Information and Software Technology 51 (2009) 894–915

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Model-Based Development of firewall rule sets: Diagnosing model inconsistencies

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

ARTICLE INFO

Article history:
Available online 13 May 2008

Keywords:
MBE
Firewalls
Consistency
Validation
Model

ABSTRACT

The design and management of firewall rule sets is a very difficult and error-prone task because of the difficulty of translating access control requirements into complex low-level firewall languages. Although high-level languages have been proposed to model firewall access control lists, none has been widely adopted by the industry. We think that the main reason is that their complexity is close to that of many existing low-level languages. In addition, none of the high-level languages that automatically generate firewall rule sets verifies the model prior to the code-generation phase. Error correction in the early stages of the development process is cheaper compared to the cost associated with correcting errors in the production phase. In addition, errors generated in the production phase usually have a huge impact on the reliability and robustness of the generated code and final system.

In this paper, we propose the application of the ideas of Model-Based Development to firewall access control list modelling and automatic rule set generation. First, an analysis of the most widely used firewall languages in the industry is conducted. Next, a Platform-Independent Model for firewall ACLs is proposed. This model is the result of exhaustive analysis and of a discussion of different alternatives for models in a bottom-up methodology. Then, it is proposed that a verification stage be added in the early stages of the Model-Based Development methodology, and a polynomial time complexity process and algorithms are proposed to detect and diagnose inconsistencies in the Platform-Independent Model. Finally, a theoretical complexity analysis and empirical tests with real models were conducted, in order to prove the feasibility of our proposal in real environments.

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 finite list of linearly ordered (total order) condition/action rules. A rule is defined as follows (Eq. (1.1)):

$$\forall i, 1 \leqslant i \leqslant n, R_i : \{condition_i\} \Rightarrow \{action_i\} \tag{1.1}$$

where i is the position of the rule in the ACL (or its priority) and n is the position of the last rule. ACLs can be forward or backward checked, but in firewalls the most common method is forward checking. The *condition* part of the rule is a set $\{S_1, S_2, \ldots, S_k\}$ whose elements are condition attributes or selectors, and where k is the number of selectors, $kz = |condition|$. The *condition* set is typically composed of five elements, which correspond to five fields in 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 in the header of a packet are subsets or equal to the values of its corresponding rule selector (Eq. (1.2)).

$$match = \left\langle \begin{array}{l} SoruceIP_{packet} \subseteq SoruceIP_{selector} \wedge DestIP_{packet} \subseteq DestIP_{selector} \wedge SourcePort_{packet} \subseteq SorucePort_{selector} \wedge \\ DestinationPort_{packet} \subseteq DestinationPort_{selector} \wedge Protocol_{packet} \subseteq Protocol_{selector} \end{array} \right\rangle \tag{1.2}$$

The *action* part of the rule represents the action that should be taken for that matching packet. In firewalls, two actions are possible: *accept* or *deny* a packet. If the packet does not match with any rule, then the firewall executes the default policy. For most firewalls, everything is denied if it is not explicitly permitted, so their default policy is to deny any packet that has not matched with any of its rules. The default action is usually explicitly expressed as the last rule in the rule set.

Firewalls implement ACLs using their own low-level language, forming what is commonly called a *rule set* (incorrectly, because rules may be repeated producing a redundancy). Fig. 1 represents an example of a rule set written in a specific low-level language.

Although deployment of firewalls is an important step in the course of securing networks, the complexity of designing and managing firewall rule sets might limit the effectiveness of firewall security. Firewalls have to face many problems in modern networks. The main ones are the high complexity of rule set design,

* Corresponding author. Tel.: +34 954559897.
E-mail addresses: sergiopozo@us.es (S. Pozo), ceball@us.es (R. Ceballos), gasca@us.es (R.M. Gasca).
URL: http://www.lsi.us.es/~quivir (S. Pozo).

0950-5849/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2008.05.001

rule set consistency diagnosis, and rule set redundancy diagnosis. Any error defining a rule set may compromise the system security by letting unwanted traffic pass or blocking desired traffic.

Networks have different access control requirements that which must be translated by a network administrator into firewall rule sets. Writing and managing rule sets are tedious, time-consuming and error-prone tasks for several reasons [3]. Translation of requirements into rule sets is not a naïve task. One of the main reasons is that the gap between the high-level access control objectives or requirements and low-level rule sets is too big. Low-level firewall languages are, in general, difficult to learn, use and understand. Each firewall platform has its own low-level language, which the network administrator needs to know in order to implement the access control requirements. Changing from one vendor to another means a complete rewrite of the rule set. Low-level firewall languages are very different from each other in syntax and semantics.

When a model and a language to describe it are to be proposed, there is a compromise between expressivity and complexity. A very expressive model is generally more complex than a less expressive one. We think of complexity as being how difficult it is to represent knowledge of the reality being modelled. There are existing language proposals to model access control lists, some of which are specific to the firewall domain and others are generic to access control. Some research groups have proposed models and languages for general access control policies [4,5]. Some organisations have even proposed models and languages to represent access control policies such as XACML [6] or Rule-ML [8]. However, none of these models is specifically for firewall ACLs, which are a much simpler subset of access control policies. These models and languages tend to be very general and complex to be used in the firewall domain, as they cover a wider spectrum of security policies and applications. They are usually as complex as a low-level firewall language or even more so. We think that complexity is the main reason they are not widely adopted by the industry: it is very difficult to translate access control requirements into one of these non-firewall-specific languages. Another main reason is that there is no automatic method to translate the model expressed in many of these languages into any of the low-level rule set languages.

Thus, there is a clear need for a firewall-specific yet simple abstract model and language with the expressive power of existing firewall-specific languages, but with significantly less complexity than currently proposed languages. The model represented by this abstract language should be automatically translated into any of the existing low-level firewall languages.

Our starting point is the concept of Model-Based Development (MBD), which has been proposed as a model-centric and generative approach to software development. Conceptually, the MBD approach has three parts: (1) developers create system models in high-level modelling languages; (2) tools are used to perform automatic model transformation; and the result is (3) a system architecture. The high-level model used in the first phase is called Platform-Independent Model (PIM). The model which results from the transformation in the second phase is called Platform Specific Model (PSM), and there will be several PSMs, one for each target platform. Finally, the result from the third and final phase is the specific code for a platform, and can be directly executed by it.

However, the use of any of these languages does not guarantee that the model is free of inconsistencies or redundancies, thus a method to isolate and identify inconsistencies and redundancies in the model must be applied. The problem of firewall ACL consistency has been addressed by many works which propose algorithms that work directly with rule sets. We think that it is important that consistency faults should be identified and resolved at specification level in order to generate a consistent rule set from the high-level model. If not, inconsistencies in the model would be translated into the final rule set. In addition, if inconsistencies are resolved directly in the rule set, then the model should also be updated with these changes. It is important to move the verification phase to earlier stages in the process, prior to code-generation.

In this paper, we propose the application of the ideas of Model-Based Development (MBD) to firewall rule set modelling and automatic rule set generation. We propose a PIM for firewall ACLs, which has been designed from an analysis of the most commonly used firewall languages in the industry, using a bottom-up methodology. The languages analyzed were Linux IPTables 1.3.7, Cisco PIX 7.0, FreeBSD 6.2 IPFilter, FreeBSD 6.2 IPFirewall, OpenBSD 3.7 Packet Filter, and Checkpoint Firewall-1 4.1. We also present a polynomial time complexity process and algorithms to diagnose inconsistencies in the PIM, so that it is then possible to generate consistent PSMs automatically.

To the best of our knowledge, this is the first published work to address the issue of rule set design complexity and automatic generation of consistent rule sets using an MBD approach. We have developed a tool (available upon request) which validates our proposal.

The paper is structured as follows. Section 2 revisits the traditional problems of firewall rule sets: design, consistency and redundancy. In Section 3, an MBD approach to automatically generating firewall rule sets is proposed, with the inclusion of a phase for automatic consistency validation of the PIM prior to PSM generation. In Section 4, an analysis of low-level firewall languages is presented, and several alternatives for constructing a PIM in a bottom-up approach from the results of the analysis are discussed. Finally, a PIM and a XML Schema Definition for it are presented. In Section 5, a process and algorithms are proposed to detect and isolate consistency errors in the PIM, with an empirical performance evaluation. In Section 6, we make some concluding remarks and propose some future works. Annex I presents the full analysis of the most widely used low-level firewall languages. Finally, Annex II presents the PIM XML Schema Definition.

2. Firewall open problems: Related works

In this section, we review the various problems that remain partially or wholly unsolved in the field of firewall ACLs. There are

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m state --state NEW -i ! eth1 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth1 -o eth1 -j REJECT
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

Fig. 1. IPTables rule set.

numerous studies about the different problems rule sets and ACLs have. We divide these problems into three areas: the high complexity of rule set design, rule set consistency and redundancy diagnosis, and rule set conformity. Fig. 2 represents a firewall ACL design process as it is carried out nowadays.

2.1. Rule set design

Firewall languages tend to be very low-level, thus writing a rule set is a very difficult task [3] and usually requires an in-depth knowledge of a particular language and the internal workings of firewall platforms. Furthermore, each vendor has its own low-level firewall language. Fortunately, some research groups have proposed languages to model access control policies. In [4], the authors propose a high-level language, Firmato, and model it as an ERD in order to generate a low-level firewall rule set automatically, and provide a mechanism to separate the security policy from the network topology completely. This language can then be compiled into firewall rule sets. However, the complexity of this language is similar to that of many low-level ones. Finally, a major limitation of Firmato is that the language can only represent knowledge in positive logic (*allow* rules), which complicates the expression of exceptions. However, due to this limitation, the rules expressed are always consistent and order-independent.

FLIP [32] is a recently proposed firewall language which can also compile into several low-level ones. Their authors claim that ACLs expressed in FLIP are consistent. However, they are because of one of its limitations: it does not support overlapping between rule selectors. Prohibiting the use of overlaps is a major limitation, since it is impossible to express exceptions, which could result in the need to write a lot of rules to express them. In addition, its syntax is even more complex than Firmato's. However, due to this lack of expressiveness, FLIP ACLs are order-independent. In [5], the authors provide a general language, Ponder, to represent network policies, which do not compile to any low-level platform. However, the complexity of Ponder surpasses the needs of firewall ACLs.

Some organisations have even proposed languages to represent access control policies as XML documents, such as XACML [6], PCIM [7], Rule-ML [8], and SRML [9]. However, none of these languages is specific enough for firewall access control policies, which are a much simpler subset of access control policies. These languages tend to be very general and complex to be used in the field of firewalls, as they cover a wider spectrum of security policies and applications. They are usually as complex as a low-level firewall language. We think that this is the main reason they are not widely adopted by the industry. Even UML has been proposed to model access control policies [29–31]. However, we consider that UML could be an aid for the requirements definition phase, but then these models need to be translated into a more specific high-level firewall language. Note that the use of any of these languages needs a prior requirements definition phase. These requirements must be then translated into these high-level languages, or even directly into low-level ones, as is done nowadays in the industry. However, using UML in the requirements definition phase could be helpful, since UML models can be automatically transformed into some other languages.

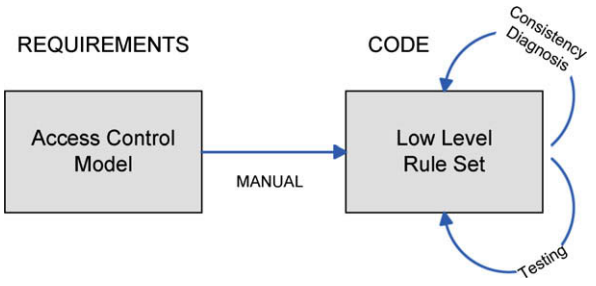

Fig. 2. Firewall rule set creation process.

These models and languages are very generic and are not intended for the area of any particular access control problem. There are some good surveys of access control policy languages available in the bibliography [17–19].

In addition, there are graphical tools that aim to ease the creation of rule sets. One of the most complete ones is Firewall Builder [20], which creates an object-oriented firewall model and can compile it into many low-level firewall languages. Other simpler tools are aimed at only one firewall language. All these tools are also too complex, since they do not hide the low-level firewall details, but only masquerade them with a graphical representation. In general, these tools are not capable of reversing the process. That is, recognizing a manually generated rule set and transforming it into the abstract model.

We think that there is a clear need for a specific yet simple abstract model and language with the expressive power of existing firewall-specific languages, but with significantly less complexity than the currently proposed languages.

In this work, the main and most widely used firewall languages are analysed in order to create an abstract model of them. Then, various alternative models created in a bottom-up process are discussed. Finally, a PIM created on the basis of the analysis is proposed. In a later stage, the PIM should be automatically translated into any of the existing low-level firewall languages. However, PIM to PSM transformations are not the subject of this paper.

