Fast Algorithms for Consistency-Based Diagnosis of Firewall Rule Sets

S. Pozo, R. Ceballos, R. M. Gasca
Department of Computer Languages and Systems
ETS Ingeniería Informática, University of Seville
Avda. Reina Mercedes S/N, 41012 Sevilla, Spain
sergiopozo@us.es {ceballos,gasca}@lsi.us.es
Tel:+34 954559897 Fax: +34 954557139
http://www..lsi.us.es/~quivir

Abstract

Firewalls provide the first line of defence of nearly all networked institutions today. However, Firewall ACL management suffer some problems that need to be addressed in order to be effective. The most studied one is rule set consistency. There is an inconsistency if different actions can be taken on the same traffic, depending on the ordering of the rules. In this paper a new algorithm to diagnose inconsistencies in firewall rule sets is presented. Although many algorithms have been proposed to address this problem, the presented one is a big improvement over them, due to its low algorithmic and memory complexity, even in worst case. In addition, there is no need to pre-process in any way the rule set previous to the application of the algorithms. We also present experimental results with real rule sets that validate our proposal.

Keywords: *diagnosis, consistency, conflict, firewall, acl, rule set*

1. Introduction

A firewall is a network element that controls the traversal of packets across different network segments [1, 2], thus it is a mechanism to enforce an Access Control Policy, represented as an Access Control List (ACL). An ACL is in general a list of linearly ordered (total order) condition/action rules. A rule is defined as follows (Equation 1):

$$\forall i, 1 \le i \le n, R_i : \{condition_i\} \Rightarrow \{action_i\}$$

Equation 1

where *i* is the position of the rule in the ACL (or its priority) and *n* is the position of the last rule. ACL can be forward or backward checked, but in firewalls the most common method is forward checking. The *condition* part of the rule is a conjunctive set $S_1 \wedge S_2 \wedge ... \wedge S_k$ of condition attributes or selectors, where *k* is the number of selectors. The *condition* set is typically composed of five elements, which correspond to five fields of a packet header [3]: Source IP, Destination IP, Source port, Destination port, Protocol. 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 of the values of its corresponding rule selector. Firewalls implement ACL using its own low-level language, forming which is commonly denominated a *rule set*.

The *action* part of the rule represents the action that should be taken for a matching packet. In firewalls, two actions are possible: accept or deny a packet. An example of a rule set which has been taken from [4] is presented in Table 1.

Although deployment of firewalls is an important step in the course of securing networks, complexity of designing and maintaining firewall rule sets might limit the effectiveness of firewall security [7]. Firewalls have to face many problems in modern networks. One of the most important ones is rule set consistency. As can be seen from the example in Table 1, selectors of rules can overlap (for example, the protocol selector), and can even be rules that are totally equal to others. There is an inconsistency when two or more rules with different actions overlap. Since a packet can be matched with any of the overlapping rules, firewalls use a positional conflict resolution technique, taking the action of the first matching rule. Rule sets are usually composed of a number of rules ranging from tens to five thousand [3]. Algorithms and tools to isolate, identify and characterize inconsistencies must be provided in order to analyze firewall rule sets.

Priority/ID	Protocol	Source IP	Src Port	Destination IP	Dst Port	Action
R1	tcp	140.192.37.20	any	*.*.*.*	80	deny
R2	tcp	140.192.37.*	any	*.*.*.*	80	accept
R3	tcp	*.*.*.*	any	161.120.33.40	80	accept
R4	tcp	140.192.37.*	any	161.120.33.40	80	deny
R5	tcp	140.192.37.30	any	*.*.*.*	21	deny
R6	tcp	140.192.37.*	any	*.*.*.*	21	accept
R7	tcp	140.192.37.*	any	161.120.33.40	21	accept
R8	tcp	*.*.*.*	any	*.*.*.*	any	deny
R9	udp	140.192.37.*	any	161.120.33.40	53	accept
R10	udp	*.*.*.*	any	161.120.33.40	53	accept
R11	udp	140.192.38.*	any	161.120.35.*	any	accept
R12	udp	*.*.*.*	any	*.*.*.*	any	deny

