsistent with R_z, both in an independent manner, this situation can be decomposed in two independent inconsistencies.

Fig. 3(a2) presents a similar situation, where R_x overlaps with the union of (R_y, R_z). This situation is also decomposable in two independent inconsistencies: R_x Inconsistent with R_y, and R_x with R_z. Note that, in order to diagnose inconsistencies, the priority of the rules is not necessary.

$$Rx: \{port \in [10-50]\} \Rightarrow \{allow\}$$

$$Ry: \{port \in [40-90]\} \Rightarrow \{allow\}$$

$$Rz: \{port \in [30-80]\} \Rightarrow \{deny\}$$

(a)

$$Ry: \{port \in [10-50]\} \Rightarrow \{allow\}$$

$$Rz: \{port \in [40-90]\} \Rightarrow \{allow\}$$

$$Rx: \{port \in [0-100]\} \Rightarrow \{deny\}$$

(b)

$$Rx: \{port \in [0-50]\} \Rightarrow \{deny\}$$

$$Rz: \{port \in [60-100]\} \Rightarrow \{deny\}$$

$$Ry: \{port \in [40-70]\} \Rightarrow \{allow\}$$

(c)

Figure 4. Inconsistency examples

The situations presented in Fig. 3(b1) and Fig. 3(b2) are the inverse of the two previous ones respect to the action. Thus, the decomposition is analogous. This situation is exemplified in Fig. 4(b). Finally, Fig. 3(c) represents a relation with three overlapping rules (an example is in Fig. 4(c)). This situation can also be decomposed in two independent ones: R_x inconsistent with R_y, and R_y with R_z.

At this point, we propose a formalization and an extension of the fault characterization provided by Al-Shaer in order to recognize *1..n* inconsistencies. The proposed fault characterization is also complete (as it is an extension of Al-Shaer work) based on the relationships that can be established between the selectors of rules: equality, subset and superset.

- **Shadow.** A rule R_y is shadowed by another rule R_x, with $R_x > R_y$, if all of its selectors to or supersets of the selectors of R_y, and R_x and R_y have different actions.

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

Shadow

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

Exact shadow

This definition can be extended to support a set of rules with the same action in R_x or R_y (but not in both). If R_x is a set of rules and R_y is a rule, then R_y is shadowed by R_x. Similarly, if R_x is a rule, and R_y is a set, then R_y are shadowed by R_x.

- **Generalization.** It is the inverse of shadow respect to the priority. A rule R_y is a generalization of R_x, with $R_x > R_y$, if all of the selectors of R_x are subsets of the selectors of R_y, and both rules have different actions. R_x is usually considered an exception and not a fault. Again, sets can be formed.

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

- **Correlation.** Two rules R_x and R_y are correlated if they have different actions, and selectors of R_x intersect with the corresponding selectors of R_y, but R_x and R_y do not have a shadow, exact shadow or generalization relation. Correlation is independent of rule priority. This definition can also be extended to sets of rules.

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

Note that inconsistencies that could be generated during rule set updates (removals, insertions, and modifications) is a topic not covered in this paper, but has been covered in another work [10].

III. CONSISTENCY-BASED DIAGNOSIS OF RULE SETS

The presented analysis has motivated the separation of characterization from diagnosis, and to solve the diagnosis problem as a first step for the optimal inconsistency characterization problem. As it is going to be shown, the result of the diagnosis process is the identification of a set of rules that cause the inconsistencies in the rule set and for each one, the set of the rules which they are inconsistent with. Each of these sets and their corresponding identified conflicting rule can be taken as input to the characterization part of the process, resulting in an effective computational complexity reduction (solving several small combinatorial problems is faster than solving a big one). However, as the proposed algorithm for the identification of inconsistent rules is not minimal, the application of an optimal characterization algorithm to its result may be senseless. In contrast, heuristic characterization algorithms [12] can be used, with a big improvement in computational complexity for the full process.

In this section, two algorithms which implement Axiom 2.1.1 and Lemma 2.1.1 and the diagnosis process explained in the previous section are presented. Algorithms are capable of handling ranges in all selectors without modifications to the input rule set.

A. Step 1. Detection and Isolation of Inconsistent Pairs of Rules

The first step of the process detects the inconsistent rules of the rule set and returns an Inconsistency Graph (IG, Definition 3.1.1) representing their relations. Note that the detection and isolation process, like Axiom 2.1.1, is order independent. Also note that the presented algorithm is complete, as it implements Axiom 2.1.1 (which is complete).

Definition 3.1.1. Inconsistency Graph, IG. An IG is an undirected, cyclic and disconnected graph whose vertices are the inconsistent rules of the rule set, and whose edges are the inconsistency relations between the inconsistent rules. Note that |IG| is the number of inconsistent rules in *RS*, and ||IG|| corresponds to the number of inconsistencies pairs of rules in *RS,* or simply the number of inconsistencies in *RS.*

Algorithm 1. Inconsistency Detection and Isolation

```
1.  Func detection(in List: ruleSetAllow, ruleSetDeny;
2.  out Graph: ig)
3.  Var
4.     Rule ri, rj
5.     Integer i, j
6.  Alg
7.     for each j=1..ruleSetAllow.size() {
8.        rj= ruleSetAllow.get(j)
9.        for each i=1..ruleSetDeny.size() {
10.          ri = ruleSetDeny.get(i)
11.          if (inconsistency(ri, rj)) {
12.             ig.addVertex(ri)
13.             ig.addVertex(rj)
14.             ig.addEdge(ri, rj)
15.          }
16.       }
17.    }
18. End Alg
19.
20. // Implements the Inconsistency Definition
21. Func inconsistency(in Rule: rx, ry; out Boolean: b)
22. Var
23.    Integer i
24. Alg
25.    b = true
26.    i = 1
27.    while (i<=rx.selectors.size() AND b)
28.       b = b AND intersection(rx.getSelector(i),
29.                              ry.getSelector(i))
30.       i=i+1
31.    }
32. End Alg
33.
```