2.2. Consistency and redundancy diagnosis

The use of any of these languages does not guarantee that the resulting low-level rule set is free from inconsistencies or redundancies, thus a method to isolate and identify inconsistencies and redundancies must be applied prior to rule set generation. Rule set consistency problems have been addressed by many works. As can be seen from the example in Table 1 (taken from [10]), rule selectors can overlap (for example, the protocol selector), and there can even be rules that are exactly the same as others. There is a possibility of inconsistency when two or more rules with different actions overlap, because a packet can be matched with all the overlapping rules, and depending on the priority of the rule, the packet will be matched against one rule or other, and a different action will be taken. For example, a packet with {*SrcIP* = 140.192.37.20, *SrcPort* = *any*, *DstIP* = 161.120.33.40, *DstPort* = 80, *Proto* = *TCP*} can match *R1* and *R2*. If *R1* is first, then the packet will be denied, but if *R2* is first, the packet will be accepted.

The difficulty of writing and modifying a rule set increases with the number of rules, since it is easier to write a rule which overlaps with another previously written one. The same problem arises with rule modification. The problem is aggravated if the rule set is maintained by different administrators, since rule sets are usually not commented on anywhere. Rule sets are usually composed of a number of rules ranging from a few dozens to five thousand [16].

One of the main works dealing with rule set consistency and redundancy [10] is a significant advance for the community because its authors defined a complete inconsistency model for firewall rule sets [11]. In their works, they provide an order-dependent characterization of different kinds of inconsistencies that may exist between pairs of rules in a firewall rule set. However, their approach can only detect and diagnose inconsistencies between pairs of rules and does not analyze problems with a combination of more

Table 1
Example of the filtering selectors of a firewall rule set

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	Allow
R3	tcp	*.*.*.*	Any	161.120.33.40	80	Allow
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	Allow
R7	tcp	140.192.37.*	Any	161.120.33.40	21	Allow
R8	tcp	*.*.*.*	Any	*.*.*.*	Any	Any
R9	udp	140.192.37.*	Any	161.120.33.40	53	Allow
R10	udp	*.*.*.*	Any	161.120.33.40	53	Allow
R11	udp	140.192.38.*	Any	161.120.35.*	Any	Allow
R12	udp	*.*.*.*	Any	*.*.*.*	Any	Deny

than two rules. In addition, they use rule decorrelation techniques to decompose the rule set into a new one with non overlapping rules. As the decorrelated rule set is free from overlaps, the new rule set would have more rules than the initial set. In addition, the proposed decorrelation process [12] is worst-case exponential time and space complexity with the number of rules. Although the proposed diagnosis and characterization algorithms are polynomial, the decorrelation pre-process imposes a worst-case exponential time and space complexity for the full process. In addition, it is important to note that results are given for the decorrelated rule set, which is bigger than and different from the original. The system administrator is responsible for understanding the decorrelated rule set in order to remove the inconsistencies.

Another work try to address this limitation [13] by proposing a new algorithm, which is a combination of [4,16]. Their authors have also extended their work to distributed firewalls [14]. However, due to their use of the same decorrelation techniques, these proposals have the same algorithmic complexity as previous works, and the resulting rule set is also bigger than and different from the original.

Others have tried to address this problem using OBDDs [15]. A very important advantage over the previous proposals is that they do not need to decorrelate the rule set, and thus, results are given for the original. However, the complexity of OBDD algorithms depends on the ordering of its nodes, which is a NP-Complete problem [27], although non-optimal heuristics can be used. However, the authors do not provide any ordering criteria for OBDD nodes. This results in a worst-case exponential time complexity with the number of rules, as with the other proposals.

Model building is a standard practice in software engineering. The construction of models during requirements analysis and system design can improve the quality of the resulting systems by providing a foundation for early analysis and fault detection. The models also serve as specifications for the later development phases and, when the models are sufficiently formal, they can provide a basis for refinement down to code. We believe that it is important that consistency faults should be identified and resolved at specification level in order to generate a consistent rule set from the high-level model. If not, inconsistencies in the model would be translated into the final rule set. In this paper, we propose moving the verification phase to earlier stages in the process, prior to code-generation.

In this paper, we provide a definition of inconsistency that is just the opposite direction of that other authors have taken, recognizing all the possible kinds of inconsistencies with a unique definition, instead of a different definition on a per-inconsistency-type basis. We then propose a consistency-based diagnosis process for inconsistencies in the PIM. Our algorithms have a polynomial worst-case time complexity in $O(n^2)$ with the number of rules in the PIM, n. The result of the process is a set of rules that can be removed in order to obtain a consistent PIM. The proposed algorithms do not need to decorrelate the PIM rules as a pre-process. We think that for a result to be useful for a final human user, it should be given for the original filter or rule set.

2.3. Rule set conformity

Another reason why the task of writing or modifying a rule set is very difficult and prone to errors is that most organisations do not have clear security objectives. Much less, they have a security policy or a more specific access control policy, which needs to be implemented and enforced with firewalls. When an inconsistency is detected in a firewall rule set: how does one know which one of the conflicting rules is problematic? Even in a conflict-free rule set, how does one know if the rule set is implementing the correct policy? These are two typical examples of *conformity* problems. To solve this kind of problem, it is necessary to specify an access control policy to be compared with the firewall rule set. This process is called *conformity checking*, and can be used before or after consistency checking, since it is a complementary process. This problem has been addressed by some authors, using automated and manual approaches. In [21,22], the authors use an undirected bipartite graph representation for the network topology, and algorithms to represent firewall rule sets and the reason about packet trajectories. The authors of [23] provide an abstract language to represent policies. It is possible to query this abstract representation about concrete paths, or even to generate all possible queries and present them to the user, who is responsible for deciding if the firewall works as expected. In [24] a method is presented based on graph algorithms to reason about IDS configurations in combination with a firewall configuration of the network. In [26], the authors propose a CSP-based approach to detect and diagnose conformity faults in a non-distributed firewall environment. In [25], authors' propose algorithms that represent a firewall rule set using ordered binary decision diagrams. These algorithms can be used to improve firewall performance and also to validate the rule sets. Finally, in [28] a test-case approach to testing the conformity of a firewall ACL to a policy is depicted.

3. Model-Based Development for firewalls

Our starting point is the concept of Model-Based Development (MBD), which has been proposed as a model-centric and generative approach to software development. Conceptually, the MBD approach has three parts: (1) developers create system models in high-level modelling languages; (2) tools are used to perform automatic model transformation; and the result is (3) a system architecture. The high-level model used in the first phase is called the Platform-Independent Model (PIM). The model which results from the transformation in the second phase is called the Platform-Spe-

cific Model (PSM), and there will be several PSMs, one for each target platform. Finally, the result from the third and final phase is the specific code for a platform, and can be directly executed by it.

Firewall platforms are very different from one company to other, and even among the available Open Source platforms. These differences range from differences in the number, type and syntax of selectors that each platform's filtering algorithm can handle, to huge differences in rule-processing algorithms that can affect the design of the rule set. For example, with regard to filtering selectors, IPTables can filter a packet taking into account nearly all the possible fields in its TCP/IP header. However, Cisco PIX only can filter using a very small subset of the fields in the TCP/IP header of a packet. In addition, for each selector, each firewall platform may support different kinds of data type to represent its content. In this way, IPTables permits for example, the use of IP-ranges in the IP address selectors, but Cisco PIX only permits the use of a unique IP or a block in CIDR format. Surely an IP-range can be translated into several blocks of IPs, so both data types (or syntaxes) are equivalent.

Some questions may arise in this situation. The first is whether all firewall platforms analyzed share a common set of filtering selectors. Another is, for the common selectors, if there is at least one common syntax among all firewall platforms; or if not, if the available syntaxes for each platform have equivalencies in the other platforms.

These common sets of selectors and syntaxes, if they exist, plus the action selector, may be the starting point for a Platform-Independent Model for firewalls. There is no need to include more information in the PIM, since filtering actions are taken in the condition part of a rule, which is based on the possible filtering parameters. Thus, the PIM will be an abstraction of an access control list. The PSM represents platform details related to all other things. That is, mainly NAT support, connection tracking, chains and other types of rule-processing, and logging. This separation fits well with the proposed MDB approach. Fig. 3 presents a proposal for the division of characteristics.

Firewall platforms have other specific characteristics. These range from how each platform threats connection tracking (that is, stateful or stateless connections), how the rule-processing is performed (forward, backward, with jumps), etc. In general, these characteristics are not related to filtering parameters or selectors. For example, in terms of connection tracking support, all analyzed firewall platforms support stateful and stateless connections except Cisco PIX, which only supports stateful ones. Another example is that IPTables permits the use of several rule sets (or chains) and can define jumps between them, in a forward-check process. However, IPFilter and PF do not support chains, and their default behaviour with regard to rule filtering is to backward-check them. For this reason, the same rule set would not be processed similarly in IPFilter and in IPTables. Those characteristics not related to filtering parameters should be considered in another more specific model, as they are platform-specific ones, and may be the starting point for the PSM for each firewall.

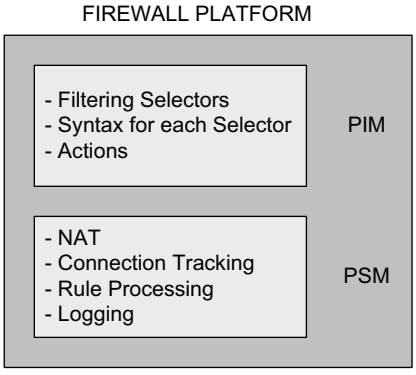

Fig. 3. PIM and PSM abstraction proposal.

Finally, an automatic transformation process should be defined in order to transform the PIM into each PSM, taking into account the particularities of each firewall platform. We propose using this architecture and applying these ideas to design firewall rule sets, in order to solve some of the problems discussed in Section 2.3. Fig. 4 depicts the complete process.

In the first step of the process, an engineer obtains the access control requirements for the organisation, in a requirements definition phase. These requirements are the Computation Independent Model or CIM. The requirements are then translated into the PIM. The use of UML in the requirements definition phase could be very helpful, since UML models can be automatically transformed into any other language.

Then, in order to generate a specific model for a firewall platform automatically, some platform-specific markers should be applied to the PIM. These markers cover knowledge of a specific platform, and can transform the PIM into a more specific model, or PSM. The PSM is specific for each firewall platform. Once a PSM is obtained, an automatic process should be applied in order to generate specific rule sets for its corresponding firewall platform. This process can be abbreviated if the PIM contains sufficient information to generate the specific rule set directly.

The objective of this work is the first part of the process: an analysis of the most widely used firewall platforms in the industry, a definition of a PIM for firewalls, which will be validated using a consistency-based diagnosis technique which we also propose in this work.

3.1. Verification and Validation of Models in MBD

In any software or hardware modelling methodology, the need for complete, consistent and precise models is of extreme importance. Error correction in early stages of the development process is cheaper compared to the cost associated with the error correction in the production phase [35]. In addition, errors generated in the production phase usually have a huge impact on the reliability and robustness of the generated code and final system [36].

In an MBD approach, this is even more important, since models are the core of the methodology, and executable code will be automatically generated from models. MBD facilitates the generation of application software from high-level models. Currently, Verification and Validation (V&V) tools and techniques for the quality assurance of such software are not integrated into the standards for mode-driven architecture. This limits the applicability of an MBD approach and leaves software engineers with the problem of how to apply these verification and validation tools and techniques most effectively.

V&V may be used to diagnose inconsistencies and faults in early stages of the process, limiting the propagation of these problems to later phases. If incomplete, imprecise or inconsistent models were not detected during the PIM construction phase, there would be a propagation of these problems to the executable code.

Verification is the task of proving that a model satisfies a specific property. Verification techniques can be formal and informal. Checklists and ad hoc algorithms are informal techniques, while theorem proving and model checking are formal ones. For example, a commonly used informal verification technique is eye-review of the source code of a program prior to running it. In general, verification can be carried out without an execution environment and prior to deployment, thus it is a static analysis technique.

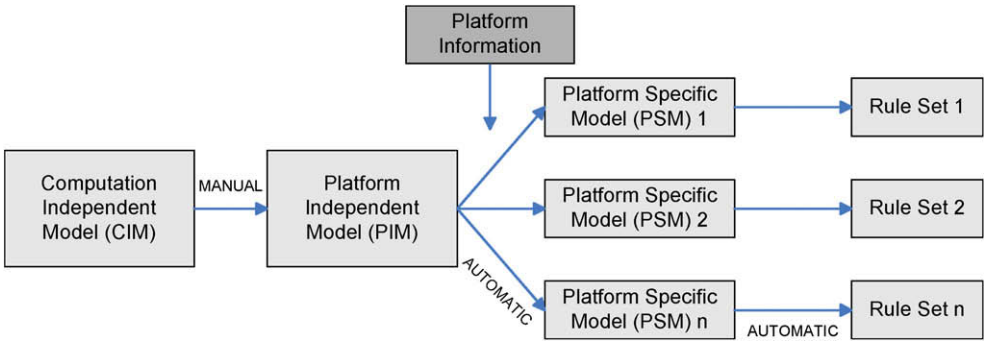

Fig. 4. Model-Based Development in firewalls.

Validation evaluates if the observable behaviour of the model complies with the requirements. Validation techniques include scenario and use case simulation, and testing.