Table 1. Example of a firewall rule set

Although deployment of firewalls is an important step in the course of securing networks, complexity of designing and maintaining firewall rule sets might limit the effectiveness of firewall security [7]. Firewalls have to face many problems in modern networks. One of the most important ones is rule set consistency. As can be seen from the example in Table 1, selectors of rules can overlap (for example, the protocol selector), and can even be rules that are totally equal to others. There is an inconsistency when two or more rules with different actions overlap. Since a packet can be matched with any of the overlapping rules, firewalls use a positional conflict resolution technique, taking the action of the first matching rule. Rule sets are usually composed of a number of rules ranging from tens to five thousand [3]. Algorithms and tools to isolate, identify and characterize inconsistencies must be provided in order to analyze firewall rule sets.

In this paper we define what is an inconsistency between an arbitrary number of rules in a firewall rule set. We propose to divide consistency management in two sequential phases: isolation and identification (diagnosis) of inconsistent rules, and characterization of the diagnosis. Finally, we propose a worst case $O(kn^2)$ time complexity process with the number of selectors *(k)* and rules *(n)*, to automatically diagnose the inconsistent rules in a rule set, without doing rule decorrelation or any other pre-process, and give two algorithms that implement the diagnosis process. The characterization part has been left out as a topic for future research. A Java tool is available upon request.

We consider this work a significant advance in consistency diagnosis in firewall rule sets, because to the best of our knowledge, this is the first time a process with such low algorithmic complexity has been proposed to automatically address this problem.

The paper is structured as follows. In section 2 we review related works comparing them to our proposal. In section 3, we analyze the internals of the consistency problem in firewall rule sets. In section 4 we propose the consistency-based diagnosis process, which is based in two algorithms. Finally, we give some concluding remarks in section 5.

2. Related works

The closest works to ours come from the firewall diagnosis field. One of the most important advances has been made by Al-Saher et al. [4, 12]. In their works, authors define an inconsistency model for firewall rule sets, and they provide an order-dependent characterization of different kinds of inconsistencies by pairs of rules. They give a combined algorithm to diagnose and characterize inconsistencies between every pair of rules. However, their algorithms do not diagnose and characterize inconsistencies with a combination of more than two rules. In addition, they use rule decorrelation techniques as a pre-process in order to decompose the rule set in a new one with non overlapping rules, which is then used as input to their algorithms. Since the proposed rule decorrelation algorithms [8] have worst case exponential time and space complexity, the worst case complexity of their process is also exponential.

A modification to their algorithms was provided by Cuppens et al. [13, 5], where they integrate the decorrelation and consistency diagnosis 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 has the same complexity problem of Al-Saher works. Cuppens et al. provide a characterization technique with multiple rules, instead of a two-by-two characterization technique of Al-Shaer. The output of their process is the inconsistency characterization (but not the rules that cause the inconsistency), and a consistent rule set. The consistent rule set is obtained by automatically removing the inconsistent rules

without asking the user, which could give a very dangerous result, since the resulting rule set could not conform to the initial user intentions [11]. Finally, it is difficult to compare their algorithms with Al-Shaer ones, since Al-Shaer performance analysis does not include the time required for decorrelation.

Others have tried to address the consistency problem using OBDDs, as is the case of Fireman [9], but since complexity of OBDDs depends on the ordering of its nodes, and the optimal ordering of nodes is a NP-Complete problem [6], they also suffer of the same complexity problem of the other reviewed works.

The main difference of these works with ours is that we propose an order-independent consistency-based diagnosis process. The problem analysis enabled us to dissociate the consistency diagnosis phase from the conflict characterization one, obtaining a process that is significantly faster and require less memory than other existing proposals. Our algorithms have polynomial worst case time complexity in $O(kn^2)$ with the number of rules in the rule set *(n)* and with the number of selectors *(k)* in the decision part of the rule, and linear space complexity with the number of inconsistent rules.

