# Model-driven Extraction and Analysis of Network Security Policies

Salvador Martínez[1], Joaquin Garcia-Alfaro[3], Frédéric Cuppens[2], Nora Cuppens-Boulahia[2] and Jordi Cabot[1]

[1] AtlanMod, École des Mines de Nantes - INRIA, LINA, Nantes, France
{salvador.martinez_perez, jordi.cabot}@inria.fr
[2] Télécom Bretagne; LUSSI Department Université Européenne de Bretagne, France
forename.surname@telecom-bretagne.eu
[3] Télécom SudParis; RST Department CNRS Samovar UMR 5157, Evry, France
joaquin.garcia_alfaro@telecom-sudparis.eu

**Abstract.** Firewalls are a key element in network security. They are in charge of filtering the traffic of the network in compliance with a number of access-control rules that enforce a given security policy. In an always-evolving context, where security policies must often be updated to respond to new security requirements, knowing with precision the policy being enforced by a network system is a critical information. Otherwise, we risk to hamper the proper evolution of the system and compromise its security. Unfortunately, discovering such enforced policy is an error-prone and time consuming task that requires low-level and, often, vendor-specific expertise since firewalls may be configured using different languages and conform to a complex network topology. To tackle this problem, we propose a model-driven reverse engineering approach able to extract the security policy implemented by a set of firewalls in a working network, easing the understanding, analysis and evolution of network security policies.

## 1 Introduction

Firewalls, designed to filter the traffic of a network with respect to a given number of access-control rules, are key elements in the enforcement of network security policies.

Although there exist approaches to derive firewall configurations from high-level network policy specifications[18,4], these configuration files are still mostly manually written, using low-level and, often, vendor-specific rule filtering languages. Moreover, the network topology, that may include several firewalls (potentially from different vendors), may impose the necessity of splitting the enforcement of the global security policy among several elements. Due to the complexity of the process, it is likely that we end up with differences between the implemented policy and the desired one. Moreover, security policies must be often updated to respond to new security requirements, which requires evolving the access-control rules included in the firewall configuration files.

Therefore, there is a clear need of an easy way to represent and understand the security policy actually enforced by a deployed network system. At the moment, this still requires a manual approach that requires, again, low-level and vendor-specific expertise. Given a network system consisting in several firewalls configured with hundreds of rules, the feasibility of this manual approach could be seriously questioned. While the security research community has provided a plethora of works dealing with the reasoning on security policies, succeeding at providing a good analysis and verification of the low-level firewall rules, we believe they fail at obtaining a comprehensive solution as they do not provide a high-level, easy to understand and manage representation nor take, generally, networks composed by several heterogeneous firewalls into account. Moreover, the extraction step is often neglected and the solution presented over synthetic rules without providing the means to bridge the gap between them and the real configurations.

In this sense, we believe that an integrated solution is missing. We believe such a solution must have the following features. First, it has to provide independence from the concrete underlying technology, so that the focus can be put into the security problem and not in implementation mechanisms like chains, routing tables, etc. Second, it has to provide a higher-level representation so that the policy becomes easier to understand, analyse and manipulate. Third, the solution, to be comprehensive, must take into account the contribution of each policy enforcing element (firewall) to the global policy, as the partial picture given by isolated firewalls does not provide enough information to understand the network policy.

In this joint work between the modeling and the security communities, we propose a model-driven approach aimed at covering this gap. Our approach, first, extracts and abstracts the information of each firewall configuration file to models conforming to a Platform-independent metamodel specially tailored to represent network-access control information in an efficient and concise way. Then, after performing structural verification of the information in the individual models, it combines these models to obtain a centralised view of the security policy of the whole network. Finally, this global network access-control model can be analysed and further processed to derive useful information. As en example, we analyse the structure of its contents to derive the network topology the firewalls operate on. Then, we provide a mapping to obtain a representation of the policy in XACML, a standardised access-control model, enabling the (re)use of the many tools developed to work with the standard.

We validate the feasibility of our approach by providing a prototype implementation working for firewalls using the netfilter iptables and Cisco PIX rule filtering languages. Our prototype can be easily extended to work with any other packet-filtering languages.