In firewalls, an access control policy is processed by a software algorithm. This algorithm matches packets against the access control policy, which is represented as an ACL. As we have explained in previous sections, firewall rule sets may have inconsistencies and redundancies. Since the rule set is automatically generated from a PSM, which has been derived from a PIM that represents the ACL, then it is important to detect and diagnose inconsistencies and redundancies in the modelling phase. That is, we propose introducing V&V into the process depicted in Section 3 (Fig. 4).

Since the focus of this work is on the first model in the MBD process, which is the PIM, then there are no execution details, and only a verification step can be added. Thus, prior to obtaining a final PIM to be used for the generation of the PSM, a mandatory verification step is necessary in order to guarantee that the modelled access control policy is at least consistent (Fig. 5). We propose constructing the PIM and then verifying it, although verification can also be performed interactively while constructing the PIM. We do not consider redundancy in this work, since redundancy faults do not change the semantics of the access control policy, but only degrade its performance once executed in a runtime environment. Carrying out verification in early stages guarantees that the resulting PSMs and rule sets will be consistent. Fig. 5 presents the parts of the MBD process that are the focus of this paper (those shadowed: the PIM and its verification).

However, this process can be complemented once the rule set is generated and deployed in a real scenario. Testing can be used with real data sets [33] and data mining techniques [34] to optimize the rule set.

4. A PIM for firewalls

When an abstract model is to be proposed, there is a compromise between expressivity and complexity. A very expressive model is generally more complex than a less expressive one. We think of complexity as being how difficult it is to represent knowledge of the reality being modelled. As noted before, existing model proposals to represent access control policies are very complex, because they cover a wide spectrum of domains and applications. We think that complexity is the main reason why none of these languages is widely adopted by the industry: it is very difficult to translate access control requirements into one of these non firewall-specific languages, and thus there is no benefit obtained from its use in comparison with the use of any of the low-level ones.

Firewall ACLs are only a small subset of access control. We think that a simpler yet highly expressive language is required to cover this need. For this reason, we have analysed the most widely used firewall languages, discussed different models, and finally proposed a new one. The model can be used to construct a specification of the firewall access control list (ACL).

In order to represent ACLs in the PIM, it should be taken into account that ACLs are composed of rules, thus only filtering selectors and actions should be considered for the PIM. Other parts of the platform, such as connection tracking or logging are not part of the PIM and should be analyzed for the PSMs, which is not part of this work. We have analyzed the syntax and semantics of the most widely used firewall languages, in order to acquire sufficient knowledge of their characteristics and possibilities. Some of these firewalls are commercial products and others Open Source. This analysis is based on the public documentation available for each language, on tests we have carried out in real environments, and in meetings with experts. The platforms analyzed were Linux IPTables 1.3.7, Cisco PIX 7.0, FreeBSD 6.2 IPFilter, FreeBSD 6.2 IPFirewall, OpenBSD 3.7 Packet Filter, and Checkpoint Firewall-1 4.1.

The complete analysis of filtering selectors and their available syntax is presented in Annexe I. We recommend its use as a reference for this paper. Remember that selectors are used for the decision part of a rule, and that only selectors and actions will be used for the PIM.

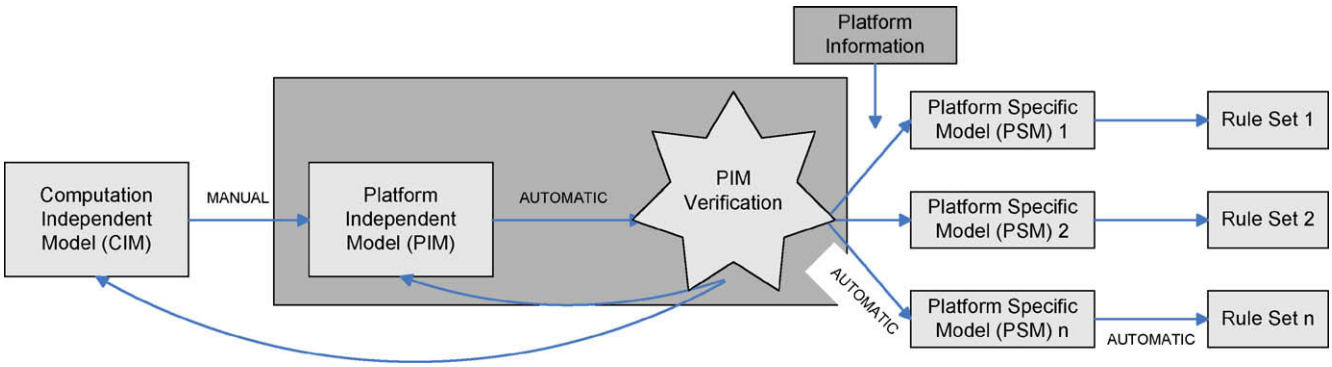

Fig. 5. Verification of PIM in Model-Based Development for firewalls.

4.1. PIM construction alternatives

When abstracting from different realities in order to generate a model in a bottom-up process, there are always three main approaches to take (Fig. 6). These approaches, applied to the firewall problem domain are:

- *Factorization:* The first approach is to analyze different firewall platforms and factorize the common filtering selectors. Each selector can be expressed using different syntaxes, so factorization of these syntaxes is also necessary. The meta-model will be composed of only the common filtering selectors and their common syntaxes of all firewall platforms.
- *Aggregation:* This is just the opposite approach to factorization. The key idea is to take all filtering selectors with all their possible syntaxes, and form the meta-model with them. Thus the meta-model will be composed of common and non-common parts of each firewall platform.
- *Customization:* Finally, this approach is the hybridization of the other two. The starting point is a factorized meta-model. Then, each non-common selector should be analyzed. If the selector can be emulated with the factorized ones, then a usability analysis should be performed. If the selector adds new functionality and can be emulated with the common selectors, then it may be added to the meta-model. The same analysis (emulation and usability) should be carried out with non-common syntaxes for each selector in the model. Syntaxes that are aimed at easing the task of PIM instantiation should also be considered, since instantiation of the PIM is a task that is usually carried out by a human. However, if the PIM were automatically generated from a higher-level specification, such as a UML model, then the model could be simplified even more, reducing the supported syntax for each selector.

Since models are simplifications of reality, none can represent the full reality. Meta-models generated taking these different approaches will be very different, and will have different levels of detail. In this section, these possibilities for firewall language modelling are discussed, and their main benefits and drawbacks are analyzed. Finally, a definitive platform-independent meta-model and a PIM are proposed.

4.1.1. Factorization of selectors

The first approach is to analyze different firewall languages and factorize the common filtering selectors. Selectors can be expressed using different syntaxes or domains, and thus factorization of these syntaxes is also necessary. Only the common selectors and their common domains should be used for the PIM (Fig. 7).

The main benefit of this approach is that the resulting PIM model is minimal and very simple, but not necessarily complete, because there is no guarantee that the PIM can represent the reality at all. For example, for two firewall languages, if the factorization of their possible syntaxes for a common Source IP selector is empty, then there is no common domain for this selector. Thus, the Source IP selector cannot be represented in the PIM. Note that if the Source IP address cannot be represented in the model, it is not possible to model rules taking decisions based on the packet's source IP.

This fact is more important for some selectors than for others. The selectors most used in real rule sets have wide domains (thin granularity), because they allow rules for a wider range of packets to be defined. The less used selectors usually have stretch domains, because they allow rules with thick granularity to be defined.

In addition, a PIM with selectors more typically used in firewall languages is easier to understand for the end user than a PIM with rare selectors that are only supported in a minority of firewall languages. Having few selectors with wider domains is also easier than multiple selectors with stretch domains. For example, the TCP Flags selector is not very important, because although it could have some utility in some specific situations, it is not widely used in isolation from typical selectors, which according to real rule sets are: Source and Destination IP address, Protocol, Source and Destination Ports, and ICMP Type. All the non-common selectors are of optional use in the platforms analyzed (Annexe I).

The main drawback of this approach is that, in general, the resulting PIM may not be sufficiently complete to represent the majority of the reality needed by users. There would be parameters that could not be represented in the PIM. In the analysis, we have

Factorized AFPL Model	Aggregated AFPL Model	Customized AFPL Model
- Simple	- Complex	- Balanced
- Minimal	- Maximal	- Customized
- Not complete	- Complete	- Not complete

Fig. 6. Different possibilities for model construction.

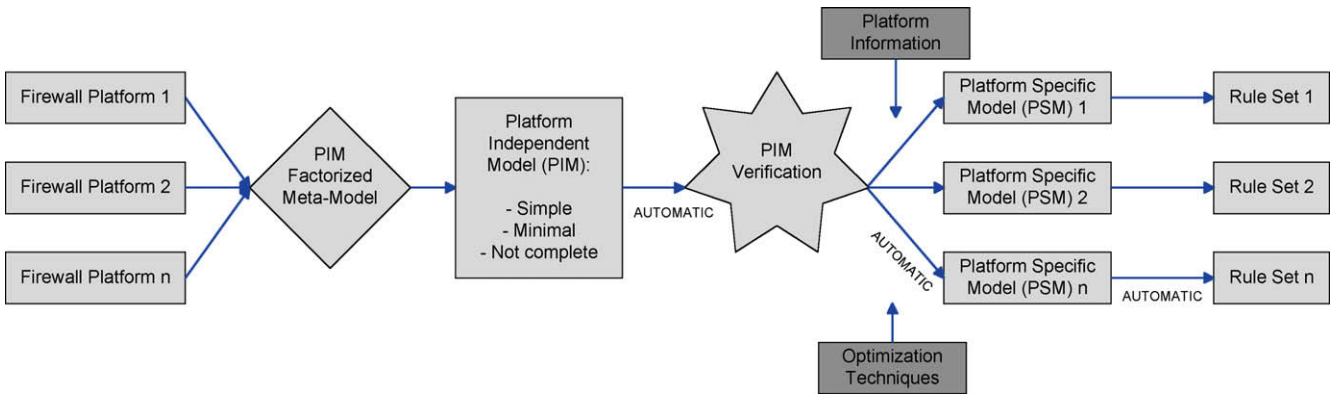

Fig. 7. Construction of PIM by selector factorization.

carried out for firewalls these are TCP Flags, TCP Options, IP Options, IP Version, Source and Destination MAC address, Type of Service and Time to Live selectors. These selectors are not common to all firewall languages and thus cannot be used in the PIM with this approach. In addition, these selectors are marked as optional in the analysis in Annexe I, because their use is optional in all the firewall platforms analyzed. If the PIM does not use these selectors, then the PSM will not use them either, and nor will the generated code. Fortunately, these selectors are not widely used because they are only applied in very specific situations. Indeed, some of them are only used in one of the firewall languages analyzed. In addition, for example the Cisco PIX platform does not support any of these selectors, but it is one of the most widely used firewalls in the world, showing that complexity is not a good approach to language design.

The same problem arises for common selectors. The syntaxes supported by the same selector are usually different in different firewall languages, with some common ones. Differences in syntax for a given selector could have some benefits: more compact rule sets that result in less memory consumption and/or fastest processing, ease the task of writing a rule set manually, etc. If the PIM model is written manually, the use of a richer syntax could ease the task. However, all these different syntaxes need to be translated into syntaxes supported in each specific PSM, which complicates the translation process. A translation is possible (*necessary condition)* if properties exhibited by a specific firewall platform can be reproduced in another with no extensions to its language or modifications to the matching algorithm. Thus, there could be cases where a translation may be impossible.

In the same way, a simpler syntax could be optimized (Fig. 8) to a specific syntax used in a specific PSM of a firewall platform. In the approach we have taken, as only common parts are permitted, only common syntax is permitted. The PSM would have the specific syntax for all the supported selectors. This obliges us to analyze the possible optimizations that could be applied in the PIM to PSM transformation process in order to generate efficient PSMs, and thus efficient code, using the possibilities of the underlying firewall platforms. The example presented in Fig. 8 depicts the situation of an IP-range syntax that is only supported by IPTables firewalls. This syntax groups a list of IP addresses, easing the task of writing rules, and resulting in a more compact rule set. It is similar to other syntaxes used in BSD firewalls, like the use of lists to group IP addresses. However, the IPTables documentation explains that a rule with a selector using this syntax is decomposed into several rules, resulting in no computing benefit in reality. The only real benefit from this syntax is the ease of use but, as the resulting code is not going to be directly manipulated by a human, there is no real benefit from using this kind of optimization in the proposed MBD approach. Optimizations should be made with care.

With this approach, resulting PSMs may not use all the possibilities of the languages and the platforms, although some optimizations can be made to the PIM. The resulting code from the PSM is equal in quality to it. However, a less efficient generated rule set is in part normal, because with a simple model, all optimizations should be made automatically. The same is the case with today's compilers. A compiled code from a high-level language wastes more memory and is less efficient than an assembler code, as the high-level language abstracts the user form registers, multimedia instructions, pointers, and even memory (de)allocation in an Object-Oriented paradigm. Lower performance and higher memory consumption is usually the price to pay for abstraction.

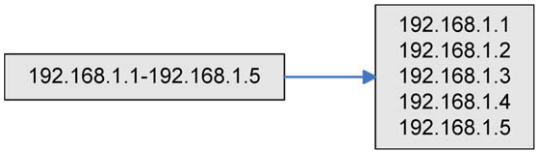

Fig. 8. Internal workings of IPTables IP-range syntax.

