CSP-Based Firewall Rule Set Diagnosis using Security Policies

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
{sergio,ceballos,gasca}@lsi.us.es
http://www..lsi.us.es/~quivir

Abstract

The most important part of a firewall configuration process is the implementation of a security policy by a security administrator. However, this security policy is not designed by higher levels of the organisation, nor is written anywhere, so it is very usual to make mistakes in its implementation. To solve this problem we propose to express this global access control policy in some informal language that is translated to a model specification in conjunction with the firewall rule set. Then we construct a Constraint Satisfaction Problem to detect and identify the possible inconsistencies between the specified policy and the firewall rule set.

Keywords: *firewall, policy, constraint, csp, consistency*

1. Introduction

At present days, firewalls continue to be the first line of defence of most (if not all) organizations that are connected to the Internet. However, to be an effective defence, a firewall must be configured properly. Firewall rule sets usually are not commented nor described anywhere: if there is no written security policy stating what the firewall must do, the implementation depends entirely on the interpretation of the security policy by security administrators.

There are several reasons these tasks are difficult and prone to errors. Firewall rule languages tend to be very low level and difficult to write, and usually require an in-depth knowledge of the specific firewall system. Rule modification could be a daunting task, because a modification of a rule in a firewall with hundred of rules could derive side effects in its own firewall, or at other firewalls linked with it. In addition, most organisations don't have a security policy for their everyday aspects of security, even less a more specific policy like an access control one for implementation at their firewalls.

We firmly believe in the need of the use of a model and formal methods to reason about firewall rule sets in order to detect and identify faults. We also think that the effectiveness of a firewall is dependent on providing a written security policy (a specification of what must be done) given by people at higher levels of the organization than systems' administrators.

In this paper we propose a first order logic model in which to transform both a natural language specification of an access control policy, and a firewall rule set. We also propose a method (we have implemented) based on Constraint Satisfaction techniques to automatically detect and identify inconsistencies between the rule set and the specified policy (Fig. 1). The high level specification language to specify the security policy is not an objective of this work.

The rest of the paper is organized as follows. In section two we comment the state of the art on the related works and explain the main differences with our approach. In section there is a very brief introduction to constraint satisfaction problems. Section four is the core of the paper, we explain or model in an incremental way. In section five there is an illustrative example. Finally, in section six we conclude and give some thoughts about future works.

2. Related works

The closest works to ours come from the firewall analysis field. In [4, 5], authors use graph algorithms to represent firewall access control policies and reason about packet trajectories. Authors of [6, 7] provide an abstract language to represent policies, and provide a mechanism to completely separate the security policy from the network topology. There is also possible to query this abstract representation (which is also based

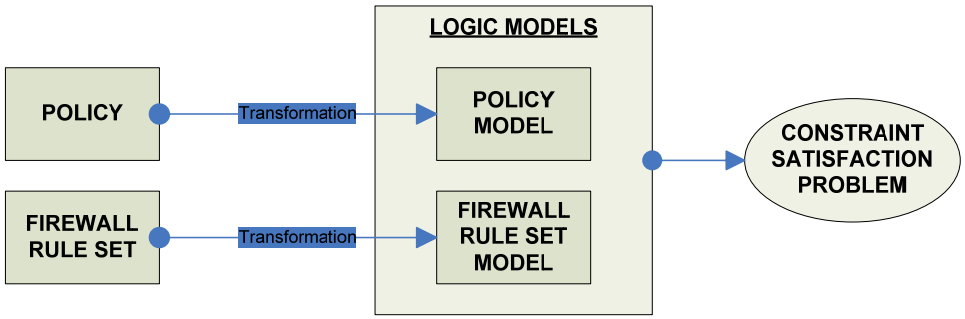

Figure 1. High level view of the firewall diagnosis process

on graph algorithms) about concrete paths, or even to generate all possible queries and present them to the user, which is responsible of deciding if the firewall works as expected. Finally, the abstract representation can be compiled to low level firewall rules. In [8] authors use constraint logic programming to represent policies transformed from low level firewall rules. They are also able to reason about the policies, but queries must be given by experts. In [9] authors present a method based on graph algorithms to reason about IDS configurations in combination with a firewall configuration of the network.