Figure 5. Inconsistency detection algorithm

Algorithm 1 presented in Fig. 5 (implemented in Object Oriented paradigm and using abstract data types) exploits the order independence of the inconsistency axiom and only checks inconsistencies between rules with different actions, dividing the ACL in two lists, one with *allow* rules and the other with *deny* ones. The algorithm receives two rule sets. One of them consist of *allow* rules and the other of *deny* rules of the original rule set. This decomposition is trivial and runs in linear complexity with the number of rules in *RS*. The algorithm takes one of the rule sets and, for each rule, it checks if there is an inconsistency with other rules in the other one. As all inconsistencies can be decomposed in two by two relations, there is no need to check combinations of more than two rules. Each time the algorithm finds an inconsistency between a pair of rules, the two rules are added as vertices to the IG, with a non directed edge between them. The algorithm returns ends returning the IG. Since all possibilities have been checked, Algorithm 1 returns the isolation of all possible inconsistent rules

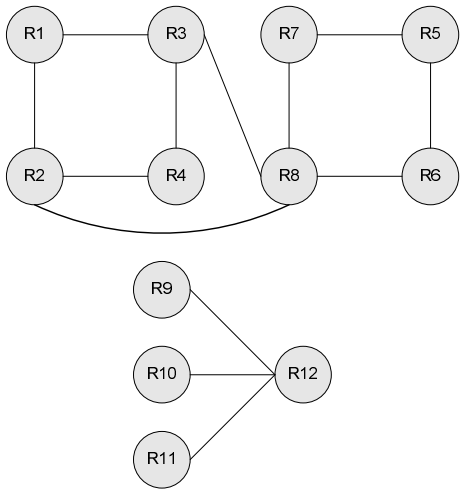

Figure 6. Inconsistency graph

(i.e. it is complete). Fig. 6 presents the resulting IG of the Table 1 example.

Time complexity of Algorithm 1 is bounded by the two nested loops (lines 7 and 9). Each rule in *ruleSetAllow* is tested for inconsistency against rules in *ruleSetDeny*. The worst case for the loop is reached when *ruleSetAllow.size()= ruleSetDeny.size()* (i.e. half rules *allow* and the other half *deny*), and the best case when *ruleSetAllow.size()=n* and *ruleSetDeny.size()=1* or *ruleSetAllow.size()=1* and *ruleSetDeny.size()=n*. Thus, the complexity of the improved isolation algorithm depends on the percentage of *allow* and *deny* rules over the total number of rules.

However, there are other inner operations that should be analyzed in lines 11 to 14. The first one, in line 11, is *inconsistency()* which is composed of an iteration. This operation implements the inconsistency lemma. In typical firewall ACLs, *k=5*, and thus the iteration runs 5 times. Anyway, the iteration is bounded by the number of selectors, which is always a constant *k*.

In addition, inside the iteration there is an intersection between each selector (lines 28 to 30). The typical 5 selectors of firewall ACLs (Table 1) are integers or intervals of integers. Knowing if two ranges of integers intersect can be done in constant time with a trivial algorithm which compares the limits of the intervals. Knowing if two IP addresses intersect can also be easily done in constant time by comparing their network addresses and netmasks. Other operations of the inner loop (lines 12 to 14) are the graph-related ones. If the graph is based on hash tables, vertex and edge insertions run in constant time, except in some cases where rehashing could be necessary.

For all these reasons, the complexity of the two nested loops is only affected by a constant factor in all cases, which depends on the number of selectors, *k*. Thus, worst case time complexity of the isolation algorithm is in $O(n^2)$, best case is in $O(n)$, and average case is in $O(n \cdot m)$ with the number of *allow* rules, *n*, and *deny* rules, *m* in the ACL.

Space used by Algorithm 1 is the sum of the space needed to store the ACL, and the one needed for the graph. In best case the graph would have *n* vertices and *n-1* edges. In the worst case, there could be *n-1* inconsistent rules and also *n-1* edges per vertex. Note that the space needed to store an edge is fewer than the needed to store a vertex, since only a reference between vertices is needed.

B. Identification of Inconsistent Rules

The second and last step of the diagnosis process identifies the set of rules that cause the inconsistencies from the isolated set of inconsistent pairs of rules (the result of the previous step) with a heuristic algorithm. Algorithm 2 (Fig. 7) was initially presented in [9]. It receives the IG as input and takes iteratively the vertex with the greatest number of adjacencies (lines 6 and 7), that is, the vertex with the greatest number of inconsistencies, *v*. Then, an independent cluster of inconsistent rules (ICIR, Definition 3.2.1) is created as a tree with *v* (the conflicting rule of the cluster) as its root, and its adjacents (the inconsistent rules) as leaves (lines 7 to 11). The root of all ICIRs form the Diagnosis Set (DS, Definition 3.2.2), or the set of rules that must be removed to get a consistent rule set. Then, *v* and its edges are removed from the IG (line 13). If vertices with no edges are left in the IG, then these vertices are also removed (line 14), since they are consistent by definition (they are rules with no relations with others). As inconsistencies have been decomposed in pair wise relations, ICIRs are always formed by two levels.