Finally, note that this approach is only valid if the resulting PIM can represent the selectors which are mandatory, with at least one possible value for each domain. Fortunately, this is the case for firewalls, as we will explain in a later section.

We think that including information that is only relevant to a minority of firewall languages in the PIM is contaminating it with platform-specific information, and as the PIM is a high-level model, we prefer to keep it as abstract as possible, but without losing too much expressiveness. This approach results in the simplest PIM, but also in a less specific PSM.

4.1.2. Aggregation of selectors

This case is just the opposite of the previous one. The PIM is composed of the selectors that all the firewall languages have, with no exception (Fig. 9). Thus the resulting model will be composed of *all* common and non-common parts of the low-level language in each firewall platform.

The main benefit of this approach is that the resulting PIM model is complete, but not necessarily minimal or simple, because it is highly possible that some selectors included in the PIM are only used in a small number of firewall platforms. The same is true for syntaxes used in selectors. This gives users a high degree of flexibility (and also complexity) when instantiating a PIM.

The main drawback of this approach is complexity. The complexity of the PIM is equal to the sum of the complexities of each firewall language, and the resulting model is not abstract or platform-independent, but highly specific to all the platforms (and not only to one). This resulting PIM could be similar in complexity to existing proposals for high-level languages, and to the models proposed by some graphical interfaces (like Firewall Builder [20]). For example, with this approach, the non-common selectors (TCP Flags, TCP Options, IP Options, IP Version, Source and Destination MAC address, Type of Service and Time to Live) can be used in the PIM. When creating a PIM for a specific problem, the user has to cope with all the complexity inherited from each firewall language (a large number of selectors, wide domains, etc.), which is clearly worse than if the user coded the solution directly in a particular low-level language. We think that this is the main reason why existing proposals for languages have not been used in industry.

Having a PIM with selectors and syntaxes that are not supported by all firewall platforms has several disadvantages over the previous approach. First of all, selectors not present in a particular firewall platform must not be present in its PSM, either. This means that those selectors need to be emulated, but emulation may be impossible. For example, if a selector is not considered in a firewall platform, it is because the platform's filtering algorithms cannot filter with that selector. This is the case, for example, for the Cisco PIX platform, which does not support any of the non-common selectors. Thus, the only way to transform a PIM with non-common selectors into the PSM of a platform that does not consider those selectors, is to ignore them. Obviously, none of the semantics of the PIM can be ignored in order to obtain a conformant PSM and code.

Another similar problem arises with selector syntaxes. If non-common syntaxes are used, then these need to be emulated in the PSM using the syntaxes permitted by the underlying platform. This is the case for the IPTables IP-range depicted in Fig. 8. As users are provided with syntaxes in order to ease design, they can usually be emulated.

With this approach, the resulting PSM could be very difficult, or even impossible to generate, since there are selectors that are

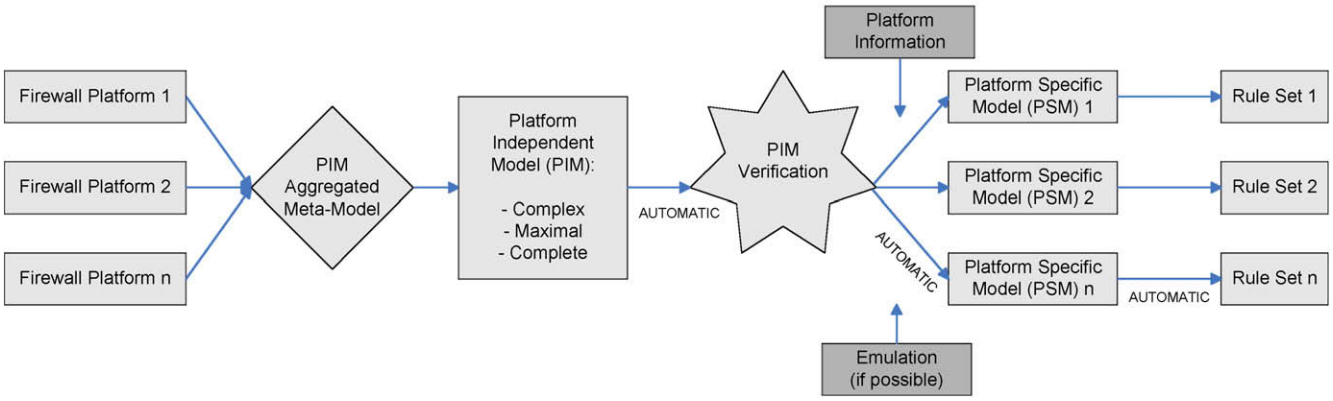

Fig. 9. Construction of PIM by selector aggregation.

impossible to emulate. Although emulation of different syntaxes is possible for selectors, the resulting PSM will always be less efficient than the PIM, since the translation is from a complex expression into a simpler one, and will usually result in more rules. Optimization, as in the case of the previous approach, is easier to perform than emulation. The resulting code is equal in quality to the PSM generated, as in the case of the previous approach.

In conclusion, the resulting PIM is not platform-independent, and very complex. Also, transformations into PSMs are very complex and even impossible, because they require emulations. These are two powerful reasons to discard this approach.

4.1.3. Customization of selectors

This is the trade-off approach. In this approach, each selector is analyzed in detail. If a non-common selector can be emulated and adds new functionality, then the selector may be incorporated into the PIM. If a selector cannot be emulated, its emulation is too complicated, or it does not provide clear advantages over not using it, then it is discarded. The same applies to selector syntaxes. Bear in mind that only syntaxes that provide clear usability improvements for humans should also be considered, as the PIM instantiation will be carried out manually. However, if the PIM were generated automatically from a higher-level specification, such as an UML model, then the model could be simplified even more, reducing the supported syntax for each selector. However, this is not the case presented in this paper.

The main benefit from this approach is that the usability and performance of the resulting rule set are balanced, and for this reason, this is the approach we have taken to design the PIM. With this approach, some selectors and syntaxes used in the PIM will be emulated in the PSM, and others will be optimized from the PIM to each PSM. We do not detail this approach in this section, because it is detailed in subsequent sections.

4.2. Platform-Independent Model, PIM

Summarising the results of the comparative analysis in Annexe I, the resulting set of factorized selectors and their syntaxes from the firewall platforms analyzed are presented in Table 2. The factorized filtering parameters for all firewall languages are Source

Table 2
Common selectors and syntaxes of analyzed firewall platforms

Selector	Obligation	Dependencies	Syntax	Comments
Source IP address	Mandatory		–IP –Block (CIDR) –Wildcard	
Destination IP Address	Mandatory		–IP –Block –Wildcard	
Interface	Optional		–Identifier	
Interface Direction	Optional		–In –Out	IPTables cannot represent direction, but can differentiate inbound from outbound interfaces
Protocol	Mandatory		–TCP –UDP –ICMP –Number –Wildcard*	*IPFilter and PF do not have wildcards, but the omission of this selector has the same effect
Source Port	Optional	Only if Protocol is TCP or UDP	–Number –Range: ≠ p –Range: [p1,p2] –Identifier –Wildcard	
Destination Port	Optional	Only if Protocol is TCP or UDP	–Number –Range: ≠ p –Range: [p1,p2] –Identifier –Wildcard	
ICMP Type	Optional	Only if Protocol is ICMP	–Number –Identifier	
Action	Mandatory		–Allow –Deny	

and Destination IP Addresses, Interface and Direction of the packet flow, Protocol, Source and Destination Ports if the protocol is TCP or UDP, and ICMP Type if the protocol is ICMP. None of the non-common selectors has been considered in this first meta-model: TCP Flags, TCP Options, IP Options, IP Version, Source and Destination MAC address, Type of Service and Time to Live. This represents our starting point, as a basic meta-model. In the next sections, more complexity will be added to this basic meta-model, in order to customize the yet-to-be considered selectors and their syntaxes.

4.2.1. Addition of non-common selectors

In this section, we analyze the possibility of adding some of the non-common selectors to the basic meta-model. A non-common selector can be added if its behaviour can be emulated by the common selectors of *all* the firewall platforms considered, and it also adds new functionality to the model. A selector *adds new functionality* to the model if it cannot be emulated using rules which do not contain it. For example, the *Time to Live* selector cannot be emulated by decomposing the rule that uses it into several other rules which do not use it. A selector is emulable if its behaviour can be reproduced using a combination of one or more rules but using only the common firewall language selectors. Non-common selectors have been analyzed in Annexe I. Fig. 10 presents a summary of the analysis.

The TCP Flags selector can only be used in IPTables, IPFilter, IPFW and PF, and only when the protocol is TCP. This selector is used to match packets with 0, one, or more SYN, ACK, FIN, RST, URG, PSH flags active. Thus, the use of this selector is very limited to very specific situations. In addition, it cannot be emulated in PIX and FW-1, because their matching engines do not support these selectors, which are impossible to emulate with the common selectors considered for all platforms. Thus, they cannot be incorporated into the PIM.

The TCP Options selector is only available in IPTables and IPFW, and can only be used when the protocol is TCP. It is used to match packets depending on their TCP options flag, which is represented by a number. For the same reasons as TCP Flags, this selector cannot be incorporated into the PIM.

Other selectors are IP Options, which can only be used by IPFW, *IP Version* which is only available for PF, Source and Destination MAC addresses which are only considered in IPTables and IPFW, *Type of Service (ToS)* which is not available in PIX and FW-1, and finally *Time to Live (TTL)*, which is only available in IPTables, IPFilter and IPFW. All these selectors cannot be incorporated into the PIM for the same reasons as above: not all matching engines support matching against these selectors, and thus they cannot be emulated with those available.

The conclusion is that none of the non-common selectors can be emulated, and thus cannot be added to the meta-model. Some of these selectors may be added to the PSM using markers to the PIM.

4.2.2. Adding more syntaxes to selectors

In this section, we analyze the possibility of supporting non-common syntaxes in all the selectors considered. In general, we consider a non-common syntax as a candidate for addition to the group of supported syntaxes for that selector in the PIM if it can be emulated with the common syntaxes of the same selector and it provides clear improvements in usability for human users. A syntax provides clear improvements in usability if its use by a human cannot introduce inconsistencies into the model and it provides compactness. This characteristic is subjective, and there are syntaxes that have not been included in the PIM, which could be considered for inclusion. However, since one of our main objectives is to keep the model as simple as possible, we have preferred to leave them out. All possible syntaxes for each selector in the firewall platform analyzed have been reviewed and explained in Annexe I. Note that nearly all non-common syntaxes are supposed to provide compaction of the rule set. However, we think that most of them also generate confusing rule sets.

Selector	Emulable	New Functionality	Add to AFPL
TCP Flags	*No*	*Yes*	*No*
TCP Options	*No*	*Yes*	*No*
IP Options	*No*	*Yes*	*No*
IP Version	*No*	*Yes*	*No*
Src/Dst MAC	*No*	*Yes*	*No*
ToS	*No*	*Yes*	*No*
TTL	*No*	*Yes*	*No*

Fig. 10. Analysis of non-common selectors.

- *Source and Destination IP:* For these selectors there are several non-common syntaxes to be analyzed. IP addresses can also be expressed as identifiers in FW-1, providing new functionality and a clear improvement in usability. IPTables and FW-1 support the specification of groups of continuous IP addresses in these selectors, but their matching engines decompose a rule with this syntax into several rules with one IP per rule. A range of IP addresses can be expressed in several rules, thus we consider that this does not provide a clear improvement in usability. IPFilter, IPFW, PF and FW-1 support domain names, but as a domain is the same as an identifier, no new functionality is provided. PF supports tables, and PF and FW-1 support tables and lists, which are very similar to ranges, and thus will not be considered. IPFW permits logic expressions of IP addresses with OR and NOT logic operators, which again provides a new user-oriented functionality which we consider too complex and error-prone. PF can use an interface name (referring to the block of IP addresses statically assigned to that interface), a block of IPs for that interface and an interface which receives its IP from DHCP. Although this provides new functionality, in this case the functionality is already in the actual meta-model, since it supports the specification of rules per interface. All firewall platforms permit negation of address selectors, except PIX and IPFilter. Although this is new functionality, it provides no clear advantages over a syntax without negation, because a negated selector can be emulated using a combination of allow and deny rules. In conclusion, no new syntax will be added for Source and Destination IP address.
- *Interface:* Interfaces can be expressed in all firewall platforms as identifiers, representing the name of the interface of the underlying platform. All platforms except PIX and FW-1 can use groups or lists to represent a collection of interfaces, providing no new functionality but a minor improvement in usability (firewalls typically have a few interfaces, which can be reproduced using a number of rules). IP addresses can be used in IPFW and FW-1, but it is the same as using the IP address of the network assigned to the interface directly, providing a new and easily emulable functionality and an improvement in usability, and thus will be considered (one usually knows the IP of an interface, but not the name of the interface). All platforms except PIX permit the use of a wildcard indicating that a rule applies to all interfaces, since PIX rules must be assigned to a specific interface. Note that the use of interfaces for matching is optional in all platforms analyzed except in PIX. This represents a new functionality and a great improvement in usability, since if not considered in the meta-model, the only way to represent a rule for all interfaces is to write the same rule but for each different interface. In addition, the wildcard can be easily emulated in PIX by generating a rule for each interface. Thus for these reasons, we will consider it in the meta-model. Negation will not be considered for the same reasons explained in Source and Destination IP.