In addition, our process does not need to decorrelate the rule set as a previous process to the analysis, and does not use decorrelation at all in any step of the process: it treats rules as they are provided in the rule set. The use of decorrelation is the main reason of the higher complexity of the algorithms given by Al-Shaer [4] and Cuppens [13].

However, our process does not cope with redundancies, because redundancy is not a consistency problem (it does not change the semantics of the rule set). We do neither cope with conflict characterization, and have left this topic for future research. The reason is that it needs a deep analysis, since our approach is completely different from the taken by other authors, that due to space constraints, it is not possible to include in this paper. Diagnosis characterization has been left out as a topic for future research.

3. Analysis of the consistency problem

Before the definition of inconsistency is given, it is important to review the inconsistencies characterized in the bibliography. A complete characterization has been given by Al-Shaer et al. [12] that include shadowing, generalization, correlation and redundancy (all of them except correlation are order-dependent). Recall that redundancy will not be considered in this work.

All research works in this area follow their definitions in one or another way.

All of these inconsistencies except redundancy are graphically represented in Fig. 1. For the sake of simplicity and due to space constraints, only one example with two-rule inconsistencies with one selector has been represented. The extension of this example to *n* rules with 5 selectors each can easily be done with a 5-face hypercube or *penteract* [14].

Attending to the characterization, two rules (R_x, R_y) are correlated if they have a relation between all of its selectors, and have different actions. Fig. 1(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. Fig. 1(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). Fig. 1(b) represents a generalization inconsistency between two rules (inverse of shadowing respect to the priority). The relation is superset of the general rule respect to the other one.

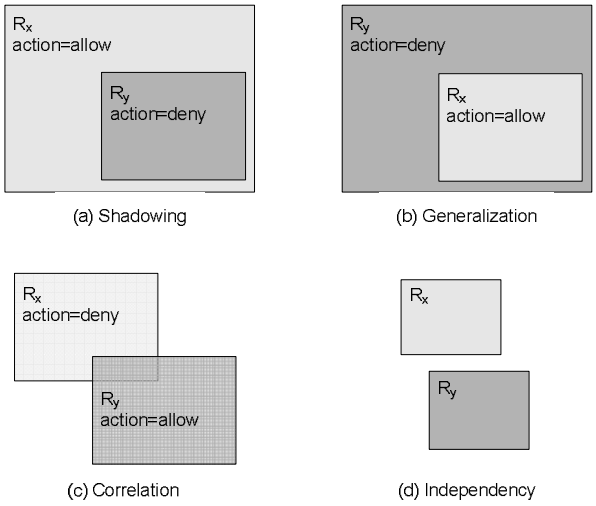

Figure 1. Graphical representation of three types of inconsistencies in rule sets

In a closer look at shadowing and generalization inconsistencies in Fig. 1, 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, we propose to have only one o priority-independent inconsistency definition in order to simplify the diagnosis process. This approach postpones characterization to a later step in order to simplify the diagnosis process.

Definition 1. Inconsistency. Two rules are *inconsistent* if the intersection of all of its selectors is not empty, and they have different actions, *independently of the priority* they have. The inconsistency between two rules expresses the *possibility* of a non desirable effect in the *semantics* of the rule set. The semantics of the rule set could change if an inconsistent rule is removed.

This definition can be extended to more than two rules. Attending to Definition 1, all cases represented in Fig. 1 are of the same kind, and are called *inconsistencies* without any particular characterization. Priority is only required if inconsistencies are going to be characterized, which is not the focus of this paper.

If a rule set of n rules is consistent, the addition of a new rule R_z cannot cause an inconsistency between any of the consistent rules, but it can cause an inconsistency with itself and with one to n of the consistent ones. In the same manner, if a new rule is added to an inconsistent rule set, this new rule cannot create an existing inconsistency between existent rules, nor eliminate it. It can only create new and independent inconsistencies, or be an independent (consistent) rule.