Definition 3.2.1. Independent Cluster of Inconsistent Rules, ICIR. An *ICIR(root, CV)* is a two level tree, rooted in the rule *root* and where *CV* is a set of rules (its leaves). It represents a cluster of mutually consistent rules, *CV,* which are at the same time inconsistent with their respective *root. ICIR(root)* is the rule which has the greatest number of inconsistencies with other rules of the same cluster. For that reason, the *root* of each ICIR is different, and all *roots* form a disjoint set of rules. Note that the action *ICIR(root)* is the contrary of the actions of all of its leaves in *CV*.

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

Definition 3.2.2. Diagnosis Set, DS. This is the set of rules that cause the inconsistencies, and coincide with the *root* of all ICIRs. If these inconsistencies are removed from *RS*, *RS* becomes consistent.

$$\text{Let } ICIRS = \{ICIR_1, ..., ICIR_m\}$$
$$\text{be the set of all ICIR of a given } RS, \text{ then}$$
$$DS = \{ICIR_1(root), ..., ICIR_m(root)\} \bullet$$
$$RS - DS \text{ is consistent}$$

Algorithm 2. Inconsistent Rule Identification

```
1.  Func identification(in Graph:ig; out List of
2.  Tree:icirs)
3.  Var
4.     Tree icir
5.  Alg
6.     while (ig.hasVertices()) {
7.        Vertex v = ig.getMaxAdjacencyVertex();
8.        List adj = ig.getAdjacents(v)
9.        icir.createEmptyTree()
10.       icir.setRoot(v)
11.       icir.addChildren(adj)
12.       icirs.add(icir)
13.       ig.removeVertexWithEdges(v)
14.       ig.removeNotConnectedVertices()
15.    }
16. End Alg
```

Figure 7: Inconsistency identification algorithm

A graphical representation of a partial trace of Algorithm 2 over the previous IG is presented in Fig. 8. At the first iteration, R_8 is selected because it has four inconsistencies (the greatest number of adjacent vertices). Then, it is removed and the first ICIR tree is formed with R_8 as root, thus R_8 is a conflicting rule and will be in DS. At the second iteration, R_{12} is selected because it has three inconsistencies (it is the vertex with the biggest number of adjacent vertices). Then it is removed and the second ICIR is formed. Vertices R_9, R_{10} and R_{11} are also removed from the IG because they had no adjacent vertices. At the third iteration, there is a possibility of selecting R_5, R_1, R_2, R_3 and R_4 as the next vertex. The selection of one or other is arbitrary. In this example, the algorithm selects R_5, removes it from the IG with all its edges and forms the third ICIR. At the end of this iteration the IG is only composed of a cycle of four vertices: R_1, R_2, R_3, and R_4. The algorithm selects to remove R_1 at the fourth iteration and R_4 at the fifth and last one, removing the vertices and edges, and forming ICIR 4 and ICIR 5 respectively. Since no more vertices are left in the IG, the algorithm finishes with a diagnosis set with cardinality five, containing the rules DS={R_8, R_{12}, R_5, R_1, R_4}.

If the rules from the DS are removed from *RS*, *RS* becomes consistent.

It has been noted in the explanation of the trace that at some time, the IG could have cycles. Cycles have a special property. In a cycle, the selection of a rule as the next to be processed is random, since all of them have the same number of adjacent vertices. Depending on the rule that is selected, the algorithm forms different ICIRs. In this example, R_1 and then R_4 have been selected but if for example, R_2 and R_3 were selected, different ICIRs would have been formed (Fig. 9). The final number of ICIRs formed is always the same in this special case, since the number of vertices and edges removed is the same with independency of the actual vertices removed from a cycle. Most important fact is that the two groups of ICIRs