- *Interface direction:* The only thing that is different when specifying the direction of the interface in the firewall platforms analyzed is that in PIX and IPFilter the use of direction is mandatory, and that they do not support wildcards. The same is the case for interfaces, as explained before. For the same reasons, wildcards will be incorporated into the meta-model. Again, for the same reasons explained above, negation is not considered.
- *Protocol:* Only IPFilter, IPFW and PF support the use of identifiers, but as explained above they are easily emulable user-centric improvements and thus will be considered. IPFilter supports the specification of TCP plus UDP protocols at the same time, but again it can be decomposed into two rules, thus it is a user facility with no great improvement in ease of use. ICMPv6-based filtering is supported in PF, but its support depends on the matching algorithm implemented in the platform. Protocol lists are also supported in some firewalls, but like other lists, they can be decomposed into several rules. Logic OR/NOT expressions can also be used in IPFW, and negation is also possible as with other selectors in some firewall platforms, but for the same reasons explained above, they will not be considered in the model.
- *Source and Destination Ports:* Many range syntaxes are possible in many firewall platforms, as in the case of ranges ' < p', ' < =p', ' > p', ' > =p', '(p1, p2)' and ')p1, p2('. These syntaxes provide no new functionality or clear improvement in usability, and can be easily emulated with the common "[p1, p2]" syntax without loss of functionality or performance. For this reason we will not include them in the meta-model. Lists of ports, logic OR/NOT expressions and negation are also possible in some platforms, but again we will not consider them for the same reasons previously explained.
- *ICMP type:* IPFW supports lists as a way of representing various types of ICMP traffic, and many languages support negation, but will not be considered in the meta-model for the same reasons. The same reasons explained before are applicable here.
- *Action:* There are two common actions: allow and deny a packet. However, all firewall platforms analyzed except Cisco PIX support a third one, reject, that denies a packet, but acknowledges it with the sender. This is a new functionality that cannot be emulated in PIX, but can be transformed into a deny action without any great loss in functionality (the packet will be denied, the only difference is the acknowledgement). For this reason, packet rejection will be permitted in the meta-model.

Note that some selectors have common syntaxes that do not provide extra functionality. This is the case for the Protocol selector. TCP, UDP, and ICMP identifiers can be removed from the meta-model without loss of functionality, because they are all numbers and thus can be represented using that syntax. This is also the case for the identifier of Source Port, Destination Port, and ICMP Type selectors, as all services can be specified with a number. The removal or not of these selectors is a decision that should be taken only considering the usability of the PIM because, as they are supported by all firewall platforms, their transformation to a low-level language is direct. We have preferred to leave them in the meta-model in order to improve usability.

Bear in mind that, although interfaces are a common selector and a mandatory one in PIX, they represent platform-specific information, and conceptually should not be added to the meta-model. However, interface information can be emulated with ease in the PIM to PSM transformation process: for each rule, check which interface the source IP belongs to. The rule should be added to that interface, in the incoming direction. For that reason, interfaces and directions are optional in the meta-model, and so will be in the proposed PIM.

4.3. PIM specification for firewall ACLs

Table 3 presents the meta-model of the final PIM. No more selectors can be added to this PIM because, as we have explained in Section 4.2, the rest of the non-common selectors cannot be

Table 3
Final PIM meta-model for firewalls

Selector	Obligation	Dependencies	Common syntax (can be optimized)	Non-common syntax (must be emulated)	Comments
Source IP address	Mandatory		–IP –Block (CIDR) –Wildcard	-Identifier	
Destination IP Address	Mandatory		–IP –Block –Wildcard	-Identifier	
Interface	Optional		–Identifier	–Wildcard –IP Address	
Interface Direction	Optional		–In –Out	-Wildcard	IPTables cannot represent direction, but can differentiate inbound from outbound interfaces
Protocol	Mandatory		–TCP –UDP –ICMP –Number –Wildcard*	-Identifier	*IPFilter and PF do not have wildcards, but the omission of this selector has the same effect
Source Port	Optional	Only if Protocol is TCP or UDP	–Number –Range: [p1,p2] –Identifier –Wildcard		
Destination Port	Optional	Only if Protocol is TCP or UDP	–Number –Range: [p1,p2] –Identifier –Wildcard		
ICMP Type	Optional	Only if Protocol is ICMP	–Number –Identifier		
Action	Mandatory		–Allow –Deny	-Reject	Reject cannot be emulated in PIX, but deny can be used instead triggering the acknowledgement

emulated since they depend on the underlying matching algorithm in each firewall platform. However, more syntaxes could be added in order to ease the task of creating the access control policy by a user. The common syntaxes presented in Table 3 have direct support in all firewall platforms analyzed and no optimization or emulation is needed. The other syntaxes have been removed because they do not provide new functionality or a clear improvement in usability. In any case, they can always be optimized in the PIM to PSM transformation using the specific syntax of the underlying firewall platform that each PSM represents.

Comments in the last column present specific considerations for some firewall platforms that will be important in the PIM to PSM transformation process. A PIM using XML Schema is presented in Annexe I.

Although the PIM meta-model presented is very simple, it has been created from an analysis of real firewall languages. The starting point for this meta-model was what these languages have in common, which has then been extended with some non-common parts. Note that simplicity is not a synonym for lack of usability, as is demonstrated by Cisco PIX language, which is the simplest of those analyzed, and also one of the most widely used in the world. As has been explained, most of the syntaxes present in the firewall languages analyzed are only there to provide compactness and to ease the task of writing the rule set by a human. However, we believe that many of these syntaxes are confusing and could easily introduce inconsistencies and redundancies in a rule set. For this reason, they have not been included in the PIM meta-model.

Remember that this PIM meta-model does not contain details for other characteristics of firewall platforms, such as logging and connection tracking, since this is going to be represented in the PSM, as they are execution environment specific details.

PIM transformation into any of the firewall languages analyzed should be very easy and direct, since all the selectors considered are supported in all the firewall platforms analyzed, and all the syntaxes considered for them are also directly supported or are easily emulable. The simpler the low-level target language is, the easier the transformation is. For example, PIM to Cisco PIX language is direct.

4.4. Example

As has been noted before, the PIM has been represented as an XML Schema Definition (XSD). However, it can be represented in other languages as well. In this paper, XML has been used, since it is more widely known than other OMG MDA-specific technologies. Representing the model in XML is an easy task, since the proposed PIM is very simple. However, writing XML manually without the aid of a GUI tool could be a time-consuming task. For illustrative purposes, Fig. 11 presents the example presented in Table 1, using the PIM schema in Annexe II. Note that rule order is implicit in the XML, since rule matching algorithms follow a linear order in firewalls. Thus it is not necessary to represent rule priorities in the model explicitly. Comparing Table 1 with this XML representation, it can be seen how easy it is to represent real-life knowledge in the proposed model.

As explained before, this PIM cannot represent characteristics other than those needed to take filtering decisions (selectors with their syntax and actions). Other characteristics related to how the rule set will be executed once the PIM has been transformed into a PSM (and the PSM into code) must be present in the PSM, since it represents the platform-specific characteristics.

But using XML instead of other languages has several advantages. First, as can be seen from this example, the model is very well structured and is easily understandable, even by a person without any particular knowledge of a specific firewall language, XML, or even the PIM schema. This also improves model modifications. Second, this XML can easily be parsed using available libraries, since most Object-Oriented programming languages have an XML parsing library in their API by default. In addition, it is undefined in the schema which value an optional selector takes if it does not appear in a rule. The parser is responsible for valuating these selectors. In this paper, it is assumed that these selectors receive a wildcard value, as this is the usual solution that most firewall languages adopt if a selector is omitted.

In Section 5, a verification stage will be run against this model in order to diagnose inconsistencies. If inconsistencies are found, PIM modification is necessary, and if the PIM has been correctly derived from the access control requirements, then these requirements require modification.

5. PIM verification: consistency-based diagnosis process

Prior to PSM generation, we propose a mandatory verification stage in order to guarantee that the modelled access control policy is consistent. Error correction in the early stages of the development process is cheaper compared to the cost associated with error correction in the production phase. In addition, errors generated in the production phase usually have a huge impact on the reliability and robustness of the generated code and final system. We do not consider redundancy in this work, since redundancy faults do not change the semantics of the access control policy, but only affect its performance. Carrying out verification at early stages guarantees that resulting PSMs and rule sets will be consistent. More details on the inconsistency diagnosis problem are presented in other works, and in this paper we only present consistency-based diagnosis algorithms. To understand the problem, it is important first of all to review the inconsistencies characterized in the bibliography. A complete characterization has been given by Al-Shaer et al. [11] which includes shadowing, generalization, correlation, and redundancy (all of them except correlation are order-dependent). Although Al-Shaer characterized all inconsistencies, not all of them are considered to be errors, as they can be used to cause desirable effects (i.e. redundancies can be used to improve matching algorithms efficiency). All of these inconsistencies except redundancy are graphically represented in Fig. 12. For the sake of simplicity, only two rule inconsistencies with one selector are represented. Prior to explaining the algorithms, a definition of inconsistency is needed. We shall now formalize a firewall ACL.

- Let RS be a PIM ACL consisting of n rules, $RS = \{R_1, \ldots R_n\}$
- Let $R = <H, Action>$, $H \in N^5$ be a rule, where $Action = \{allow, deny\}$ is its action
- Let $R_j[k], 1 \leqslant k \leqslant n, k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$ be a selector of a rule R_j.
- Let '<' and '>' be operators which define the priority of the rules, where $R_i < R_j$ means that then R_i has greater priority than R_j and its action will be taken first, and vice versa

Definition 1. 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 semantics of the rule set changes if an inconsistent rule is removed.

$$Inconsistent(R_i, RS), 1 \leqslant i \leqslant n \Longleftrightarrow \exists R_j \in RS, 1 \leqslant j \leqslant n, j \neq i \bullet$$
$$R_i[k] \cap R_j[k] \neq \emptyset \wedge R_i[Action] \neq R_j[Action]$$
$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$
Inconsistency of one rule in a RS

$$Inconsistent(R_i, R_j, RS), 1 \leqslant i, j \leqslant n, i \neq j \Longleftrightarrow$$
$$R_i[k] \cap R_j[k] \neq \emptyset \wedge R_i[Action] \neq R_j[Action]$$
$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$
Inconcistency between two rules in a RS

This definition can be extended to more than two rules. According to 1, all inconsistencies represented in Fig. 12 are of the same kind, and we call them inconsistencies. Thus, there are three inconsistencies, represented by the three possible relations between rules: subset, superset, correlation. In the example presented in Table 1, with the traditional definitions given in [11], R4 is shadowed by R3. However, with the proposed inconsistency definition, R3 and R4 are inconsistent (they have a non-empty intersection of all of their selectors, independently of their priority). Again with the traditional definitions, R2 is a generalization of R1 (or R1 is an exception to R2), but with our definition R1 and R2 are inconsistent. Finally, R1 and R3 are correlated according to traditional definitions, but also according to the new one. Therefore, with 1, it is possible to detect all inconsistencies characterized by the traditional definitions. We consider that redundancy is not an inconsistency, because it does not change the semantics of an ACL, and thus, it has been left out of this work as a topic for future research.

The process shown in this section is broken down into two steps. In the first step, all inconsistencies in the PIM are detected and isolated for every pair of rules. In the second step of the process, the rules that cause the inconsistencies are identified, completing the PIM consistency-based diagnosis process. The full process, as will

```
<policy>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.20</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>80</prtdst>
                </matches>
                <action>deny</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.0/24</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>80</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>all</ipsrc>
                        <ipdst>161.120.33.40</ipdst>
                        <prtdst>80</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.0/24</ipsrc>
                        <ipdst>161.120.33.40</ipdst>
                        <prtdst>80</prtdst>
                </matches>
                <action>deny</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.30</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>21</prtdst>
                </matches>
                <action>deny</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.0/24</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>21</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>140.192.37.0/24</ipsrc>
                        <ipdst>161.120.33.40</ipdst>
                        <prtdst>21</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>tcp</protocol>
                        <ipsrc>all</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>all</prtdst>
                </matches>
                <action>deny</action>
        </rule>
        <rule>
                <matches>
                        <protocol>udp</protocol>
                        <ipsrc>140.192.37.0/24</ipsrc>
                        <ipdst>161.120.33.40</ipdst>
                        <prtdst>53</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>udp</protocol>
                        <ipsrc>all</ipsrc>
                        <ipdst>161.120.33.40</ipdst>
                        <prtdst>53</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>udp</protocol>
                        <ipsrc>140.192.38.0/24</ipsrc>
                        <ipdst>161.120.35.0/24</ipdst>
                        <prtdst>all</prtdst>
                </matches>
                <action>allow</action>
        </rule>
        <rule>
                <matches>
                        <protocol>udp</protocol>
                        <ipsrc>all</ipsrc>
                        <ipdst>all</ipdst>
                        <prtdst>all</prtdst>
                </matches>
                <action>deny</action>
        </rule>
</policy>
```

Fig. 11. Table 1 example represented in XML following PIM XSD specifications.