Also, because of the symmetry introduced in Definition 1 (inconsistencies are diagnosed with independence of the priorities), once a combination of two rules is tested in the diagnosis process, the inverse must not. That is, if R_x is inconsistent with R_y, then R_y is also inconsistent with R_x. Thus, inconsistencies are commutative.

As this analysis is extensible to an arbitrary number of rules, inconsistency definition can isolate and identify all possible inconsistencies between any number of rules, since any problem can be decomposed in two by two relations.

These properties introduce a huge reduction in time complexity (of several orders of magnitude) to the diagnosis process compared to other approaches taken in the bibliography.

4. Consistency-based diagnosis of firewall rule sets

The diagnosis process explained in this section is divided in two steps. In the first step, all inconsistencies are isolated between every pair of rules in an order-independent process. In the second step, the minimum number of conflicting rules is identified, completing the consistency-based diagnosis process. The full process, as will be showed, requires no modification to the original rule set, or any pre-process or rule decorrelation. The algorithms run in worst-case quadratic time and have lineal space complexity with the number of rules in the rule set.

4.1 Step 1. Isolation of inconsistent pairs of rules

The first step of the process isolates the potentially conflicting rules of the complete rule set and forms a graph representing their relations. The algorithm presented in Figure 2(a) receives a firewall rule set and for each rule, it checks if there is an inconsistency with any of the previous ones in the rule set, but not with the following ones. The reason to only check if there is an inconsistency with the previous rules and no with the following ones is due to the commutative property of inconsistency: in later iterations, rules that go after the considered one are going to be tested against the preceding ones.

Each time the algorithm finds an inconsistency between a pair of rules, the two rules are added as vertices to a graph, with a non directed edge between them. The algorithm returns a graph called Potential Conflict Graph, or PCG. All possibilities have been checked, as the inconsistency definition recognizes all possible cases that generate inconsistencies, thus the algorithm is complete.

```
Func buildPCG(in List: ruleset; out Graph: pcg)
Alg
   for each i=1..ruleset.size() {
      for each j=i+1..ruleset.size() {
         Vertex origin=uleset.get(i)
         Vertex destination=ruleset.get(j)
         if inconsistency(origin, destination) {
            // if vertices or edge are in the graph, nothing is done
            pcg.addVertex(origin)
            pcg.addVertex(destination)
            pcg.addEdge(origin, destination)
         }
      }
   }
End Alg


Func inconsistency(in Rule: Rx, Ry; out Boolean: b)
Alg
   b = false
   if (Rx.action != Ry.action) {
      b = true
      for each i=1..Rx.selectors.size()
         b = b AND intersection(Rx.selector.get(i), Ry.selector.get(i))
   }
End Alg
```

Figure 2(a) Algorithm 1. Inconsistency isolation

The vertices of the PCG represent the rules that are inconsistent with other ones, and the undirected edges their relations. The PCG can be characterized in a particular kind of graph: its edges are undirected because of the commutative property of inconsistency; it can have cycles depending of the relations between the inconsistent rules (if they are correlated, it would be cycles); and finally, it can be unconnected if there are inconsistent rules which do not have a relation with other inconsistent rules. Fig. 3 represents the resulting PCG obtained from the application of Algorithm 1 to the example of Table 1.

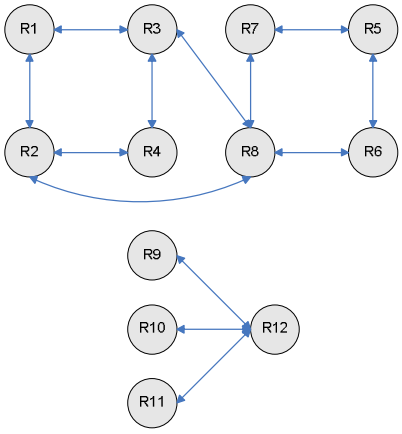

Figure 3. Algorithm 1 result: Potential Conflicts Graph, PCG

4.2. Step 2. Identification of the Set of Conflicting Rules

The second and last step of the process identifies the minimal set of rules that generate inconsistencies with another rule, or the diagnosis set.