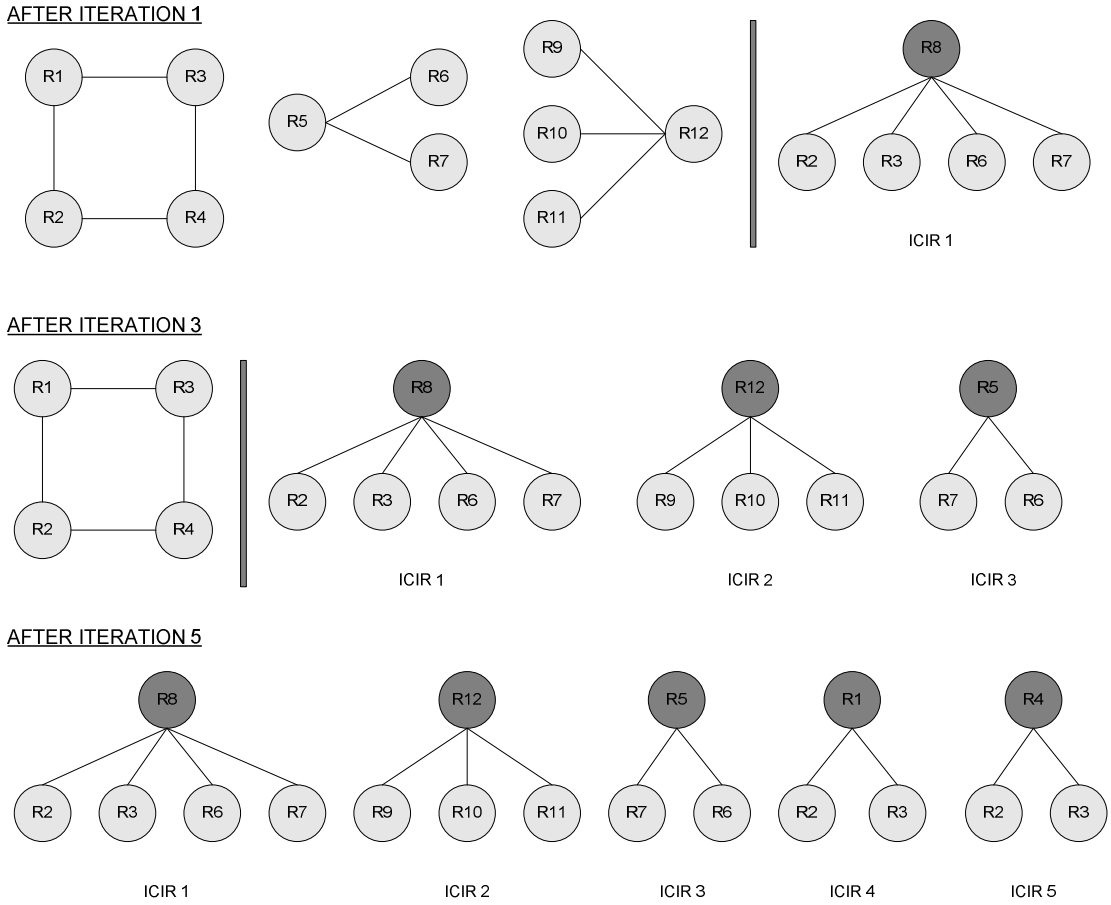

Figure 8. Partial trace of Algorithm 2 applied to the example IG

are equivalent, since the rules involved in the inconsistencies are the same, only its order has changed. This property of cycles is derived from the fact that all the rules in a cycle are correlated between themselves. For example, R_1 appears in ICIR 4b and 5b, because R_1 is correlated with both roots R_2 and R_3. Looking at ICIR 4a, the relations are ICIR4a={$root=R_1$, R_2, R_3}, which is formed of that three rules.

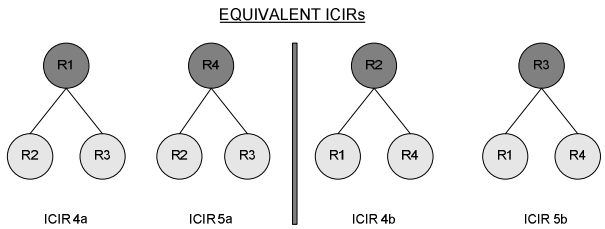

Figure 9. Equivalent ICIRs from Fig. 8 It. 3

Time complexity of Algorithm 2 is bounded by the loop of line 5, which runs as many times as ICIRs are formed (it corresponds with the cardinality of the Diagnosis Set, $|DS|$). The worst case is reached, as in Algorithm 1, when $ruleSetAllow.size()=ruleSetDeny.size()=n/2$ (Fig. 10(b)), resulting in a $|DS|=n/2$. In this case, *getMaxAdjacencyVertex()* (line 7), a maximum calculus, runs in $O(n)$ with the number of vertices of the graph (the number of inconsistencies). Operations of lines 8, 9, 10, 11, and 12 run in constant time. *removeVertexWithEdges()* (line 13) runs in linear time with the cardinality of its adjacency list ($n/2-1$ in the worst case). Finally, *removeUnconnectedVertices()* (line 14) is also linear with the number of vertices in the graph at each iteration, $O(n)$. Thus, the resulting worst case time complexity of Algorithm 2 is in $O(|DS|\cdot(n+n/2-1+n))=O(n/2\cdot n)=O(n^2)$.

The best case is reached, as in Algorithm 1, when $ruleSetAllow.size()=n$ and $ruleSetDeny.size()=1$ or vice versa (Fig. 10(a)). The IG only has one vertex, v, connected to all the other vertices. In this case, $|DS|=1$ and the algorithm is in $O(n)$. In an average case the algorithm is in $O(|DS|\cdot h)$, $|DS|<<h$ (h is the number of inconsistencies).

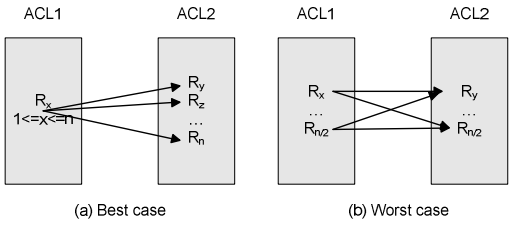

Figure 10. Identification best and worst cases

Algorithm 2 needs some space to store the ICIRs. Each ICIR needs space for its root and for the conflictive rules. But note that, as the algorithm is creating the ICIRs, the corresponding vertices and edges are removed from the IG, and thus at each iteration only the space to store the adjacency list of the removed vertex is necessary. Complexities are presented in Table 2.

The result of the diagnosis process is the set of all ICIRs. As each ICIR represents a different independent inconsistency, exhaustive search optimal characterization algorithms can be applied to each one independently, reducing the effective computational complexity of the whole process. Furthermore, heuristic characterization algorithms can also be applied [12]. Also note that the presented proposal makes no assumptions about how selector ranges are expressed. This is important, because as the original rule set is directly used by algorithms, inconsistency results are given over it.