The main difference of all of these works with ours is that we provide a specified high level policy. This policy is used as a goal to test if the firewall complies with it. This approach is a recommendation suggested in [4, 5], and a logical consequence of the works of [6, 7] which query all possible things about the firewall and then present the results to the user; and of [8], in which the user is responsible of querying the reasoning engine. Another significant difference is that we provide a very simple but strong logic model of firewalls, and also a strong reasoning method based on Constraint Satisfaction Problems. To the best of our knowledge, this is the first approach to model a firewall as a logic machine and transform this model into a CSP.

3. Constraint satisfaction problems

Constraint Satisfaction Problems have been object of research in Artificial Intelligence in the last few decades [1]. A Constraint Satisfaction Problem (CSP) is a framework for modelling and solving real-problems as a set of constraints among variables defining the state of the problem. A CSP is defined as a set of variables $X = \{X_1, X_2..., X_n\}$, each associated with a discrete-valued domain $D = \{D_1, D_2..., D_n\}$, and a set of constraints $C = \{C_1, C_2...C_m\}$ restricting the values that variables can take simultaneously (reducing the domain for each variable). Each constraint C_i is a pair (W_i, R_i), where R_i is a relation $R_i \subseteq D_{i1} \times ... \times D_{ik}$ defined in a subset of variables $W_i \subseteq X$. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously.

A CSP could have multiple solutions. It is possible to find (1) just one solution with no preference to which one of the total solution space, (2) all solutions, and (3) an optimal solution by means of an objective function defined in terms of one or more variables. There are many techniques to search and prune through all possible value assignments to variables.

In many real-life applications, it is possible to be interested in a good solution, and not in whichever solution. The objective function usually is a function that tries to maximize (or minimize) the number of satisfied constraints. Such kinds of problems are referred as MaxCSP, or more generally, Constraint Satisfaction Optimization Problems (CSOP) [1].

4. System model

4.1 Policy and Firewall Rule Set Transformation Rules

A typical rule from a firewall has different fields that represent information relevant to the filtering and/or logging tasks. The fields needed to be modeled are:

- Source and Destination. The Source IP field of the packet represents one possible IP of the network address space at which the source machine is connected to, but can also represent a subnet, or the entire zone. Destination IP field is analogous.

- Service and Protocol. The protocol represents which of the many existent protocols (TCP, UDP, ICMP, etc.) the service uses. The service represents the destination port number at which the service is located. For simplicity, we will only use the port number.
- Action. The action represents if the firewall leave the packet pass to its destination (allow) or not (deny).

Note that the direction of the packet (that is, which interface does it come from and where does it go to) is implicit into the Source and Destination fields, thus it is not necessary to model the packet directions or interfaces.

In our model it is also possible (and preferred) to use symbolic names instead of absolute numbers for all the fields, because it is very usual to use symbolic names instead of concrete network address: with independence of the low level network topology, the high level access control goals should be the same. The main benefit of this process is the separation of the real network topology from the policy and rule set models. A policy or rule in our model is therefore represented as a four field tuple (Fig. 2)

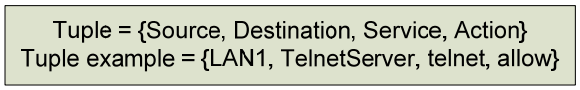

Figure 2. Tuple representation

We are going to use the same model for both the policy and firewall rules because they are equivalent (they express the same thing with the same information fields).

As noted by the action field in the tuple, our model can express both allow and deny policies and rules. For the sake of simplicity, policies can be expressed only with allow action fields, so order is not important; but rule sets can be expressed with deny action fields, so order is important; also Internet must be expressed as an explicit zone. We assume that the last policy and the last rule are general deny all ones.