Algorithm 2 (Fig. 2(b)) takes the PCG as input. The algorithm takes iteratively the vertex with the greatest number of adjacencies, that is, the vertex with the greatest number of inconsistencies. If there are vertices with the same number of adjacencies, the relative ordering between them does not care. Then, this vertex and its edges are removed from the PCG, generating an independent cluster of inconsistent rules (ICIR) as a tree with the removed vertex (the conflicting rule of the cluster) as its root, and the inconsistent rules as its leafs. If vertices with no edges are left in the PCG, then these vertices are removed, since they are consistent by definition (they do not have no more relations with other rules of the PCG). Note that, as inconsistencies have been decomposed in pair wise relations, the tree is always formed by two levels.

```
Func buildICIRs(in Graph: pcg; out List: icirs)
Var
   Tree icir
Alg
   while (pcg.hasVertices()) {
      vertex v = pcg.getMaxAdjacencyVertex()
      List adj = pcg.getAdjacents(v)
      pcg.removeVertexWithEdges(v)
      pcg.removeUnconnectedVertices()
      icir.createEmptyTree()
      icir.setRoot(v)
      icir.addChilds(adj)
      icirs.add(icir)
   }
End Alg
```

Figure 2(b) Algorithm 2. Determination of the Set of Conflicting Rules

A graphical representation of a partial trace of Algorithm 2 over the previous PCG is presented in Fig. 4. At the first iteration, R8 was selected because it had four inconsistencies (the greatest number of adjacent vertices). It had been removed and formed the first ICIR tree, with R8 as root, thus R8 is a conflicting rule. In the second iteration, R12 was selected because it has three inconsistencies (it is the vertex with the biggest number of adjacent vertices). Then it was removed and the second ICIR is formed. Vertices R9, R10 and R11 were removed from the PCG because they had no adjacent vertices. In the third iteration, there is a possibility of selecting R5, R1, R2, R3 and R4 as the next vertex. The selection of one or other is arbitrary. In this example, the algorithm selected R5, removed it from the PCG with all its edges and formed the third ICIR. At the end of this iteration the PCG was only

composed of a cycle of four vertices: R1, R2, R3, and R4. The algorithm selected to remove R1 in the fourth iteration and R4 in the fifth and last iteration, removing the vertices and edges, and forming ICIR 4 and ICIR 5 respectively. Since no more vertices are left in the PCG, the algorithm finished with a Set of Conflicting Rules (SCR) of cardinality five, containing the rules SCR={R8, R12, R5, R1, R4}. These rules are ICIR roots, or the minimal number of rules that cause an inconsistency with others.

Since the ICIR are small enough to be analyzed by a human even in large rule sets, and are independent clusters of inconsistencies, there is no real need to characterize them. However, using existing taxonomies [12], it is possible to automatically characterize each inconsistency. For example, in ICIR5={root=R5, R7, R6}, R6 is a generalization of R5, and R7 is correlated with R5. As has been noted before, characterization is a topic for future research.

4.3. Algorithmic complexity. Empirical results

Time complexity of Algorithm 1 is linear with the number of selectors of rules (*inconsistency()* method), k, and quadratic with the number of rules of the rule set (iteration), n. Therefore, worst case time complexity is in $O(k(n^2-n))=O(kn^2)$. In a typical rule set with 5 selectors per rule, $k=5$.