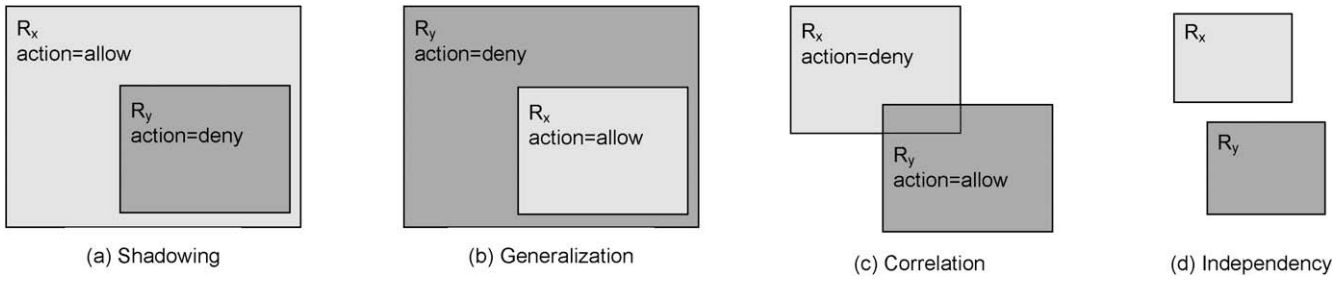

Fig. 12. Graphical representation of three types of inconsistencies in rule sets.

be shown, requires no modification to the PIM, nor any pre-process. It runs in polynomial time and has lineal space complexity with the number of inconsistencies in the rule set. Note that for the examples presented in this section, Interface and Direction selectors have been left out for simplicity. Anyway, the use of these selectors is optional in the PIM, as has been explained in the previous section.

5.1. Step 1. Detection of inconsistent pairs of rules

The first step in the process detects the inconsistent rules in the rule set and returns an Inconsistency Graph (IG, 2) representing their relations. Note that the detection process, like 1, is order-independent.

Definition 2. *Inconsistency Graph, IG.* An IG is an undirected, cyclic and disconnected graph whose vertices are the inconsistent rules of the PIM, and whose edges are the inconsistency relations between these rules. Note that |IG| is the number of inconsistent rules in *RS*, and ||IG|| corresponds to the number of inconsistencies between pairs of rules in *RS*, or simply the number of inconsistencies in *RS*.

$$\text{Let } IG =< V, E > \text{ be an undirected, cyclic and disconnected graph wl}$$
$$V(IG) = R_i \in RS, 1 \leqslant i \leqslant n \bullet Inconsistent(R_i, RS)$$
$$E(IG) = R_i, R_j \in V, 1 \leqslant i, j \leqslant n, i \neq j \bullet Inconsistent(R_i, R_j, RS)$$

Algorithm 1 presented in Fig. 13 (implemented in an Object-Oriented paradigm and using abstract data types) exploits the order-independence of the inconsistency definition and only checks inconsistencies between rules with different actions, dividing the PIM ACL into two lists, one with *allow* rules and the other with *deny* rules. The algorithm receives two rule sets. One of them is composed of *allow* rules and the other of *deny* rules from the original rule set. This decomposition is a linear complexity operation. The algorithm takes one of these two PIM ACLs and, for each rule, checks if there is an inconsistency with other rules in the other ACL. As all inconsistencies can be decomposed into 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 an undirected edge between them. The algorithm returns an IG. As all inconsistencies can be decomposed into two by-two relations, there is no need to check combinations of more than two rules. Since all possibilities have been checked, Algorithm 1 detects all possible inconsistent rules. A trace of the different iterations of Algorithm 1 applied to the model in Fig. 11 is presented in Fig. 14 to illustrate the process. Fig. 15 presents the resulting IG.

The 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 of the rules *allow* and the other half *deny*), and the best

Algorithm 1. Inconsistency Detection algorithm

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

Fig. 13. Algorithm 1. Inconsistency detection algorithm.

case when *ruleSetAllow.size() = n* and *ruleSetDeny.size() = 0* or *ruleSetAllow.size() = 0* and *ruleSetDeny.size() = n*. Thus, the complexity of the improved detection algorithm depends on the percentage of *allow* and *deny* rules out of the total number of rules. However, there are other internal operations that should be analyzed in lines 11–14. The first one, in line 11, is *inconsistency()* which is composed of an iteration. This operation implements the inconsistency definition. In typical firewall ACLs, $k = 5$, and thus the iteration runs 5 times. In any case, the iteration is bounded by the number of selectors, which is a constant, k. In addition, inside the iteration there is an intersection between each selector (lines 27–29). The typical 5 selectors in firewall ACLs (Table 1) are integers or ranges of integers, except IP address. Whether two ranges of integers intersect can be determined in constant time using a naïve algorithm which compares the limits of the intervals. Whether two IP addresses intersect can also be determined easily in constant time

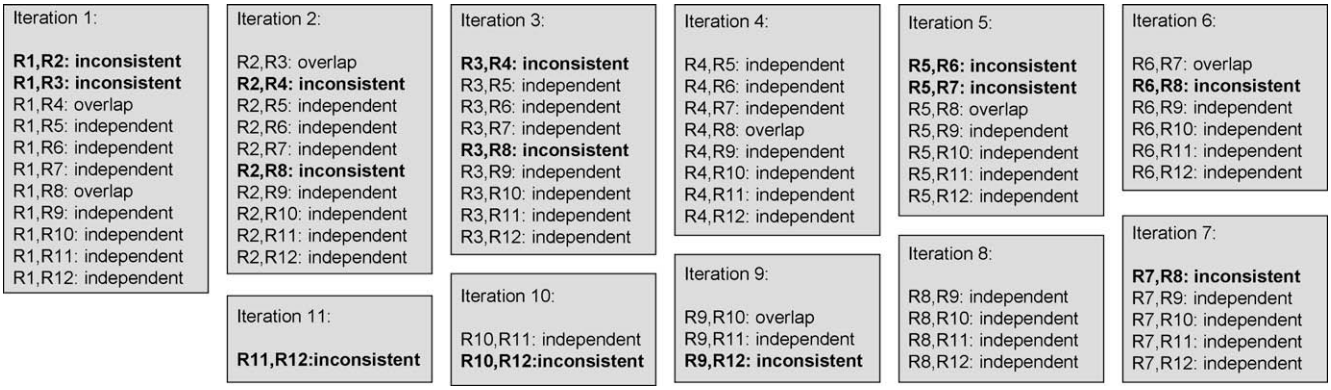

Fig. 14. Trace of Algorithm 1 applied to the model in Fig. 10.

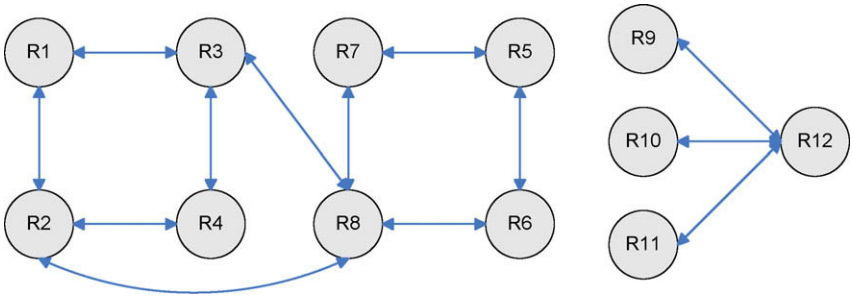

Fig. 15. Inconsistency Graph, IG.

by comparing their network addresses and netmasks. Other operations in the inner loop (lines 11–13) are graph-related. If the graph implementation is based on hash tables, vertex and edge insertions run in constant time, except in some cases where rehashing might be necessary.

For all these reasons, in all cases the complexity of the two nested loops is only affected by a constant factor, which depends on the constant number of selectors, k. Thus, worst-case time complexity of the detection 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.

The space used by Algorithm 1 is the sum of the space needed to store the ACL, and that needed for the graph. In the 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 less than that needed to store a vertex, since only a reference between vertices is needed.

5.2. Step 2. Identification of the set of conflicting rules

The second and final stage in the diagnosis process identifies the PIM rules that cause the inconsistencies from the set of inconsistent pairs of rules (the result of the previous step). Algorithm 2 (Fig. 16) receives the IG as input and iteratively takes the vertex with the greatest number of adjacencies (lines 5 and 6), that is, the vertex with the greatest number of inconsistencies, v. Then, an independent cluster of inconsistent rules (ICIR, 3) is created as a tree with v (the conflicting rule in the cluster) as its root, and its adjacents (the inconsistent rules) as leaves (lines 7–11). The root of all ICIRs from the Diagnosis Set (DS, 4), is the set of rules that should be removed to obtain a consistent PIM. Then, v and its edges are removed from the IG (line 12). If vertices with no edges are left in the IG, then these vertices are removed (line 13), since they are consistent by definition (they are rules with no relations to others). As inconsistencies have been decomposed in pair wise relations, ICIRs are always formed on two levels.

Algorithm 2. Inconsistent Rule Identification algorithm

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

Fig. 16. Algorithm 2. Determination of the Diagnosis Set.

Definition 3. *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 *root*. *ICIR*(*root*) is the rule which has the greatest number of inconsistencies with other rules of the same cluster. Note that the action *ICIR*(*root*) is the opposite of the actions of all of its leaves in *CV*.

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

Definition 4. *Diagnosis Set, DS.* This is the set of PIM rules that cause the inconsistencies, and coincide with the root of all ICIRs.

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

A graphical representation of a partial trace of Algorithm 2 from the previous IG is presented in Fig. 17. During the first iteration, R8 was selected because it had four inconsistencies (the greatest number of adjacent vertices). It was removed to form the first ICIR tree, with R8 as root, thus R8 was an inconsistent rule. In the second iteration, R12 was selected because it had three inconsistencies (it is the vertex with the largest number of adjacent vertices). Then it was removed and the second ICIR was formed. Vertices R9, R10, and R11 were removed from the IG because they had no adjacent vertices. In the third iteration, R5, R1, R2, R3, and R4 could be selected as the next vertex. The selection of one or other is arbitrary. In this example, the algorithm selected R5, removed it from the IG with all its edges and formed the third ICIR. At the end of this iteration the IG was only composed of a cycle of four vertices: R1, R2, R3, and R4. The algorithm chose 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 IG, the algorithm finished with a Diagnosis Set of cardinality five (|DS| = 5), containing the rules DS = {R8, R12, R5, R1, R4}, which are the ICIR roots or the PIM rules that cause an inconsistency with others. R8 and R12 were the most conflicting.

Now there are two possibilities to give to the engineer. First, these five rules can be removed from the PIM in order to obtain a consistent model (this can be easily automated). Second, the ICIR rules could be checked manually in the PIM in order to remove the inconsistency. Requirements (CIM) should also be checked to discover whether the inconsistency comes from the requirements engineering or from the PIM modelling steps.

The time complexity of Algorithm 2 is bounded by the loop of line 4, which runs as many times as ICIRs are formed (it corresponds to the cardinality of the Diagnosis Set, $|DS|$). The worst-case is reached, as in Algorithm 1, when *ruleSetAllow.size()* = *ruleSetDeny.size()* = $n/2$, resulting in a $|DS| = n/2$. In this case, *getMaxAdjacencyVertex()* (line 7), a maximum calculation, runs in $O(n)$ with the number of vertices in the graph (the number of inconsistencies). Operations in lines 7–11 run in constant time. *removeVertexWithEdges()* (line 12) runs in linear time with the cardinality of its adjacency list ($n/2 - 1$ in the worst-case). Finally, *removeDisconnectedVertices()* (line 13) 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 $O(|DS| \cdot (n + n/2 - 1 + n)) = O(n/2 \cdot n) = O(n^2)$.