The rest of the paper is organized as follows. Section 2 presents a motivating and running example of a network. In Section 3 we present and detail our approach whereas in Section 4 we present some application scenarios. In 5 we

discuss a prototype implementation. Section 6 discusses related work. The paper finishes in Section 7 with some conclusions and future works.

## 2 Motivation

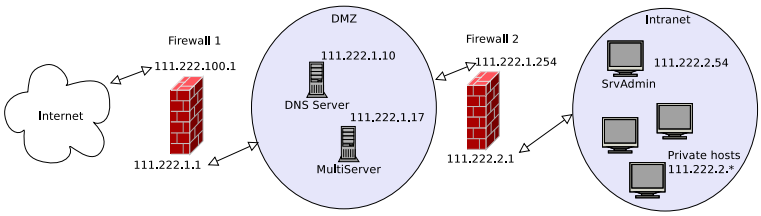


**Fig. 1.** Network example

In order to motivate our approach, we present here a network example that will be used through the rest of the paper.

Let us consider we have a De-Militarized Zone (DMZ) network architecture like the one depicted in Figure 1. This is a very common architecture used to provide services both to a local network and to the public untrusted network while preventing the local network to be attacked. It is composed by the following elements:

- An intranet composed by a number of private hosts where one of the private hosts acts as an administrator of certain services provided by the network system.
- A DMZ that contains two servers. A DNS server and a multiserver providing the following services: HTTP/HTTPS (web), FTP, SMTP (email) and SSH.
- Two firewalls controlling the traffic towards and from the DMZ. The first firewall controls the traffic between the public hosts (the Internet) and the services provided by the DMZ. The second firewall controls the traffic between the intranet and the DMZ.

The two firewalls in charge of enforcing the security policy of our example network, could be of the same kind. However, following the *defense in depth*[1] security strategy, it is highly recommended, to use two different firewalls so that a possible vulnerability does not affect the whole network. In our example, the *firewall 1* is a linux iptables packet-filtering firewall whereas *firewall 2* is a Cisco firewall implementing Cisco PIX filtering.

In Listing 1.1 we show an excerpt of the configuration file of *firewall 1* wrt the HTTP and SMTP services. It controls the traffic from the public hosts to the services provided in the DMZ. This sample configuration uses the netfilter iptables[19] rule language. Note that this configuration file is written using the

iptables *custom chains* feature, which allows the user to define exclusions to rules without using drop or deny rules.

First, it states in the first three lines that the global policy for the firewall is the rejection of any connection not explicitly allowed. Then, the first chain controls the outcoming SMTP messages towards the public host. It allows them for every hosts but for the host in the local network. The second chain controls the incoming SMTP messages to the server. If the request is done through one machine belonging to the local network, it is rejected while it is allowed for any other machine. The third rule controls the HTTP requests from the public hosts. Again, connections are allowed for any host but for the local ones.

**Listing 1.1.** Firewall 1 netfilter configuration

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

iptables -N Out_SMTP
iptables -A FORWARD -s 111.222.1.17 -d 0.0.0.0/0 -p tcp --dport 25 -j Out_SMTP
iptables -A Out_SMTP -d 111.222.0.0/16 -j RETURN
iptables -A Out_SMTP -j ACCEPT

iptables -N In_SMPT
iptables -A FORWARD -s 0.0.0.0/0 -d 111.222.1.17 -p tcp --dport 25 -j In_SMTP
iptables -A In_SMTP -s 111.222.0.0/16 -j RETURN
iptables -A In_SMTP -j ACCEPT

iptables -N NetWeb_HTTP
iptables -A FORWARD -s 0.0.0.0/0 -d 111.222.1.17 -p tcp --dport 80 -j NetWeb_HTTP
iptables -A NetWeb_HTTP -s 111.222.0.0/16 -j RETURN
iptables -A NetWeb_HTTP -j ACCEPT
```

Firewall number 2 controls the traffic from the private hosts to the services provided in the DMZ. Listing 1.2 shows the rules that control the access to the SMTP and HTTP services. It is written in the Cisco PIX language that does not provide support to a feature like the iptables *custom chains*.

Rules one to six, control the SMTP requests to the server. They are all allowed for the hosts in the private zone discarding only the administrator host, identified by the IP address 111.222.2.54, and for a free-access host, identified by IP address 111.222.2.53. Rules seven to twelve do the same for the HTTP requests. Again, HTTP requests are allowed for all the hosts in the private zone discarding only the administrator host and the free-access host.

**Listing 1.2.** Firewall 2 Cisco PIX configuration