Worst case time complexity of Algorithm 2 is $O(h*n)$ with the number of conflicting rules or ICIRs (iteration), h, and with the number of vertices in the graph (minimum calculation), v (in worst case, $v=n$)

Space complexity of Algorithm 1 is linear with the number of rules in the rule set. In the presented example, there are 12 inconsistencies (Fig. 6), forming a graph with 12 vertices and 13 edges (Fig. 7). Note that the number of edges is the same that the number of inconsistent rules.

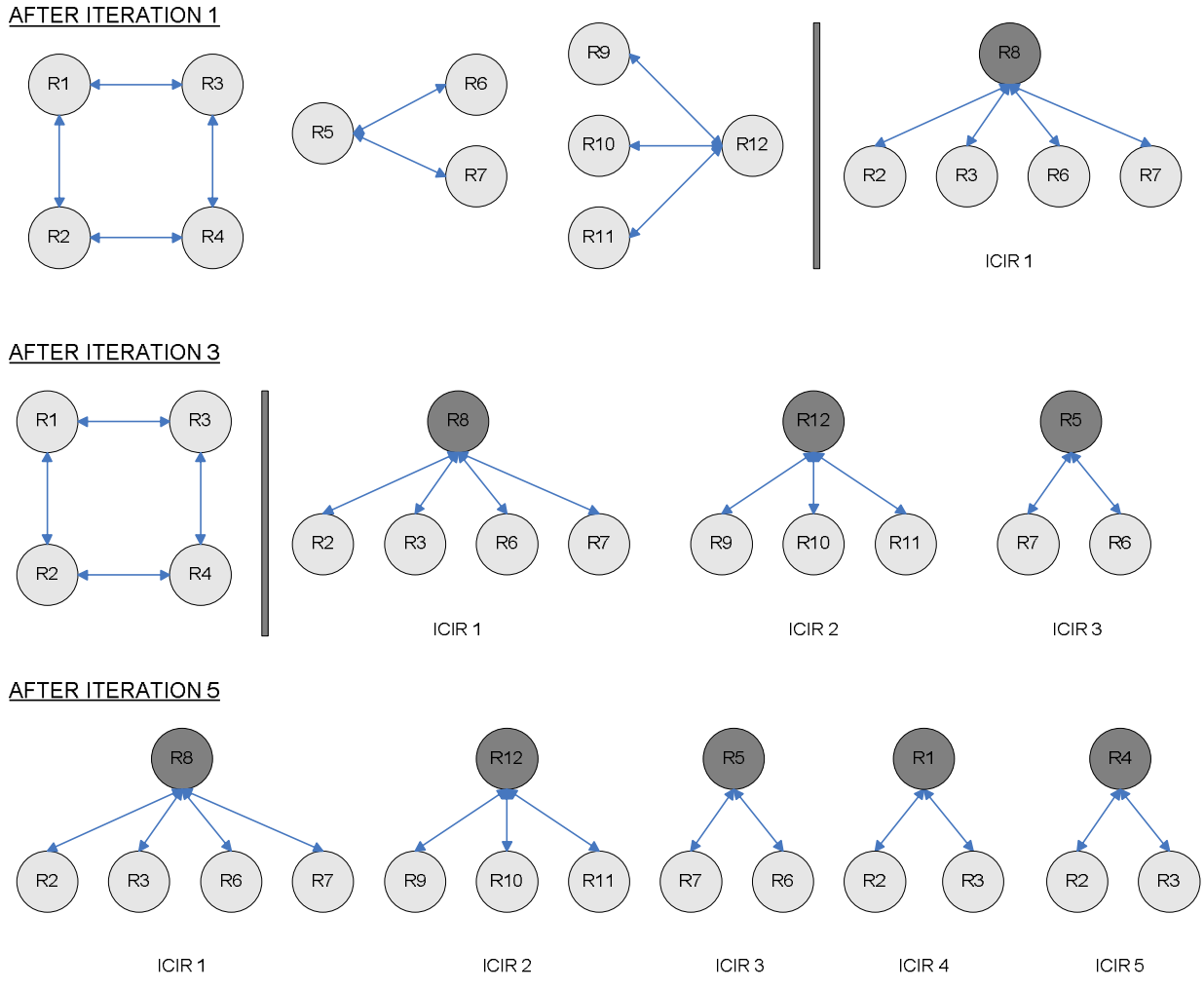

Figure 4. Trace of Algorithm 2 applied to Fig. 3 PCG

Algorithm 2 also needs some space to represent all ICIR. The number of ICIR increase linearly with the number of inconsistent rules. But note that, as the algorithm is creating the ICIR, the corresponding vertices and edges are removed from the PCG, so only small amount of additional temporal space is necessary over the space needed for the PCG, in order to represent the duplicate vertices that are left in the PCG because they already have inconsistencies with other vertices. For example, in the iteration one of Fig. 8, ICIR 1 has the vertices {R2, R3, R6, R7}, that will be removed in later iterations.

Therefore the combined worst case complexity of the full process is the maximum between them, $O(kn^2)$ time complexity and $O(n)$ space complexity. This complexity analysis show that presented solution is, to the best of our knowledge, the most efficient solution to this problem in time and space, even for worst case.

Comparing the results of the presented algorithms with Al-Shaer et al. ones [4], the presented algorithms diagnose one more inconsistency, which is the one between rules R11 and R12.

Applying the algorithms to the example presented by Cuppens et al. [13] the generated ICIRs are ICIR1={root=R4, R3, R2, R5} and ICIR2={root=R1, R2, R5}. Again, the presented algorithms can diagnose more inconsistencies than the ones proposed by them.