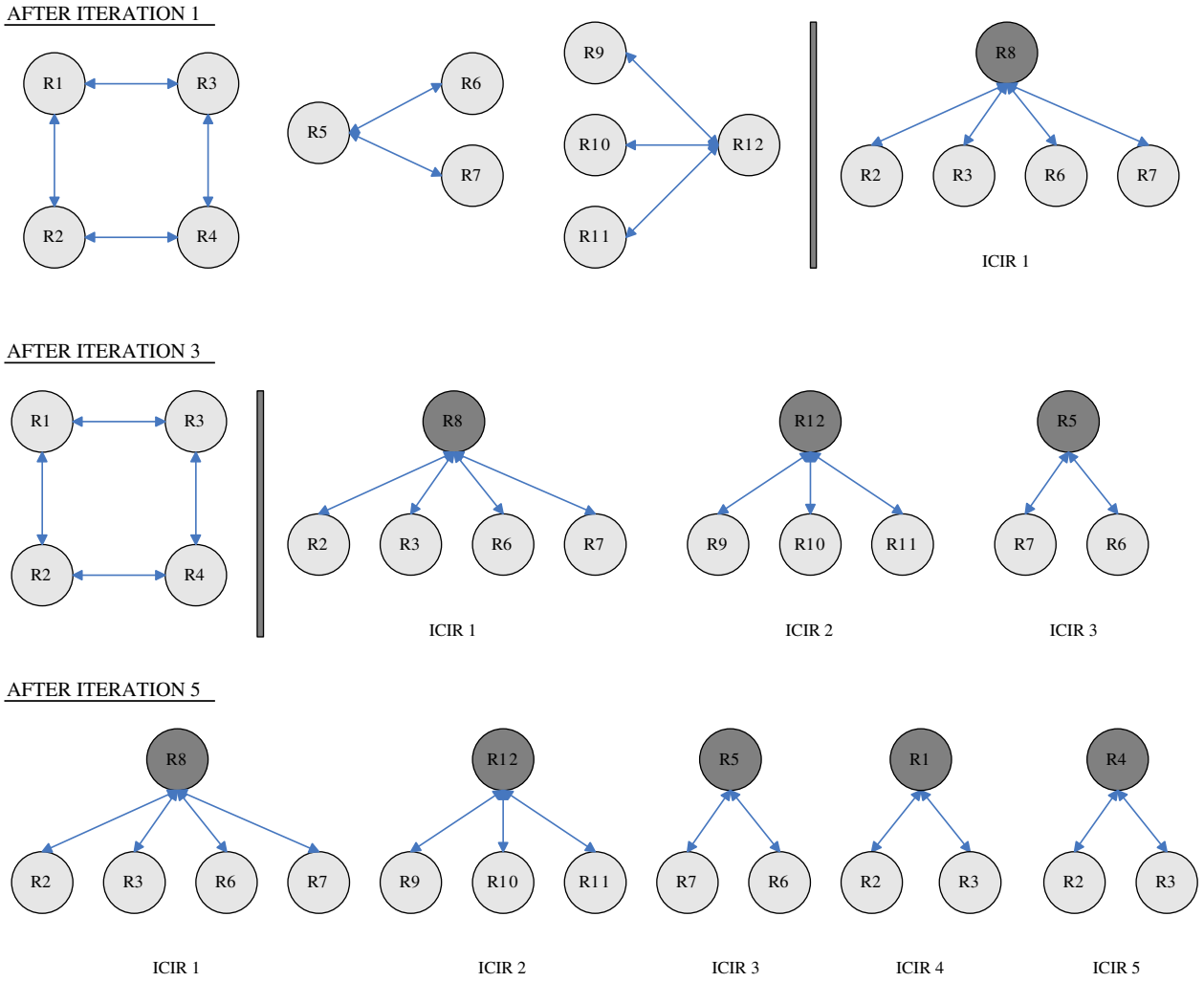

Fig. 17. Trace of Algorithm 2 applied to Fig. 15 IG.

The best case is reached, as in Algorithm 1, when *ruleSetAllow.size()* = *n* and *ruleSetDeny.size()* = 0 or vice versa. The IG has only 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 $O(|DS| \cdot h)$, $|DS| \ll h$ (h is the number of inconsistencies).

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 creates the ICIRs, the corresponding vertices and edges are removed from the IG, and thus at each iteration the only space needed is to store the adjacency list for the removed vertex.

Note that the described validation step is static, since it is run prior to PIM to PSM transformation. Verification was used to detect and diagnose inconsistencies in the first stage of the process, limiting the propagation of these problems to later phases. If incomplete, imprecise or inconsistent models were not detected during the PIM construction stage, there would be propagation of these problems to the executable code.

5.3. Experimental results

In order to test the efficiency of the proposed algorithms empirically, and to discover their suitability for real environments, several tests were conducted. These tests use real PIMs taken directly from production rule sets in an inverse engineering approach. These rule sets were taken from IPTables rule sets using a Java parser and transformation tool which is available upon request. The measured time does not take into account XML PIM parsing time, but only inconsistency diagnosis. Experiments were performed on a Java implementation with Sun JDK 1.6.0_03 64-Bit Server VM, on an isolated HP Proliant 145G2 (AMD Opteron 275 2.2 GHz, 2 Gb RAM DDR400). Execution times are in ms.

The performance analysis conducted represents a wide spectrum of cases, with ACLs ranging in size from 50 to 5000 rules, and percentages of allow and deny rules from 2% to 65%. Remember that the worst-case is obtained when a rule set has 50% allow and 50% deny rules. In real firewall ACL models, the number of deny and allow rules will be very different. As real firewall ACLs are usually designed with a deny-all default policy, most rules will have allow actions. Note that as the percentage of allow or deny rules decreases, the number of inconsistencies does not necessarily, because the number of inconsistencies depends on how many rules with different actions intersect and not on the number of rules with allow or deny actions. The result is that the worst-case would not normally happen in real firewall ACLs, and thus experimental results are nearly the best case.

Results show (Fig. 18) that the time is under 70 ms for models of 5000 rules or less. For models around the sizes used by Al-Shaer and García-Alfaro in their examples, that is between 0 and 80 rules, the time is between 0 (0 rules) and 2 ms (80 rules). Even for rule sets with 800 rules (used in Fireman), the time is under 15 ms.

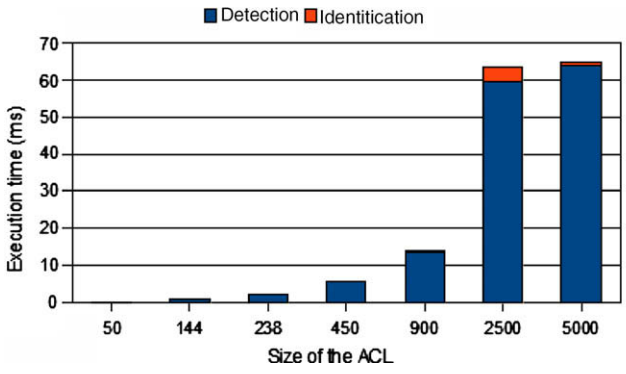

Fig. 18. Performance evaluation with real models.

As real rule sets have been used, these graphs represent average cases. We think that the theoretical complexity analysis and the empirical results presented prove the feasibility of the proposal in real environments.

6. Conclusions and future works

In this paper, two of the traditional firewall rule set problems have been revisited: design and consistency. We have analyzed different solutions to each of these problems, and proposed a new approach based on Model-Based Development ideas (see Fig. 19).

In this new approach, access control requirements are obtained from a requirements engineering phase, forming a Computation Independent Model (CIM). This model is then translated in a manual process into a Platform Independent Model for firewall access control lists. Then, this PIM can be translated into different Platform-Specific Models, each one representing an underlying firewall platform. Finally, rule sets can be generated directly from these PSMs. This paper is focuses only on the PIM model. Details relating to how each particular firewall platform executes the resulting rule set should be represented in the PSM, which is not the focus of the paper.

We have shown that one of the disadvantages of the MBD process is the automatic generation of code without prior verification or validation (V&V) of models, because the generated code will carry all errors the models might possibly have. We have proposed the inclusion of a mandatory automatic static verification phase in the MBD process for firewalls. In this phase, the PIM is automatically diagnosed for inconsistencies before transformation to PSM and then to code. If inconsistencies are detected, then correction of the model can be performed in the PIM or in the CIM. Error correction in early stages of the MDB process is cheap compared to the cost associated with correcting errors in the production phase. In addition, errors produced in the production phase usually have a huge impact on the reliability and robustness of the generated code and final system. In the proposed MBD approach for firewalls, the resulting rule set will be consistent.

We have analyzed several widely used firewall languages in order to acquire knowledge about the possible filtering selectors that can be used in the condition part of a rule, and the possible syntaxes that can be used by end-users. We have discussed the possibilities of a bottom-up approach to building a PIM: factorization, aggregation, and customization. We have opted for the customization approach, since it is the most flexible. Then we have proposed a PIM and a XML Schema Definition for it.

Finally, we have proposed a method to diagnose the inconsistent rules in the PIM. We have given two algorithms that have a combined worst-case quadratic time complexity. However, complexity is never near the worst case in real firewall ACLs. A theoretical complexity analysis has been provided, and empirical results with models ranging from 0 to 5000 rules have been provided in order to test the feasibility of our proposal in real environments.

To the best of our knowledge, this is the first time that the problem of automatic rule set generation for firewall rule sets has been modelled on a structured Model-Based Development approach. Our proposal also includes a verification stage in order to diagnose consistency faults in early stages of the process, which is also to the best of our knowledge, the first time that error detection has been moved to early stages in automatic firewall rule set generation.

In future works the MBD process should be completed. An analysis of firewall platforms is needed in order to construct the PSMs. Redundancy detection and validation techniques could also be used in order to improve the quality of the resultant rule set. In addition, the use of UML as a model to represent access control requirements and automatic transformation into the PIM should be considered, since it could improve initial PIM quality.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="policy">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="rule" minOccurs="1" maxOccurs="unbounded">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element name="matches" type="matchesType"/>
                                                        <xs:element name="action" type="actionType"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:complexType name="matchesType">
                <xs:all>
                        <xs:element name="interface" type="interfaceType" minOccurs="0"/>
                        <xs:element name="direction" type="directionType" minOccurs="0"/>
                        <xs:element name="ipsrc" type="ipType"/>
                        <xs:element name="ipdst" type="ipType"/>
                        <xs:element name="protocol" type="protoType"/>
                        <xs:element name="prtsrc" type="portType" minOccurs="0"/>
                        <xs:element name="prtdst" type="portType" minOccurs="0"/>
                        <xs:element name="icmptype" type="icmptType" minOccurs="0"/>
                </xs:all>
        </xs:complexType>
        <xs:simpleType name="actionType">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="allow"/>
                        <xs:enumeration value="deny"/>
                        <xs:enumeration value="reject"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="interfaceType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="([0-2]?[0-9]?[0-9]\.[0-2]?[0-9]?[0-9]\.[0-2]?[0-9]?[0-9]\.[0-2]?[0-
9]?[0-9])|([a-zA-Z0-9])+"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="directionType">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="in"/>
                        <xs:enumeration value="out"/>
                        <xs:enumeration value="both"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="ipType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="(([0-2]?[0-9]?[0-9]\.[0-2]?[0-9]?[0-9]\.[0-2]?[0-9]?[0-9]\.[0-
2]?[0-9]?[0-9])(/[0-3]?[0-9])?)|([a-zA-Z0-9])+"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="protoType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="tcp|udp|icmp|([a-zA-Z])+|([0-2]?[0-9]?[0-9])"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="portType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="(([0-6]?[0-9]?[0-9]?[0-9]?[0-9])(:[0-6]?[0-9]?[0-9]?[0-9]?[0-
9])?)|([a-zA-Z])+"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="icmptType">
                <xs:restriction base="xs:string">
                        <xs:pattern value="([0-2]?[0-9]?[0-9])|([a-zA-Z])+"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

Fig. 19. PIM XML Schema Definition.

Acknowledgements

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 the anonymous reviewers for their useful comments, and to Juan Rodríguez Feria, for his input on firewall languages and the IPTables parser.

Annex I. Comparative analysis of firewall filtering selectors

This analysis is related to the condition part of a rule, and refers to the conditions that may be matched against a packet that arrives at the firewall, or the selectors used for the condition part of the rule. Each firewall language has a different syntax for each selector, and permits a different number and types of them. The analysis is divided in several sections, which analyze one aspect of firewall filtering. The analysis of actions to be performed on a packet is also included.

I.1. Source and Destination addresses (Table I.1)

The left column represents the syntax that each firewall language can use, as follows:

- IP. IP address in canonical form a1.a2.a3.a4, where each ax is an integer in [0..255]
- Identifier. A variable name with an assigned value (with permitted syntax)
- Block. Block of IP addresses in CIDR format
- Range. Range of continuous IP addresses, in the form o1.o2.o3.o4-a1.a2.a3.a4
- Domain. A domain name
- Table. A table name. The contents of the table is a list of IP addresses
- List. A list name representing a list of IP addresses
- Logic OR/NOT expression. A list of IP addresses composed by OR–NOT logic operators
- Interface. The name of a network interface of the underlying platform, referring to the block of IP addresses statically assigned to that interface
- Block interface. The name of a network interface of the underlying platform, referring to the block of IP addresses statically assigned to that interface, plus a CIDR netmask
- DHCP interface. The name of a network interface whose IP address is obtained via DHCP
- Any IP. This is a wildcard '*' representing all IP space
- Negation. A modification to one of the previous values, representing the opposite of the specified in the value
- Default value. If permitted to be unspecified, the default taken by this selector is the wildcard '*'

I.2. Interface (Table I.2)

This selector represents the interface which the packet is coming from or is going through. The left column represents the syntax that each firewall language can use as follows:

- Identifier. A variable with the name of an interface used by the platform
- Group identifier. A variable name with an assigned value, representing a group of IP interfaces
- IP Address. The IP address of the network assigned to an interface

I.3. Interface direction (Table I.3)

The left column represents the syntax that each firewall language can use, as follows:

- In and Out. Represent incoming traffic or outgoing traffic

I.4. Protocol (Table I.4)

I.5. Source and Destination ports (Table I.5)

These two selectors are only valid when protocol is TCP or UDP. Destination port represents the port of the service, and source port represent the opened port at the sender.