```
access-list eth1_acl_in  remark Fw2Policy 0 (global)
access-list eth1_acl_in deny tcp host 111.222.2.54 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 1 (global)
access-list eth1_acl_in deny tcp host 111.222.2.53 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 2 (global)
access-list eth1_acl_in permit tcp 111.222.2.0 255.255.255.0 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 4 (global)
access-list eth1_acl_in deny tcp host 111.222.2.54 111.222.1.17 eq 80

access-list eth1_acl_in remark Fw2Policy 5 (global)
access-list eth1_acl_in deny tcp host 111.222.2.53 111.222.1.17 eq 80

access-list eth1_acl_in remark Fw2Policy 3 (global)
access-list eth1_acl_in permit tcp 111.222.2.0 255.255.255.0 111.222.1.17 eq 80

access-group eth1_acl_in in interface eth1
```

### 2.1 Example Evaluation

Faced with this example, a security expert willing to understand the enforced access control rules will have to directly review the configuration files of the firewalls in the system (disregarding the low-level and often incomplete management tools provided by the firewall vendors, obviously only valid for the firewalls of that vendor), which in this case, involves two different rule languages. Not even the topology picture of the network, provided here with the purpose of easing the discussion, can be taken for granted but instead needs to be derived from the configuration files themselves.

Therefore, we can see that the task of extracting the global access control policy enforced by the set of rules in these two firewalls (that are just minimal excerpts of what a full configuration policy would be) requires expert knowledge about netfilter iptables and Cisco PIX. Its syntax along with its execution semantics would have to be mastered to properly interpret the meaning of the configuration files. Moreover, the information from the two configuration files and the default policies would have to be combined as they collaborate to enforce the global policy and can not be regarded in isolation.

In corporate networks potentially composed by up to a thousand firewalls, composed by hundreds of rules and potentially from different vendors using different configuration languages and execution semantics, the task of manually extracting the enforced access control policy would become very complex and expensive, seriously hampering the analysis and evolution tasks the dynamic environment of corporations impose. This is the challenge our approach aims to tackle as described in the next sections.

## 3 Approach

This section details our MDE approach to generate a high-level platform-independent model providing a global view of all access-control rules in a set of firewall configurations files.

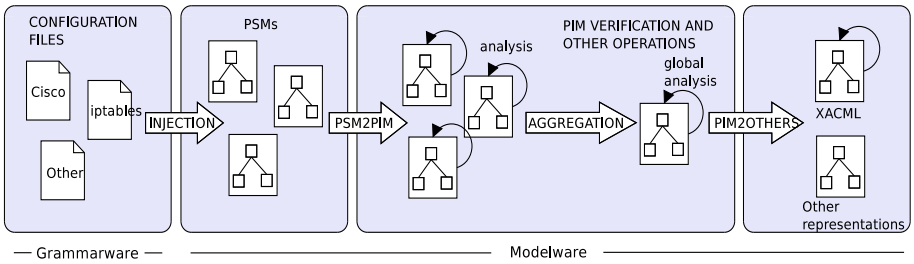


**Fig. 2.** Extraction approach

Our model-driven reverse engineering approach, that extends the preliminary one in [14], is summarized in Figure 2. It starts by injecting the information

contained in the firewall configuration files into platform-specific models (PSMs). Afterwards, each PSM is translated into a different network access-control PIM and an structural analysis to detect misconfigurations is performed. These PIMs are then aggregated into a global model, representing the access-control policy of the whole network. Operations are also performed over this global model to classify the information in "locally" or globally relevant.

### 3.1 Injection

The first step of our approach constitutes a mere translation between technical spaces where the textual information in the configuration files is expressed in terms of models. A PSM and a parser recognizing the grammar of each concrete firewall rule-filtering language present in the network system is required. In Listing 1.3 we excerpt a grammar for *CISCO PIX* whereas in Section 5, we show how we use it to obtain the corresponding parser and PSM. Due to space limitations, we do not show here the grammar for the *linux Iptables* filtering language (it is available on the web of the project [2]). The integration of any other language will follow the same strategy.

**Listing 1.3.** Cisco grammar excerpt