In order to empirically compare the efficiency of the proposed algorithms, we have developed a preliminary characterization step based on a work in progress, applying the characterization described by Al-Shaer et al. [4], but with the improvement of being able to characterize inconsistencies caused by the union of rules. This step is not described in this paper due to space constraints. However, to make more realistic the comparison, we have measured the time of the proposed algorithms plus the characterization step. Take into account that the characterization algorithms are a polynomial approximation to the optimum. Also note that the full process time is dominated by the Algorithm 1 complexity.

As real rule sets have been used, Fig. 5 and Table 2 represent an average case. Tests have been run with and without wildcard rules (WR, *deny all* and *allow all* rules), in order to know the impact these rules have in the complexity of the algorithms. Results show that time is under 350ms until rule set size reaches 900 (used in Fireman [9]). For rule sets about the sizes used by Al-Shaer and Cuppens examples, that is between zero and 80 rules, time is between zero (zero rules) and 0,6 milliseconds (80 rules), and between zero and 0,35ms if wildcard rules are removed. Experiments were performed on a monothreaded Java implementation with JVM 1.6.0_02, and on a machine with Intel Core Duo T2400 and 1Gb RAM DDR667.

Empirical results show that the presented proposal is, to the best of our knowledge, orders of magnitude faster than other published ones, and requires less memory.

Size	TOTAL w/ WR (ms)	#Removed WR	TOTAL w/o WR (ms)
50	0,37	4	0,26
144	3,57	4	2,55
238	13,47	10	7,83
450	51,75	10	24,43
900	318,28	11	89,64

Table 2. Performance evaluation

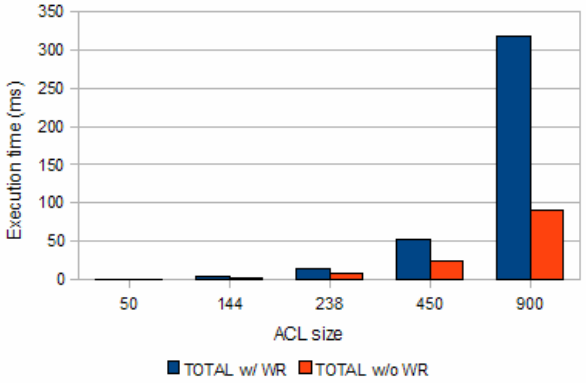

Figure 5. Performance evaluation

5. Conclusions and future works

In this paper, we have analyzed the consistency diagnosis problem in firewall rule sets, and decided to divide diagnosis and characterization in two different steps. We explained that redundancy is not an inconsistency, because it does not change the semantics of the rule sets, and focused our work on the relations that can change the semantics of the rule set. We have proposed an abstract definition of inconsistency that covers all cases characterized in the bibliography. Based on this definition, we revisited the consistency problem in firewall rule sets and showed that all relations between more than two rules can be decomposed in simpler pair wise relations, without decorrelating the rule set.