TABLE 2: ISOLATION AND IDENTIFICATION TIME COMPLEXITIES

Number of inserted rules	Best case	Average case	Worst case	Space Worst
Detection and Isolation	$O(n)$	$O(n \cdot m)$	$O(n^2)$	n *Rules*· h *Edges*
Identification	$O(n)$	$O(\|DS\| \cdot h)$, $\|DS\| << h$	$O(h^2)$	n *Rules*· h *Edges*
Combined (Diagnosis)	$O(n)$	$O(n \cdot m)$	$O(n^2+h^2)$	n *Rules*· h *Edges*

C. Experimental Results

In absence of standard rule sets for testing, the proposed heuristic process has been tested with real firewall rule sets (Tables 3 and 4). The first column represents the size of the rule set; the second one the percentage of deny rules over the rule set size; the third the cardinality of the Diagnosis Set, $|DS|$, (or the number of generated ICIRs); the fourth represents the average size of each ICIR (that is, the number of ICIRs divided by $|DS|$), or the average size of the characterization problems to be solved (how many rules are in them); the fifth the number of inconsistencies; from sixth to nineth the execution time of the isolation and identification parts of the process (trivial detection and isolation, proposed detection and isolation, identification, and the sum of the proposed isolation and identification). Results are provided in rule sets with and without wildcard rules (WR, *deny all* and *allow all* rules).

The conducted performance analysis represents a wide spectrum of cases, with ACLs of sizes ranging from 50 to 10600 rules, and percentages of *allow* and *deny* rules ranging from 2% to 65%. Recall that worst case is half rules *allow* and the other half *deny*. Also note that real ACLs have some important differences with synthetically generated ones. The most important one is the number of *deny* and *allow* rules: as real firewall ACLs are usually designed with *deny all* default policy, most rules are going to have *allow* actions. In ACLs designed with *allow all* policy, most rules would have *deny* actions. Also note that as the percentage of *allow* or *deny* rules decreases, the number of inconsistencies does necessarily not, because the number of inconsistencies depends on how many rules with different actions intersect. Tests have been run with and without WR, in order to know the impact these rules have in the complexity of the algorithms. However, WR provide no useful information to the diagnosis process, since they are inconsistent by definition with all rules with the contrary action. The result is that the worst case would not normally happen for the isolation algorithm in real firewall rule sets, but the dependence of the identification algorithm on the number of inconsistencies is completely arbitrary and thus cannot be predicted (however, note that leaving WR in the rule set results in a huge increase of inconsistencies because of the reasons stated above). Experiments were performed on a monothreaded Java implementation with Sun JDK 1.6.0 64-Bit Server VM, on an isolated HP Proliant 145G2 (AMD Opteron 275 2.2GHz, 2Gb RAM DDR400). Execution times are in milliseconds.

The experimental efficiency comparison of the proposed algorithms with others reviewed is a very difficult task for two main reasons. In one hand, there are no standard rule sets to be used. In other hand different proposals cover different parts of the process (for example, Al-Shaer proposal covers the characterization part, García-Alfaro proposal the full process, and our proposal the diagnosis part). One of the most important contributions of the presented experimental analysis is the average reduction of the diagnosis characterization problem, which is in average |DS|*Average ICIR size. Another important contribution is the improvement in time of the detection and isolation part of the diagnosis process, over the trivial isolation algorithm.

As Tables 3 and 4 and Fig. 11 show, execution time for the full diagnosis process is very reasonable, even in large rule sets. Note that rule set of sizes 238 and 450 are very near worst case. Rule set of size 10611 has not been represented to prevent image scale distortion, but note that even with a very high number of inconsistencies w/ WR (11866) execution time of the full process is 354ms. Take into account that a rule set of 10611 rules is a very big one [14]. Tables 3 and 4 and Fig. 11 present a lot of information. Note for example in Fig. 11(a) how little the performance of isolation between 2500 and 5000 rule set sizes differ, because they are very near the best case for isolationn (a real rule set could be in the 10-15% range of deny rules [14]). Also note that for the worst case of isolation (238 and 450 rules in the ACL), running time

TABLE 3. PERFORMANCE EVALUATION W/ WR

ACL Size	%Deny	|DS|	Average ICIR size	Number of Inconsistencies	Trivial Detection (ms)	Detection (ms)	Identification (ms)	TOTAL w/ WR (ms)
50	28.21	2	9	37	0.22	0.09	0.03	0.12
144	30.91	2	54	108	1.34	0.62	0.06	0.68
238	66.43	10	23	231	3.56	2.04	0.15	2.19
450	34.73	10	42	422	13.22	5.61	0.27	5.88
900	14.8	10	87	871	51.57	3.46	0.73	4.19
2500	6.97	32	104	3349	387.86	55.01	3.98	58.99
5000	1.98	6	822	4937	3160.09	64.33	7.90	72.23
10611	2.05	39	301	11866	12046.67	332.85	21.57	354.42

TABLE 4. PERFORMANCE EVALUATION W/O WR

ACL Size	%Deny	|DS|	Average ICIR size	Number of Inconsistencies	Trivial Detection (ms)	Detection (ms)	Identification (ms)	TOTAL w/o WR (ms)
46	24.32	0	-	0	0.13	0.07	0	0.07
140	29.63	0	-	0	1.21	0.53	0	0.53
228	68.89	8	12	96	3.07	1.88	0.04	1.92
440	34.97	8	12	96	12.35	5.37	0.04	5.41
889	14.71	8	12	96	49.58	12.53	0.04	12.57
2490	6.91	30	34	1020	382.06	51.88	0.76	52.64
4998	1.94	4	7	34	2231.33	60.84	0.02	60.86
10601	2.03	37	36	1468	13308.45	310.23	0.85	311.11