```
Model:
    rules += Rule*;
Rule:
    AccessGroup | AccessList;
AccessGroup:
    'access-group' id=ID 'in' 'interface' interface=Interface;
Interface:
    id=ID;
AccessList:
    ('no')? 'access-list' id=ID decision=('deny' | 'permit') protocol=Protocol
    protocolObjectGroup=ProtocolObjectGroup
    serviceObjectGroup=ServiceObjectGroup
    networkObjectGroup=NetworkObjectGroup;
ProtocolObjectGroup:
    (pogId=ID)? sourceAddress=IPExpr sourceMask=MaskExpr;
ServiceObjectGroup:
    targetAddress=IPExpr targetMask=IPExpr;
NetworkObjectGroup:
    operator=Operator port=INT;
Operator:
    name=('eq' | 'lt' | 'gt');
Protocol:
    name= ('tcp'| 'udp' | 'ip');
IPExpr:
    INT '.' INT '.' INT '.' INT;
```

Note that this step is performed without losing any information and that the obtained models remain at the same abstraction level as the configuration files.

### 3.2 Platform-specific to Platform-independent model

The second step of our approach implies transforming the PSMs obtained in the previous step to PIMs so that we get rid off the concrete details of the firewall technology, language and even writing style. Central to this step is the definition of a Network access-control metamodel able to represent the information contained in the PSMs. In the following we present and justify our proposal for such a metamodel.

Generally, firewall access-control policies can be seen as a set of individual security rules of type $R_i : \{conditions\} \rightarrow \{decision\}$, where the subindex $i$

specifies the ordering of the rule within the configuration file, *decision* can be accept or deny and *conditions* is a set of rule matching attributes like the source and destination addresses, the used protocol and the source and destination port.

Such a policy representation presents several disadvantages. First of all, the information is highly redundant and disperse, so that the details relevant to a given host or zone may appear, unassociated, in different places of the configuration file (potentially, containing up to several hundreds of rules). Metamodeling and model-driven technologies contain a big potential to reduce this issues, however, a proper representation must be chosen in order to maximize its benefits.

Second, this representation is not suited for representing the firewall policy in a natural and efficient way. Although firewall policies could be written by only using positive or negative logic (what leads however to over-complicated and not natural rule sets, impacting legibility and maintainability) a firewall access-control policy is better explained by expressing just rules in one sense (either negative or positive) and then exceptions (see [10] for a detailed study of the use of exceptions in access control policies) to the application of these rules. This way, in a close policy environment (where everything not explicitly accepted is forbidden) it is very common to define a security policy that accepts the traffic from a given zone and then denies it only for some elements of the zone. Native support for the representation of exceptions simplifies the representation and management of network policies while decreasing the risk of misconfiguration. The *custom chains mechanism*, recently provided by the linux iptables filter language, evidences the need for such a native support.

**3.2.1. Network Access-control Metamodel.** The platform independent network access-control metamodel we propose here (see Figure 3) provides support for the representation of rules and exceptions. Moreover, our reverse engineering approach is designed to recover an exception-oriented representation of network security policies from configuration files disregarding if they use a good representation of exceptions like in the iptables example in Section 2, or not, like in the Cisco PIX example in the same section.

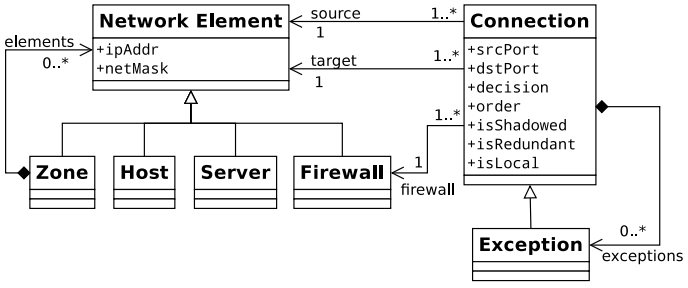


**Fig. 3.** Filter PIM excerpt

Our metamodel proposal contains the following elements (note that, for simplicity, some attributes and references are not represented in the image):

- *Network Element.* Represents any subject (source of the access request) or object (target of the access request) within a network system. It is characterised by its ip address and its network mask.