4.1.1 Policy Transformation from Natural Language Specification. As we have noted before, the main contribution of this paper is the use of a high level policy to express the global access control requirements of the organisation. The firewall rule set must comply with this policy. To be able to reason about both the policy and the rule set, it is necessary to model them into tuples. The tuple model is the same for both the policy and rule set.

The transformation of a natural language policy to a tuple of our model is very simple (Fig. 3). Note the use of symbolic names.

4.1.2 Rule Transformation from Firewall Rule-Set. The transformation from rules is even more direct, since the fields in them are the same of the tuples. For example, an IP Tables [2] rule can also be transformed to a tuple (Fig. 4).

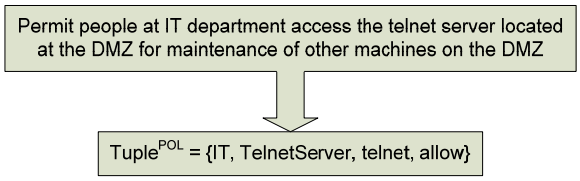

Figure 3. Tuple transformation from natural language specification

Note that the transformation from a rule to a tuple is direct and can be automated parsing firewall rule fields. Note that, as we have indicated before, we are not taking into account the protocol (TCP in this example) for the sake of simplicity, but its incorporation into the model is direct.

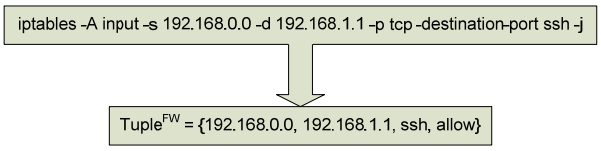

Figure 4. Tuple transformation from low level firewall rule

4.2 Firewall Model

Fortunately, a firewall is a very simple machine whose working internals are well known. Fig. 5 represents a logic model for rule processing in a firewall.

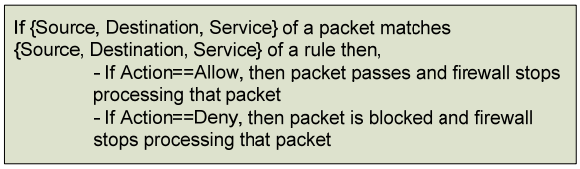

Figure 5. Firewall rule processing model

This process should be repeated until there is a match between a rule and a packet. If there is no match with the specified rules, then the default action is

executed (in our model the default action is always a deny all rule).

In our proposal, there is a high level policy specifying the access control policy. For our purposes it is necessary to model both the policy and the rules.

4.2.1 Logic model. As we are going to use a more formal language in this subsection, it is also necessary to give some definitions to model the tuples and the complete policy and rule set.

The variables $Source^{POL}(i)_{0<i\le n}$, $Source^{FW}(j)_{0<j\le m}$ represent the source field of a tuple representing a policy and a tuple representing a firewall rule respectively. In a similar way, the variables $Destination^{POL}(i)_{0<i\le n}$, and $Destination^{FW}(j)_{0<j\le m}$ represent the destination field of a tuple representing a policy and a rule respectively. Variables $Service^{POL}(i)_{0<i\le n}$ and $Service^{FW}(j)_{0<j\le m}$ represent the service field of a tuple representing a policy and a rule respectively. Finally variables $Action^{POL}(i)_{0<i\le n}$, and $Action^{FW}(j)_{0<j\le m}$ represent the action field of the tuple representing a policy and a rule respectively.

A policy is a finite set of tuples of the form:

$Tuple^{POL}(i)_{0<i\le n} =$

$\left\{Source^{POL}, Destination^{POL}, Service^{POL}, Action^{POL}\right\}$

represents the transformation into our model of a policy from its natural language representation. Policy tuples are indexed by the integer *i*, where *n* is the number of tuples. The size is always greater than zero because we have assumed the existence of a deny all policy as the last one.