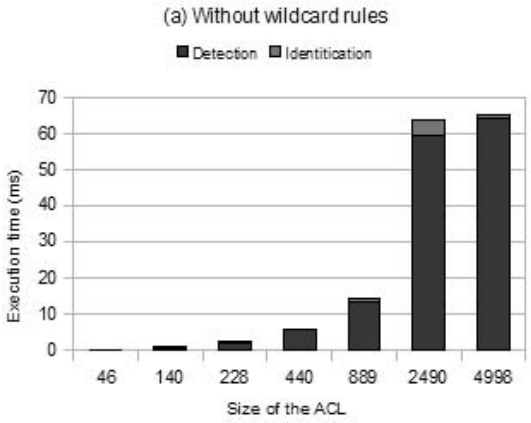

Figure 11(a). Running time w/o WR

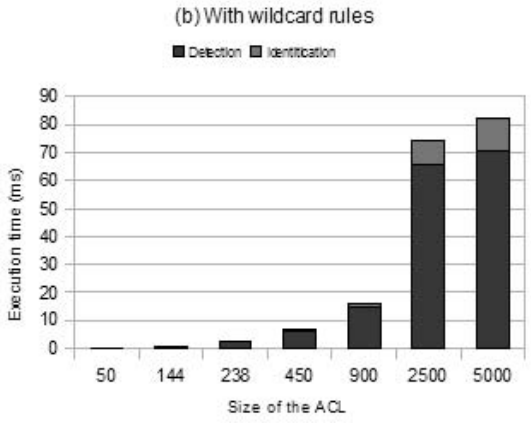

Figure 11(b). Running time w/ WR

almost doubles. Take into account that results are basically the same with and without WR for the isolation algorithm, since the only difference between them is the number of removed WR rules, which ranges from two to ten in the tested rule sets. Since the size of the problem for the isolation algorithm is measured in hundred or thousand of rules, removing such a little number of them, implies a negligible performance impact.

However, looking again at Tables 3 and 4 and Fig. 11, but for the identification algorithms, other important facts should also be explained. This time, complexity is bounded by |DS| and the number of inconsistencies. But note how the number of reported of inconsistencies varies depending on the removal or not of the WR. Now note how the performance of identification algorithm degrades when number of inconsistencies raise. In fact, leaving WR in the rule set, also implies an increase of the cardinality of the diagnosis set, |DS| and also of the average ICIR sizes. A WR implies that there would be an ICIR containing *f-1* rules, where *f* is the number of rules with the contrary action of the WR. For example, in the rule set with size 10611, a WR with *deny* action will generate an ICIR with 10348 rules, raising the average size of the ICIRs.

In addition, note how complexity is dominated by the isolation algorithm, which is the problem with the higher theoretic time complexity. The difference between leaving and removing WR does not have an important impact over the performance of the full process, also because this fact (WR does not affect very much the performance of the isolation algorithm).

Other important thing worth noting is related with problem reduction. The average ICIR size in Tables 3 and 4 represents the average number of children of each generated ICIR (the number of ICIRs is represented in |DS| column). That is, |DS| is the number of characterization problems to be optimally solved if optimal characterization algorithms are going to be used,

and Average ICIR Size is their average size. Clearly, solving (optimally or not) such small number of small problems is faster than solving a big combinatorial one over the full problem size, *n*.

Finally, Fig. 12 presents a comparison between the trivial isolation algorithm and the one presented in this paper. Note how the trivial algorithm scales quadratically with the number of rules. However, the complexity of the proposed algorithm depends on the percentage of *allow* and *deny* rules. As can be seen, there is a huge difference with real rule sets.

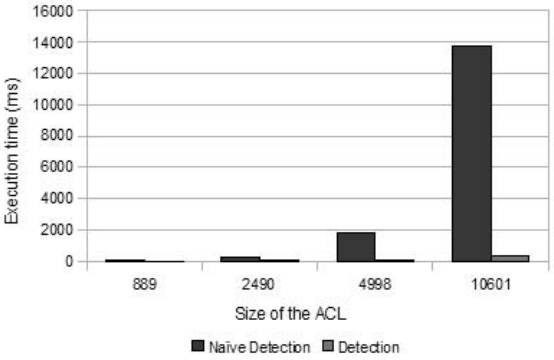

Figure 12. Comparison between isolation algorithms

In conclusion, the proposed detection and isolation algorithm represents a real improvement over the trivial one in real cases. In addition, due to the problem reduction due to the proposed consistency diagnosis process, exhaustive and optimal identification algorithms can be used over the diagnosis result (ICIRs). However note that when using heuristics for the diagnosis characterization, complexity would in general be dominated by the isolation algorithm. For that reason, we state that improvements in the isolation algorithm must be proposed in the future. It is also possible to use more complex heuristics if the final time fits performance requirements of the specific application of these algorithms. Due to its low computational complexity, the presented isolation algorithm can be used with very big rule sets or even in resource constrained devices [10] in real time.