- *Zone, Host, Server and Firewall.* Several different types of *Network Element* may exist in a network environment. For the purpose of this paper, the relevant ones are: *Host*, *Zone* which in turn, contains other Network Elements, *Server* and *Firewall*. However, the list of elements can be extended to manage different scenarios, like the presence of routers, intrusion detections systems (IDSs), etc.
- *Connection.* Represents a connection between Network Elements. Apart from its source and target Network Elements, it is characterized by the following attributes: source and destination port, identifying the requested service; decision, stating if the connections is accepted or denied (our metamodel can represent open, close and mixed policies); order, reflecting the rule ordering in the corresponding configuration file; firewall, that identifies the firewall from where the connections were extracted; isLocal that tells is the connection is only locally relevant, isShadowed that identifies the connection as not reachable and finally, isRedundant, stating that the connection can be removed without affecting the policy.
- *Exception.* A connection may contain several exceptions to its application. These exceptions are connections with opposite decisions matching a subset of the elements matched by the containing connection.

**3.2.2. PSM-to-PIM transformation.** Our PIM metamodel provides the means for representing network access-control policies in a concise and organised way. However, a proper processing of the information coming from the configuration files is required in order to fully exploit its capacities (a policy could be represented by using only *Connections* without using the *Exception* element). Therefore, the process of populating the PIM model from a PSM model is composed by two sub-steps.

The first sub-step fills our PIM with the information as it is normally found in configuration files, i.e., in the form of a set of rules representing exceptions with mixed positive and negative logic. However, this representation can lead to policy anomalies and ambiguities. Concretely, as defined in [8], a firewall rule set may present the following anomalies:

*Rule shadowing:* a rule $R$ is shadowed when it never applies because another rule with higher priority matches all the packets it may match.

*Rule redundancy:* a rule $R$ is redundant when it is not shadowed and removing it from the rule set does not change the security policy.

*Rule irrelevance:* a rule $R$ is irrelevant when it is meant to match packets that does not pass by a given firewall.

Thus, the second sub-step, refines the initial PIM model and improve its internal organization to deal with the aforementioned problems. More specifically, this step applies the following algorithm on the PIM model (we describe the process for closed policies with exceptions, however, a version adapted to open policies would be straightforward):

1. Collect all the *Connection* elements $C$ whose decision is *Accept*.
2. For each retrieved *Connection* $C_i$, get *Connections* $C_j$ with the following constraints:
   (a) $C_j$ decision is *Deny*
   (b) $C_j$ conditions match a subset of the set matched by the conditions of $C_i$.
   (c) $C_j$ ordering number is lower than the $C_i$ ordering number (if not, mark $C_j$ as shadowed).

   Then, for each retrieved $C_j$ create an *Exception* element and aggregate it to the $C_i$. Remove the $C_j$ *Connection*.
3. For each remaining *Connection* element $C_j$ whose decision is *Deny* and isShadowed equals *false*:
   – mark $C_j$ as isRedundant

The algorithm we have presented is a modification of the one presented in [9], e.g. to drop the requirement of using as input policy one free of shadowing and redundancy. On the contrary, it is meant to work on real configurations and helps to discover these anomalies: *shadowed* deny rules and *redundant* deny rules. The security expert can retrieve them easily from the PIM as any left *Connection* in the PIM with decision *Deny* is an anomaly and as such is marked as *isShadowed* or as *isRedundant*.