Table I.1
Source and Destination addresses syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Src/Dst IP	Optional	Mandatory	Optional	Mandatory	Optional	Optional
IP	√	√	√	√	√	√
Identifier						√
Block	√	√	√	√	√	√
Range	√					√
Domain			√	√	√	√
Table					√	
List					√	√
Logic OR/NOT expression				√		
Interface					√	
Block interface					√	
DHCP interface					√	
Negation	√			√	√	√
Wildcard	√	√	√	√	√	√
Default value	*Any ip*		*Any ip*		*Any ip*	*Any ip*

Table I.2
Interface syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Interface	Optional	Mandatory	Optional	Optional	Optional	Optional
Identifier	√	√	√	√	√	√
Group identifier	√		√	√	√	
List					√	
IP address				√		√
Negation	√			√	√	
Wildcard	√		√	√	√	√
Default value	Any interface		Any interface	Any interface	Any interface	Any interface

Table I.3
Interface direction syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Direction	Implicit	Mandatory	Mandatory	Optional	Optional	Optional
In	√	√	√	√	√	√
Out	√	√	√	√	√	√
Negation				√		
Wildcard	√			√	√	√
Default value	Any direction			Any direction	Any direction	Any direction

Table I.4
Protocol syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Protocol	Optional	Mandatory	Optional	Mandatory	Optional	Optional
Number	√	√	√	√	√	√
Identifier			√	√	√	
TCP	√	√	√	√	√	√
UDP	√	√	√	√	√	√
TCP/UDP			√			
ICMP	√	√	√	√	√	√
ICMP6					√	
List	√				√	√
Logic OR/NOT expression				√		
Negation	√			√		√
Wildcard	√	√	√	√	√	√
Default value	Any protocol		Any protocol		Any protocol	Any protocol

I.6. TCP flags (Table I.6)

This selector is used if the matching engine supports inspection of TCP flags of packets. Of course, this only works if protocol is TCP. TCP flags are SYN, ACK, FIN, RST, URG, and PSH. It is possible to match all flags or none of them.

I.7. TCP options (Table I.7)

This selector is also only used if protocol is TCP, and if the underlying engine supports matching against TCP Options.

Table I.7
TCP options syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
TCP options	Optional			Optional		
Number	√			√		
List				√		
Negation	√			√		
Default value	Do not inspect			Do not inspect		

Table I.5
Source and Destination port syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Src/Dst port	Optional	Optional	Optional	Optional	Optional	Optional
Number	√	√	√	√	√	√
Identifier	√	√	√	√	√	
Range $\neq p$	Using negation	√	√	Using negation	√	Using negation
Range $< p$	Using negation	√	√		√	√
Range $> p$	Using negation	√	√		√	√
Range $\leqslant p$	√		√		√	Using negation
Range $\geqslant p$	√		√		√	Using negation
Range [*p1,p2*]	√	√	√	√	√	√
Range (*p1,p2*)			√		√	
Range)*p1,p2*(	Using negation		√	Using negation	√	Using negation
Block				√		
List	√			√	√	
Logic OR/NOT expression				√		
Negation	√			√		√
Wildcard	√	√	√	√	√	√
Default value	Any port	Any port	Any port	Any port	Any port	Any port

Table I.6
TCP flags syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
TCP flags	Optional		Optional	Optional	Optional	
Active or inactive flags	√		√	√	√	
Negation	√			√		
Default value	Do not inspect		Do not inspect	Do not inspect	Do not inspect	

Table I.8
ICMP type syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
ICMP type	Optional	Optional	Optional	Optional	Optional	Optional
Number	√	√	√	√	√	
Identifier	√	√	√	√	√	√
List				√		
Negation	√			√		
Default value	Do not inspect	Do not inspect	Do not inspect	Do not inspect	Do not inspect	Do not inspect

I.8. ICMP type (Table I.8)

This selector is used if protocol is ICMP. As all analyzed firewall platforms support ICMP, then this selector can be used by all of them in order to select the type of the ICMP message.

I.9. IP options (Table I.9)

This selector is also only used if protocol is IP, and if the underlying engine supports matching against IP Options.

I.10. IP version (Table I.10)

PF firewall supports filtering of IPv4 and IPv6 protocols.

I.11. Source and Destination MAC addresses (Table I.11a and I.11b)

I.12. Type of Service (ToS) (Table I.12)

Some platforms support the use of ToS for filtering: Minimize-Delay, Maximize-Throughput, Maximize-Reliability, Minimize-Cost, and Normal-Service.

I.13. Time to Live (TTL) (Table I.13)

I.14. Actions (Table I.14)

Some firewall platforms permit packet rejection with and without acknowledgement to the sender. Reject does send ACK, but

Table I.11a
Source MAC address syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Source MAC	Optional			Optional		
MAC	√			√		
Any MAC				√		
Negation	√			√		
Default value	Any MAC			Any MAC		

Table I.11b
Destination MAC address

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Destination MAC				Optional		
MAC				√		
Any MAC				√		
Negation				√		
Default value				Any MAC		

Table I.12
Type of Service syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
ToS	Optional		Optional	Optional	Optional	
Number	√		√	√	√	
HEX number	√		√	√	√	
Identifier	√		√	√	√	
List				√		
Negation	√			√		
Default value	Any ToS		Any ToS	Any ToS	Any ToS	

Table I.9
IP options syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
IP options				Optional		
ssrr				√		
lsrr				√		
rr				√		
ts				√		
List				√		
Negation				√		
Default value				Do not inspect		

Table I.10
IP version syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
IP version					Optional	
IPv4					√	
IPv6					√	
Negation						
Default value					Automatic	

Table I.13
Time to Live syntax

	iptables	PIX	ipfilter	ipfw	pf	Fw-1
TTL	Optional		Optional	Optional		
Number	√		√	√		
Range: < n	√					
Range: > n	√					
Range: [ttl1, ttl2]				√		
List				√		
Negation	√			√		
Default value	Any TTL		Any TTL	Any TTL		

Table I.14
Action syntax

Actions	iptables	PIX	ipfilter	ipfw	pf	Fw-1
Allow	√	√	√	√	√	√
Deny	√	√	√	√	√	√
Reject	√		√	√	√	√

deny does not. The only platform that does not support rejection with ACK is Cisco PIX.

Annex II. Firewall PIM represented as XML Schema

The PIM is composed of a list of Condition/Action rules. The PIM should have at least one rule. This is because there is a required rule at the end of the PIM that represents the default policy that should be taken if no match is found in the PIM.

The condition part of the rule uses the selectors presented in Table 3, which also have specific syntaxes. These selectors are Source and Destination IP (mandatory), Source and Destination Ports (only if protocol is TCP or UDP), Protocol (mandatory), ICMP Type (only if protocol is ICMP), and finally interface and direction (optional).

References

[1] E. Zwicky, S. Cooper, D.B. Chapman, Building Internet Firewalls, second ed., O'Reilly & Associates Inc., 2000.
[2] W. Cheswick, S. Belovin, A.D. Rubin, Firewalls and Internet Security, second ed., Addison-Wesley, 2003.
[3] A. Wool, A quantitative study of firewall configuration errors, IEEE Computer 37 (6) (2004) 62–67.
[4] Y. Bartal, A. Mayer, K. Nissim, A. Wool, Firmato: a novel firewall management toolkit, ACM Transactions on Computer Systems 22 (4) (2004) 381–420.
[5] N. Damianou, N. Dulay, E. Lupu, M Sloman, The Ponder Specification Language, Workshop on Policies for Distributed Systems and Networks (POLICY), HP Labs Bristol, UK, 2001, pp. 29–31.
[6] OASIS eXtensible Access Control Markup Language (XACML). Available from: <http://www.oasis-open.org/committees/xacml/>.
[7] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, Policy Core Information Model (PCIM), IETF RFC 3060, 2001.
[8] Rule Markup Language (RuleML). Available from: <http://www.ruleml.org/>.
[9] Simple Rule Markup Language (SRML): A General XML Rule Representation for Forward-chaining Rules, ILOG S.A, 2001.
[10] E. Al-Shaer, Hazem H. Hamed, Modeling and management of firewall policies, IEEE eTransactions on Network and Service Management 1 (1) (2004).
[11] H. Hamed, E. Al-Shaer, Taxonomy of conflicts in network security policies, IEEE Communications Magazine 44 (3) (2006) 134–141.
[12] M. Condell, L. Sánchez, On the Deterministic Enforcement of Unordered Security Policies, BBN Technical Memorandum 1346, 2004.
[13] F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro, Detection and Removal of Firewall Misconfiguration, in: IASTED International Conference on Communication, Network and Information Security (CCNIS), Phoenix, ACTA Press, AZ, USA, 2005.
[14] 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 (2007) 1615–5262 (Online).
[15] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, P. Mohapatra, FIREMAN: A Toolkit for FIREwall Modelling and Analysis, in: IEEE Symposium on Security and Privacy (S&P), Oakland, CA, USA, 2006, pp. 199–213.
[16] David E. Taylor, Survey and taxonomy of packet classification techniques, ACM Computing Surveys 37 (3) (2005) 238–275.
[17] De Capitani di Vimercati, S. Foresti, S. Jajodia, P. Samarati, Access control policies and languages, International Journal of Computational Science and Engineering 3 (2) (2007).
[18] C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, P. Samarati, XML-based Access Control Languages, Elsevier Information Security Technical Report, 2004, 35–46 (Online).
[19] Adel El-Atawy, Survey on the Use of Formal Languages/Models for the Specification, Verification, and Enforcement of Security Policies, DePaul University Technical Reports CTI 06-005, 2006.
[20] Firewall Builder. Available from: <http://www.fwbuilder.org/>.
[21] J. Guttman, Filtering postures: local enforcement for global policies, in: IEEE Symposium on Security and Privacy (S&P), Oakland, CA, USA, 1997, pp. 120–129.
[22] J. Guttman, A. Herzog, Rigorous automated network security management, Springer-Verlag International Journal of Information Security 4 (1–2) (2005) 29–48.
[23] A. Mayer, A. Wool, E. Ziskind, Offline firewall analysis, Springer-Verlag International Journal of Information Security 5 (3) (2005) 125–144.
[24] T.E. Uribe, S. Cheung, Automatic analysis of firewall and network intrusion detection system configurations, in: ACM Workshop on Formal Methods in Security Engineering (FMSE), Washington, DC, USA, 2004, pp. 66–74.
[25] S. Hazelhurst, A. Attar, R. Sinnappan, Algorithms for improving the dependability of firewall and filter rule lists, in: Workshop on Dependability of IP Applications, Platforms and Networks (in IEEE Proceedings of International Conference on Dependable Systems and Networks, DSN), New York, NY, USA, 2000, pp. 576–585.
[26] S. Pozo, R. Ceballos, R.M. Gasca, CSP-based firewall rule set diagnosis using security policies, in: International Symposium on Frontiers in Availability, Reliability and Security (FARES) (in International Conference on Availability, Reliability and Security, ARES), IEEE Computer Society Press, Vienna, Austria, 2007, pp. 723–729.
[27] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Transactions on Computers 45 (9) (1996) 993–1002.
[28] D. Senn, D. Basin, G. Caronni, Firewall conformance testing, in: IFIP International Conference on Testing of Communicating Systems, vol. 3502, Springer-Verlag LNCS, Montreal, Canada, 2005, pp. 226–241.
[29] T. Lodderstedt, D. Basin, J. Doser, SecureUML: a UML-based modeling language for model-driven security, in: Fifth International UML, vol. 2460, Springer-Verlag LNCS, Dresden, Germany, October, 2002, pp. 426–441.
[30] D. Basin, J. Dorser, T. Lodderstedt, Model driven security: from UML models to access control infrastructures, ACM Transactions on Software Engineering and Methodology 15 (1) (2006) 39–91.
[31] J. Jürjens, UMLsec: Extending UML for secure systems development, in: Fifth International UML, vol. 2460, Springer-Verlag LNCS, Dresden, Germany, 2002, pp. 1–9.
[32] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, C. Pitcher, Specifications of a high-level conflict-free firewall policy language for multi-domain networks, ACM Symposium on Access Control Models and Technologies (SACMAT), Sophia Antipolis, France, 2007. pp. 185–194.
[33] E. Hooper, Intelligent autonomic strategy to attacks in network infrastructure protection: feedback methods to IDS using policies, alert filters, and firewall packet filters for multiple protocols, in: IEEE International Symposium on Dependable, Autonomic and Secure Networking (DASC), Indianapolis, Indiana, USA, 2006, pp. 235–242.
[34] K. Golnabi, R. Min, L. Khan, E. Al-Saher, Analysis of firewall policy rule using data mining techniques, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), Vancouver, Canada, 2006, pp. 305–315.
[35] P. Douglas, G. Alliger, R. Goldberg, Client-server and object-oriented training, IEEE Computer 9 (6) (1996) 80–84.
[36] O. Pastor, J.C. Molina, Model-Driven Architecture in Practice, Springer Verlag, 2007.