A firewall rule is a finite set of tuples:

$Tuple^{FW}(j)_{0<j\le m} =$

$\left\{Source^{FW}, Destination^{FW}, Service^{FW}, Action^{FW}\right\}$

Rule tuples are indexed by the integer *j*, where *m* is the number of tuples. The size is always greater than zero because we have assumed the existence of a deny all rule as the last one.

$R^{ij}{}_{0<i\le n,0<j\le m}$ is a logic variable that represents a match between a $Tuple^{POL}(i)_{0<i\le n}$ and a $Tuple^{FW}(j)_{0<j\le m}$:

$$R^{ij}{}_{0<i\le n,0<j\le m} = Source^{POL}(i) = Source^{FW}(j) \wedge Destination^{POL}(i) = Destination^{FW}(j) \wedge Service^{POL}(i) = Service^{FW}(j)$$

$P(T_j)$ is defined as a path variable that indicates the rules that have been processed, in a similar way we defined it in a previous work [3]. As order is important in tuples transformed from rule sets, it is needed to simulate that order in our model. For a fixed $Tuple^{POL}(i)$, P is defined take the values:, $\forall j : 0 < j \le k \bullet P(T_j) = TRUE \ \wedge \ \forall j : k < j \le m \Rightarrow P(T_j) = FALSE$

This predicate is also going to serve us to detect where the fault exactly is, as is going to be explained in the next definition.

The predicate $\mathrm{OK}^{\mathrm{FW}} = TRUE$ if

$$\forall i : 1 \le i \le n, \ \exists j : 1 \le j \le m \bullet \left\{R^{ij} \wedge Action^{POL}(i) = Action^{FW}(j)\right\}$$

In any other case, $\mathrm{OK}^{\mathrm{FW}} = FALSE$. If there is a fault, then the last predicate $P(T_k) = TRUE$ identifies it. Take into account that the predicate P is always for firewall tuples, since the policy is axiomatic.

With all these definitions we have the logic model presented in Fig. 6. With this model we can express access control policies only if they involve information flows between different zones; this model cannot express requirements between information flows in the same zone, since no firewall can control that flow. On the other hand, the zones may represent various physical or logical networks that may have routers within them (with no filtering capabilities). These internal routers are of no interest for our model, since we are only interested in filtering devices.

1. $\neg P(T_j)_{j<m} \Rightarrow \neg P(T_{j+1})_{j\le m}$
2. $P(T_j)_{j<m} \Rightarrow (P(T_{j+1})_{j<m} = \neg R^{ij}{}_{0<i\le n,0<j\le m}) \wedge (R^{ij}{}_{0<i\le n,0<j\le m} \Rightarrow \mathrm{OK}^{\mathrm{FW}} = (\mathrm{Action}^{\mathrm{POL}}(i) == \mathrm{Action}^{\mathrm{FW}}(j)))$
3. $P(T_j)_{j=m} \Rightarrow \mathrm{OK}^{\mathrm{FW}} = (\mathrm{Action}^{\mathrm{POL}}(i) == \mathrm{Action}^{\mathrm{FW}}(j))$

Figure 6. Logic model

4.2.2 CSP model. The constraint representation of the previous model is quite direct, and requires no additional definitions (Fig. 7). We have based this constraint model in a previous work [3]. There are only two unbounded variables with boolean domain in the CSP: the variables $P(T_j)_{0<j\le m}$ and $\mathrm{OK}^{\mathrm{FW}}$ that obey to definitions 8 and 9 respectively. There is a special constant representing that the value of the first path variable is always true. This is necessary because is the only way to start the reasoning process at (3).

Then there are four constraints.