This algorithm can be complemented by a direct application of additional algorithms described in [8] to uncover other less important anomalies. Note that the correction of these anomalies will often require the segmentation and rewriting of the rules, therefore we consider the correction as an optional step to be manually triggered by the security expert after analysing the detected anomalies.

### 3.3 Aggregation of individual PIMs

At the end of the previous step we get a set of PIM's (one per firewall in the network). Clearly, an individual firewall gives only a partial vision of the security policy enforced in the whole network. In our example, analyzing one firewall will yield that the public host can access the SMTP server, however, this server can be also accessed by the private network with some exceptions. Thus, in order to obtain a complete representation of the network policy the individual PIM models have to be combined into one global network access-control model. Note that as we keep information regarding which firewall contains a given *Connection* element and the ordering with respect to the original configuration file, this step

would be reversible, so that the individual policies may be reproduced from the global model.

We obtain the global model by performing a model union operation between the individual models, so that no *Network Element* or *Connection* is duplicated. Then, as an extra step, a refining transformation is performed to assign the proper type to the *Network Elements*. This step is performed by analysing the *ip* addresses and the incoming and outgoing connections. This way, we are able to establish if a network element is a zone or being an individual network element behaves as a host or a server (a unique firewall element is created upon the initialization of each PIM model in order to represent the firewall the rules come from). Once we have obtained the global model, some operations become available.

First of all, local *Exceptions* and *Connections*, i.e., *Exceptions* and *Connections* that only make sense in the context of a concrete firewall, can be identified (so that they can be filtered out when representing the global policy.). Local exceptions are usually added due to the mechanisms used to enforce the global policy. As en example, in the Listing 1.1 the elements in the network zone 222.111.0.0 are not allowed to send or receive smtp messages. However, elements in 222.111.2.0 are allowed to send them regarding the configuration Listing 1.2. This contradiction is due to the enforcing architecture that imposes the traffic to pass through a certain firewall (in this case, hosts in the local network are meant to access the DMZ through the second firewall). The algorithm to detect local *Exceptions* and *Connections* works as follows:

1. Collect all the *Exceptions* $E$ in the aggregated model.
2. For each *Exception* $E_i$, $L$ is a set of *Connections* $C$ with the following constraints:
   (a) $C_i$ is retrieved from a firewall different that the one containing $E_i$
   (b) $C_i$ conditions , are subset (or equal) of $E_i$ conditions.
   If the obtained set of *Connections* $L$ is not empty:
   - Mark $E_i$ as local.
   - For each $C_i$ in L, mark $C_i$ as local if it has not been already marked.

This will be also useful when extracting a representation of the network topology covered by the firewalls (see next section).

## 4 Application Scenarios

Once all the access-control information is aggregated in our final PIM, we are able to use the model in several interesting security application scenarios.

**Metrics and advanced queries.** First of all, having the access-control information of a network represented as a model, enables the reutilization of a plethora of well-known, off-the-shelf MDE tools. Editor generators, model transformation engines, etc. become automatically available. An immediate application would

be the use of the well-known OCL query language to calculate interesting metrics on the model and perform some advanced queries on the rules represented in it. In the following example, we query our model (in the example, the context of *self* is the root of our PIM) for the existence of any connection allowing the administrator host (111.222.2.54) to connect to the server (111.222.1.17):