IV. Related Works

One of the closest works to ours is related with consistency detection and isolation in general network filters. In the most recent work, Baboescu et al. [2] provides algorithms to detect inconsistencies in router filters that are 40 times faster than $O(n^2)$ ones for the general case of *k* selectors per rule, where *n* is the number of rules in the ACL. Although its algorithmic complexity is not given, it improves other previous works of isolation algorithms [7], [4]. However, they pre-process the ACL and convert selector ranges to prefixes. However, the range to prefix conversion technique could need to split a range in several prefixes [13] and thus the final number of rules could increase over the original ACL. In [14], Taylor outlines that this kind of conversion could be inefficient, because transport layer specifications vary widely (for example it is possible to specify open port ranges, such as *"all ports greater than 1023"*. Taylor also calculated that, in the worst case, a range covering w-bit port numbers may require $2(w-1)$ prefixes, and that a single ACL including only two port ranges could require $2(w-1)^2$ entries, or 900 entries (for 16-bit port numbers), raising the number of rules needed for the range to prefix conversion. Note that the range to prefix conversion is a very usual technique used in several matching algorithms. However, in diagnosis algorithms, this kind of techniques is not suitable. Thus, following Baboescu proposal, results are given over the pre-processed ACL, which is bigger and different that the original one.

Other researchers apply brute force, combinatorial algorithms to optimally solve the combined diagnosis and characterization problems. One of the most important advances was made by Al-Shaer et al. [1], where authors define an inconsistency model for firewall ACLs with 5 selectors. They give a combined algorithm to diagnose and characterize the inconsistencies between pairs of rules. In addition, they use rule decorrelation techniques [8] as a pre-process in order to decompose the ACL in a new, bigger, one with no overlapping rules. Results are given over the decorrelated ACL, which has the disadvantages commented for the Baboescu diagnosis algorithm. Although the proposed characterization algorithm proposed by Al-Shaer is polynomial, a decorrelation pre-process imposes a worst case exponential time and space complexity for the full process. In addition, their algorithms only characterize inconsistencies between pairs of rules, providing no composition as a later step to get a minimal characterization between more than two rules.

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. Due to the use of the same decorrelation techniques, this proposal also has worst case exponential complexity. The resulting ACL is also bigger and different from the original one. However, García-Alfaro et al. provides a characterization technique with multiple rules.

Ordered Binary Decision Diagrams (OBDDs) have been used in Fireman [15], where authors provide a diagnosis and characterization technique with multiple rules. A very important improvement over previous proposals is that they do not need to decorrelate the ACL, and thus, results are given over the original one. Note that the complexity of OBDD algorithms depends on the optimal ordering of its nodes, which is a NP-Complete problem [3]. This results in a worst case exponential time complexity with the number of rules, as other proposals.

There are several differences in our work with respect to the reviewed ones. In one hand, we provided an analysis of the consistency diagnosis problem in rule sets, separating the problem in three parts: detection and isolation, identification, and characterization. This enabled us to identify the performance bottlenecks of the problem, to reduce the combinatorial part

(characterization) to several smaller problems, and to design heuristic polynomial diagnosis algorithms for them if needed. The proposed diagnosis algorithms have a theoretical best case *O(n)* and worst case $O(n^2)$ time complexity with the number of rules in the rule set, *n*. More precisely, the complexity of our algorithms depends on the percentage of *allow* and *deny* rules over the total number of them (in the case of the isolation algorithm), on the cardinality of the diagnosis set, and finally, and on the number of inconsistencies (in the case of the identification algorithm). Our process is capable of handling full ranges in all selectors, and does not need to decorrelate or do any range to prefix conversion to the ACL as a pre-process to the algorithms. We think that for a result to be useful for a user, it should be given over the original ACL. However, our proposal does not cope with redundancies, because we think that redundancies are not a consistency problem.

V. CONCLUSIONS

We have deeply analyzed the consistency diagnosis problem in firewall ACLs, and decided to divide it in three sequential steps: detection and isolation, identification, and characterization. Detection and isolation plus identification is called diagnosis. All reviewed proposals deal with the full characterization problem with brute force algorithms, with yield unusable results (although optimal) for real-life, big rule sets.

In this paper we take a different approach, proposing the design of different, specialized, algorithms for each part of the diagnosis problem.

One of the main contributions has been a complete and abstract definition of inconsistency. Based on this definition, we revisited the consistency problem in firewall rule sets, demonstrating that all relations between more than two rules can be decomposed in pair wise relations.

The other major contribution of this paper is the proposal of two quadratic algorithms that should be applied sequentially to get a diagnosis of the inconsistent rules in the rule set. The first one detects and isolates the inconsistent rules, and is complete. The second one identifies the set of rules that cause the detected inconsistencies, and is complete but not minimal. The diagnosis can then be taken as input to optimal characterization algorithms resulting in an effective computational complexity reduction (solving several small combinatorial problems is faster than solving one big one), or to heuristic ones. The full process has best case *O(n)* and worst case $O(n^2)$ time complexity with the number of rules in the rule set, *n*. An experimental performance evaluation with real rule sets of different sizes was also presented, showing that real rule sets are very near to the best case, and the effective problem reduction.

However, our approach has some limitations that give us opportunities for improvement in future works. The most important one is that our process can diagnose inconsistent rules, but not redundant rules. Another direction to take in the future is the improvement of the isolation algorithm, since it dominates the complexity of the diagnosis process.

B. References

Number citations consecutively in square brackets [1]. Punctuation follows the bracket [2]. Use "Ref. [3]" or "Reference [3]" at the beginning of a sentence:

Give all authors' names; use "et al." if there are six authors or more. Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. In a paper title, capitalize the first word and all other words except for conjunctions, prepositions less than seven letters, and prepositional phrases.

For papers published in translated journals, first give the English citation, then the original foreign-language citation [6].