- The constraint (1) represents the predicate $R^{ij}{}_{0<i\le n,0<j\le m}$ (definition 7), a match between $Tuple^{POL}(i)_{0<i\le n}$ and $Tuple^{FW}(j)_{0<j\le m}$.
- The constraint (2) represents that

$$\exists j : 0 < j \le m \bullet Tuple^{FW}(j)_{j<m} \wedge \neg P(T_j) \Rightarrow \neg P(T_{j+1})$$

since if the firewall is stopped at $Tuple^{FW}(j)_{0<j\le m}$, then it must be stopped for all following tuples. Note that once a path variable is false (representing that the previous tuple has matched), then all following ones must also be false.

- The constraints (3) and (4) are the same but (3) for a tuple that is not last one and (4) for the last tuple (a deny all rule always matches with everything). These constraints represent that if there is no match at $Tuple^{FW}(j)_{0<j\le m}$ for a given $Tuple^{POL}(i)_{0<i\le n}$, then the match must checked against the following tuple, with $P(T_{j+1})=1$. Also, if there is a match, then $OK^{FW}=1$ (representing a full match: including the action field); $OK^{FW}=0$ in other case.

Note that this constraint model only represents the consistency checking between a given policy $Tuple^{POL}(i)_{0<i\le n}$ and a firewall rule set. To be complete, the explained process should be repeated for all policies, but for the sake of simplicity we have left this for the implementation.

The problem is solved in two phases. First, the constraint solver checks if the complete firewall rule set is consistent with each policy. This process determines the faulty rules at the firewall (different action fields), and also the policies that specify actions that are not represented in the firewall rule set (rules that must be written), as we have stated before.

Unbounded variables

$P(T_j)_{0<j\le m}$, OK^{FW}

Domains

$P(T_j)_{0<j\le m} \in \{0,1\}$

$OK^{FW} \in \{0,1\}$

Constants

$P(T_0)=1$

Constraints

1. $R^{ij}{}_{0<i\le n,0<j\le m} = Source^{POL}(i) = Source^{FW}(j) \wedge Destination^{POL}(i) = Destination^{FW}(j) \wedge Service^{POL}(i) = Service^{FW}(j)$
2. $\neg P(T_j)_{j<m} \Rightarrow \neg P(T_{j+1})_{j\le m}$
3. $P(T_j)_{j<m} \Rightarrow (P(T_{j+1})_{j\le m} = \neg R^{ij}{}_{0<i\le n,0<j\le m}) \wedge (R^{ij}{}_{0<i\le n,0<j\le m} \Rightarrow (\mathrm{Action}^{\mathrm{POL}}(i) == \mathrm{Action}^{\mathrm{FW}}(j)) \Rightarrow \mathrm{OK}^{\mathrm{FW}}=1) \wedge$
   $(R^{ij}{}_{0<i\le n,0<j\le m} \Rightarrow \neg(\mathrm{Action}^{\mathrm{POL}}(i)==\mathrm{Action}^{\mathrm{FW}}(j)) \Rightarrow \mathrm{OK}^{\mathrm{FW}}=0)$
4. $P(T_j)_{j=m} \Rightarrow (\mathrm{Action}^{\mathrm{POL}}(i)==\mathrm{Action}^{\mathrm{FW}}(j)) \Rightarrow \mathrm{OK}^{\mathrm{FW}}=1) \wedge$
   $\neg(\mathrm{Action}^{\mathrm{POL}}(i)==\mathrm{Action}^{\mathrm{FW}}(j)) \Rightarrow \mathrm{OK}^{\mathrm{FW}}=0)$

Figure 7. Constraint model

If the process returns that the problem is consistent, then the reverse process must be also run to be complete. This process identifies rules in the firewall rule set that are not considered in the policy. As the policy is axiomatic, then the faults are that actions that are not considered in the policy.