In order to validate our results, we have designed two algorithms that should be applied sequentially. The first one creates a graph that represents all potential inconsistencies between rules. This graph responds to the inconsistency isolation process. In the second one the graph is used to construct independent clusters of inconsistent rules, and represent each cluster as a two level tree. The root of the tree was defined as a conflicting rule of the rule set. The set formed by the roots of all ICIR form the minimal set of conflicting rules, diagnosis set, or the set rules that can be removed from the rule set in order to get a consistent

one. Isolation and identification conforms the full consistency-based diagnosis process.

We compared our results with other works in the bibliography, and showed that our algorithms can diagnose more cases. A time and space complexity analysis was done and showed that the full process has worst case $O(kn^2)$ time complexity with the number of rules in the rule set *(n)* and with the number of selectors in the condition part of the rule *(k)*. Space complexity is linear with the number of rules in the rule set. A basic performance evaluation was also presented. To the best of our knowledge, this is the most efficient solution in time and space to address this problem.

However, our approach has some limitations that give us opportunities for improvement in future works. The most important one is that our proposal cannot diagnose redundancies, although they are not considered inconsistencies in this paper. Proposals made by other researchers, although not diagnosing all inconsistencies and having worst time and space complexity, can diagnose redundant rules.

Although the process presented in this paper can diagnose inconsistencies, it cannot characterize them. This is another main topic for future research.

Acknowledgements

This work has been funded by the Spanish *Ministerio de Educación y Ciencia* under grant DPI2006-15476-C02-01. Many thanks to Pablo Neira Ayuso for providing us with real rule sets for testing.

References

[1] D. Chapman and E. Zwicky. Building Internet Firewalls, Second Edition, O'Reilly & Associates, Inc., 2000.

[2] W. Cheswick and S. Belovin. Firewalls and Internet Security, Second Edition, Addison-Wesley, 2003.

[3] David E. Taylor. Survey and taxonomy of packet classification techniques. ACM Computing Surveys, Vol. 37, No. 3, 2005. Pages 238 – 275.

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

[7] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62-67, 2004.

[8] S. Luis, M. Condell. "Security policy protocol." IETF Internet Draft IPSPSPP-01, 2002.

[9] 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.

[10] A. Mayer, A. Wool, E. Ziskind. Offline Firewall Analysis. International Journal of Information Security, Vol. 5, No. 3, Pages 125-144. Springer-Verlag, 2005.

[11] S. Pozo, R. Ceballos, R. M. Gasca. "CSP-based Firewall Rule Set Diagnosis using Security Policies." International Symposium on Frontiers in Availability, Reliability and Security (FARES), in International Conference on Availability, Reliability and Security (ARES), Vienna, Austria. IEEE Computer Society Press, April 2007.

[12] H. Hamed, E. Al-Shaer "Taxonomy of Conflicts in Network Security Policies." IEEE Communications Magazine Vol.44, No.3, 2006.

[13] F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro. "Detection and Removal of Firewall Misconfiguration." IASTED International Conference on Communication, Network and Information Security (CCNIS), Phoenix, AZ, USA. ACTA Press, 2005. 0-88986-537-X

[14] Coxeter, H.S.M. Regular Polytopes, (3rd edition, 1973), Dover edition, ISBN 0-486-61480-8 p.296, Table I (iii): Regular Polytopes, three regular polytopes in n-dimensions (n>=5)