E. Equations

Equations should be centered in the column. The paragraph description of the line containing the equation should be set for 6 points before and 6 points after. Number equations consecutively with equation numbers in parentheses flush with the right margin, as in (1). Italicize Roman symbols for quantities and variables, but not Greek symbols. Punctuate equations with commas or periods when they are part of a sentence, as in

$$a + b = c . \tag{1}$$

Symbols in your equation should be defined before the equation appears or immediately following. Use "(1)," not "Eq. (1)" or "equation (1)," except at the beginning of a sentence: "Equation (1) is ..."

ACKNOWLEDGMENT

This work has been partially funded by Spanish *Ministry of* Science *and Education project* under grant DPI2006-15476-C02-01, and by FEDER (under ERDF Program). Many thanks to Pablo Neira Ayuso for providing us with real rule sets for testing, and to the anonymous reviewers for their useful comments.

REFERENCES

[1] Al-Shaer, E., Hamed, H. Modeling and Management of Firewall Policies". IEEE eTransactions on Network and Service Management (eTNSM) Vol.1, No.1, 2004.

[2] Baboescu, F., Varguese, G. "Fast and Scalable Conflict Detection for Packet Classifiers." Elsevier Computers Networks (42-6) (2003) 717-735.

[3] Bollig, B., Wegener, I. "Improving the Variable Ordering of OBDDs is NP-Complete". IEEE Transactions on Computers, Vol.45 No.9, September 1996.

[4] Eppstein, D., Muthukrishnan, S. "Internet Packet Filter Management and Rectangle Geometry." Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), January 2001.

[5] García-Alfaro, J., Boulahia-Cuppens, N., Cuppens, F. Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies, Springer-Verlag

International Journal of Information Security, Vol.7, No.2, 2008.
[6] Hamed, H., Al-Shaer, E. "Taxonomy of Conflicts in Network Security Policies." IEEE Communications Magazine Vol.44, No.3, 2006.
[7] Hari, B., Suri, S., Parulkar, G. "Detecting and Resolving Packet Filter Conflicts." Proceedings of IEEE INFOCOM, March 2000.
[8] Luis, S., Condell, M. "Security policy protocol." IETF Internet Draft IPSPSPP-01, 2002.
[9] Pozo, S., Ceballos, R., Gasca, R.M. "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.
[10] Pozo, S., Ceballos, R., Gasca, R.M. "Fast Algorithms for Local Inconsistency Detection in Firewall ACL Updates". 1st International Workshop on Dependability and Security in Complex and Critical Information Systems (DEPEND). Cap Esterel, France. IEEE Computer Society Press, 2008.
[11] Pozo, S., Ceballos, R., Gasca, R.M. "A Heuristic Polynomial Algorithm for Local Inconsistecy Diagnosis in Firewall Rule Sets". 3rd International Conference on Security and Cryptography (SECRYPT). Porto, Portugal. IEEE Computer Society Press, 2008.
[12] Pozo, S., Ceballos, R., Gasca, R.M. "Polynomial Heuristic Algorithms for Inconsistency Characterization in Firewall Rule Sets". 2nd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE). Cap Esterel, France. IEEE Computer Society Press, 2008.
[13] Srinivasan, V., Varguese, G, Suri, S., Waldvogel, M. "Fast and Scalable Layer Four Switching." Proceedings of the ACM SIGCOMM conference on Applications, Technologies, Architectures and Protocols for Computer Communication, Vancouver, British Columbia, Canada, ACM Press, 1998.
[14] Taylor, David E. Survey and taxonomy of packet classification techniques. ACM Computing Surveys, Vol. 37, No. 3, 2005. Pages 238 – 275.
[15] Yuan, L., Mai, J., Su, Z., Chen, H., Chuah,, C. Mohapatra, P. FIREMAN: A Toolkit for FIREwall Modelling and ANalysis. IEEE Symposium on Security and Privacy (S&P'06). Oakland, CA, USA. May 2006.
[16] Wool, A. A quantitative study of firewall configuration errors. IEEE Computer Journal, Vol.37, No.6, pp. 62-67, 2004.
[17] *iter's Handbook*. Mill Valley, CA: University Science, 1989.

S. Pozo holds an MSc in Computer Engineering from the University of Seville, in Spain, where he is a full-time Lecturer with the Computer Languages and Systems Department. He is currently a PhD candidate. He is part of the QUIVIR Research Group. His main research interests are computer and network dependability issues, models for security, and model-based diagnosis. More precisely, he is focused in firewall ACL languages and models, consistency/redundancy/conformance diagnosis in firewall ACLs, and the application of Model-Based Engineering paradigm to IT Security. He is also reviewer of computer security conferences and journals.

Rafael Ceballos holds an MSc in Computer Engineering from the University of Seville, in Spain, where he is a full-time Lecturer with the Computer Languages and Systems Department. He is currently a PhD candidate. He is part of the QUIVIR Research Group. His main research interests are software diagnosis, automatic debugging, constraint programming, design by contract, model-based diagnosis, testing, and IT security. More precisely, he is focused in diagnosing software using contracts, and in model-based diagnosis.

Rafael M. Gasca holds a PhD in Computer Science from the University of Seville, where he is a full-time Professor with the Computer Languages and Systems Department since 1991. He is the head of the QUIVIR Research Group, where has been the advisor of several fundamental research projects as well as applied RD projects in cooperation with the industry. His main research interests are information security, model-based diagnosis, semi-qualitative reasoning, constraint programming, and constraint databases. He is also reviewer and organizer of artificial intelligence and model-based diagnosis conferences and journals. Finally, he provides advising in information security to private companies.