5. Example

We are going to present an example to illustrate how our model works. The complete process has been implemented using a commercial constraint solver. The example network (Fig. 8) consists of three zones and several systems (take into account that our model can express a firewall with an unlimited number of interfaces):

- Internet. This zone represents all systems with pubic IP addresses that are not in the other segments.
- DMZ. This zone contains the systems that can be accessed from the Internet. This network has private IP addresses.
- Administration. This zone contains systems from employees that use information from sales, salaries, etc.
- Sales. This zone contains systems from employees that use information relevant to customers, prizes, promotions, etc. This zone also contains a mass storage system that must also be accessed from all administration systems.

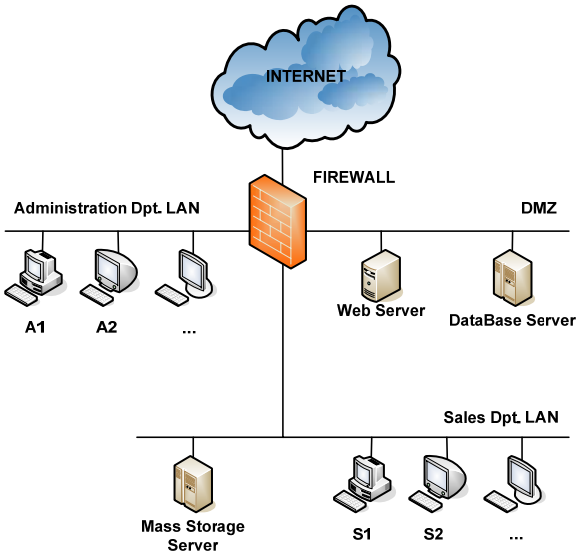

Figure 8. Network with a firewall with four interfaces: Internet, DeMilitarized Zone, Sales and Administration departments

An example security policy for this network, written in natural language, is presented in Fig 9. This policy is also decomposable on several smaller sentences, but it is really not necessary to explicitly do it, since its transformation to tuples is direct (Fig. 10).

1. Users on Internet can access the web server
2. Only the sales person of the month (this month is "S1") of the sales Department can access web pages on Internet. However all people at sales department can receive and send eMail through Internet.
3. Users of the administration department can access web pages on Internet, and also the storage server at the sales department.
4. No more accesses are permitted.

Figure 9. Natural language policy

The next phase is to take the firewall configuration file and translate it to tuples. It is exactly the same process done for the policy, but this time the firewall configuration files can be parsed automatically. Recall that in the firewall rule set there can be any number of deny rules. Fig. 10 presents the modelled firewall rule set.

POLICY

$Tuple^{POL}(1)$={Internet, WebServer, http, allow}
$Tuple^{POL}(2)$={S1, Internet, http, allow}
$Tuple^{POL}(3)$={Sales, Internet, pop3, allow}
$Tuple^{POL}(4)$={Sales, Internet, smtp, allow}
$Tuple^{POL}(5)$={Administration, Internet, http, allow}
$Tuple^{POL}(6)$={Administration, StorageServer, 789, allow}
$Tuple^{POL}(7)$={any, any, *, deny}

RULE SET

$Tuple^{FW}(1)$={Internet, DataBase, *, deny} // Log rule
$Tuple^{FW}(2)$={Internet, WebServer, http, allow}
$Tuple^{FW}(3)$={S3, Internet, http, allow}
$Tuple^{FW}(4)$={Sales, Internet, pop3, allow}
$Tuple^{FW}(5)$={Sales, Internet, smtp, allow}
$Tuple^{FW}(6)$={Administration, Internet, http, deny}
$Tuple^{FW}(7)$={Administration, Internet, pop3, deny} // Log
$Tuple^{FW}(8)$={Administration, StorageServer, 789, allow}
$Tuple^{FW}(9)$={any, any, *, deny}

Figure 10. Modeled policy and rule set

5.1 Firewall diagnosis

At the first phase the solver checks each policy against the complete firewall rule set (so the path variables are for the firewall tuples). There are two faults diagnosed. With $Tuple^{POL}(2)$ the process returns $OK^{FW} = 0$. The last path variable set to true

is $P(T_9)$, the path variable of the last tuple. This implies that there is no rule that complies with that policy, and needs to be written. With $Tuple^{POL}(5)$ the process returns $OK^{FW} = 0$. The last path variable set to true is $P(T_6)$. This implies that the rule 6 matches against the tuple 5, but they have contradictory actions. Since the policy is axiomatic, then it can be argued that $Tuple^{FW}(6)$ is faulty (its action has to be changed).

At the second phase, the solver checks each firewall rule against the complete policy. This time the path variables are for the policies. With $Tuple^{FW}(3)$ the process returns $OK^{FW} = 0$. The last path variable set to true is $P(T_7)$, the path variable of the last tuple. This implies that there is not a policy that complies with that rule. Since the policy is axiomatic, then the fault is necessarily on the third rule of the firewall. So $Tuple^{FW}(3)$ is a faulty rule that expresses that the sales person with Internet access is not the employee of the month, but another sales person.

6. Conclusions and future work

This paper presents a CSP based approach to automatically diagnose firewall rule sets. One of the main contributions of this paper is the specification of a high level policy to diagnose the firewall conformity with a specified security policy. With this approach, it is not necessary to manually test rules, since the specification of which things the firewall must do, or its objectives, are represented in the policy.

Another important contribution is the logic model we have used to model firewall functionality. Both models (policy and firewall rule set) are transformed into a Constraint Satisfaction Problem. Finally, consistency techniques are used to reason, detect and identify (diagnose) faults on firewall rule set.

Our model is also capable of representing the policy and firewall rules independently from the network topology, because the model can use symbolic names to represent the source, destination and service fields of both policy and rules.

The work presented in this paper can be extended and improved in several directions. First, a GUI can be constructed to facilitate the specification of the high level policy by users (and its automatic conversion to tuples), so users doesn't need any more to express the policy in a written language. Second, performance analysis of the CSP engine with a complex firewall rule set should be done in order to optimize our process. At this time, we are trying to extend the model to represent a network with several inter-related firewalls, and also trying to eliminate the equals operator limitation. With this improved model, we expect to be able to diagnose inconsistencies in distributed firewalls more accurately. A performance analysis with real policies could also be very interesting and can provide us with very useful results to improve and optimize our model.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments on the early version of this paper. This work has been funded by the Spanish *Ministerio de Ciencia y Tecnología* under grant DPI2006-15476-C02-01.

References

[1] K. Marriott, P. J. Stuckey. Programming with Constraints: An Introduction, MIT Press, 1998.

[2] IPTables. Netfilter Project. http://www.netfilter.org/

[3] R. Ceballos, R.M. Gasca, C. del Valle, F. de la Rosa. "A Constraint Programming Approach for Software Diagnosis". Proceedings of the Fifth International Symposium on Automated and Analysis-Driven Debugging (AADEBUG'03). Belgium, September 2003.

[4] J. Guttman. "Filtering Postures: Local Enforcement for Global Policies". Proceedings of IEEE Symposium on Security and Privacy (SSP'97), May 1997.

[5] J. Guttman, A. Herzog. "Rigorous Automated Network Security Management". International Journal of Information Security, Vol. 4, No. 1-2, Pages 29-48 Springer-Verlag 2005.

[6] Y. Bartal, A. Mayer, K. Nissim, A. Wool. "Firmato: A Novel Firewall Management Toolkit". ACM Transactions on Computer Systems, Vol. 22, No. 4, Pages 381-420. November 2004.

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

[8] P. Eronen, J. Zitting. "An Expert System for Analyzing Firewall Rules". Proceedings of Nordic Workshop on Secure IT-Systems (NordSec'01), November 2001.

[9] T. E. Uribe, S. Cheung. "Automatic Analysis of Firewall and Network Intrusion Detection System Configurations". Proceedings of ACM Workshop on Formal Methods in Security Engineering (FMSE'04), Ocober 2004