```
Evaluating:
self.connections->exists(e | e.source.ipAddr='111.222.2.54' and e.target.ipAddr
    ='111.222.1.17')
Results:
false
```

**Forward engineering.** Our PIM model extracts and abstracts information from working networks. Nevertheless, the PIM is still rich enough to be able to be used as starting point for the regeneration of the configuration files if necessary (e.g. after modifications on the PIM to update the security policy of the network according to the new requirements). In that sense, some existing forward engineering efforts[18,4] that produce firewall configurations from high level representations can be reused.

**Visualization of the topology.** Our PIM can also be used to derive the topology of the network, i.e., the arrangement of components and how the information flow through them. For this purpose, a model-driven visualization tool like Portolan[4] can be used. A transformation from our aggregated PIM towards the Portolan Cartography model (Portolan is able to graphically represent any model corresponding to its Cartography metamodel) has been written. This transformation analyzes the global PIM to first, extract the *Firewall* elements and represent them as nodes. Then, represent the other *Network Elements* also as nodes and the local containment of *Zones*. Finally, it extracts the *Connections* and build the links between each *Connection* source *Network Element* to the corresponding *Firewall* element and from the *Firewall* element to the target *Network Element*.

In Figure 4 we show the visualization the tool provided. In the figure, servers (element 111.222.1.17), firewalls, zones and contained elements are easily identifiably as well as the enabled connections between them. If we compare this figure with the figure 1 presented in section 2, we can see that the topology is accurately represented.

**Network PIM to XACML.** Our proposed network access-control PIM is a specific representation specially tailored to the network domain. We consider that a translation from our PIM towards a more generic access-control representation will complement our approach by enabling reusing tools and results that work on the general access-control model have produced.

XACML [13] is an OASIS standard for the specification of access-control policies in XML and is ABAC[23] and RBAC[20] (two of the most successful access-control models) capable. Its flexibility to represent multiple policies for the same system and the fact of counting with a reference implementation, along

[4] http://code.google.com/a/eclipselabs.org/p/portolan/

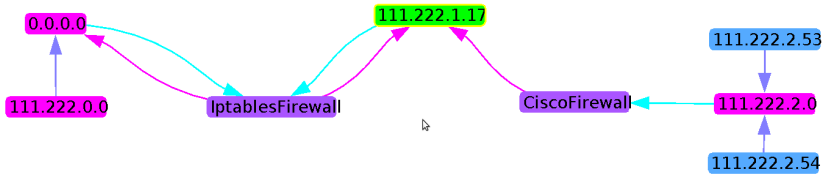


**Fig. 4.** Extracted network topology

with the increasing adoption by industry and academy, makes XACML a good choice for a generic access-control representation. Indeed, some works in the model-driven community already chose XACML as a target language as in [3] and [16].

In the following, we briefly introduce the XACML policy language and the mapping from our PIM.

XACML policies are composed by three main elements *PolicySet*, *Policy* and *Rule*. A *PolicySet* can contain other *PolicySets* or a single *Policy* that is a container of *Rules* (Policy and PolicySet also specify a rule-combining algorithm, in order to solve conflicts between their contained elements). These three elements can specify a *Target* that establishes its applicability, i.e., to which combination of *Subject*, *Resource* and *Action* the *PolicySet*, *Policy* and *Rule* applies. *Subject*, *Resource* and *Action* identifies subjects accessing given resources to perform actions. These elements can hold *Attribute* elements, that represent additional characteristics (e.g., the role of the subject). Optionally, a *Rule* element can hold a *Condition* that represents a boolean condition over a subject resource or action. Upon an access request, these elements are used to get an answer of the type: permit, deny or not applicable.

The translation from our network access-control metamodel to XACML follows the mapping summarized in Table 1. In Listing 1.4 we excerpt the XACML representation of a PIM *Connection*.

| XACML | PIM Metamodel |
|---|---|
| PolicySet | A PolicySet containing a Policy is created for each firewall in the PIM |
| Policy | All the Connections and Exceptions belonging to a given firewall |
| Rule | A single connection or Exception |
| Subject | Source NetworkElement address and source port of a given Connection or Exception |
| Resource | Target NetworkElement address and target port a given Connection or Exception |
| Action | Not mapped. The action is always the ability of sending a message. |
| Condition | Protocol field |

**Table 1.** PIM to XACML Mappings

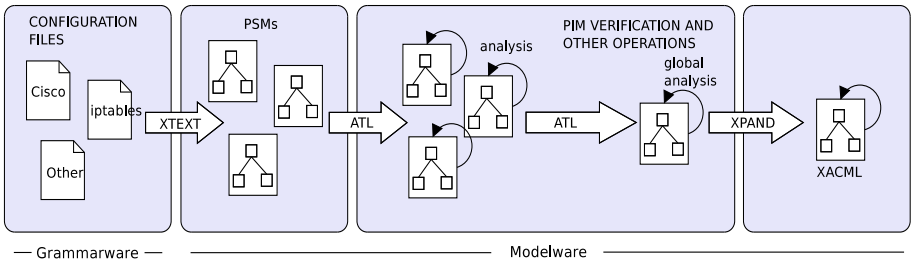


**Fig. 5.** Extraction approach implementation

With this translation, the utilisation of a wide range of tools and research results based in the standard become enabled. Between the more interesting ones, we can reuse the several formalizations of the language as provided by [7,11]. Reusing these formal approaches, operations like automatic policy comparison and change impact analysis can be performed.

**Listing 1.4.** Firewall rule in XACML

```
<Rule Effect="Deny" RuleId="1">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="...function:ipAddress-regexp-match">
          <AttributeValue
           DataType="...XMLSchema#string">(111)\.(222)\.(2)\.([0-9][0-9]?[0-9]?)
          </AttributeValue>
          <SubjectAttributeDesignator
           SubjectCategory="...subject-category:access-subject"
           AttributeId="...:subject:subject-id"
           DataType="....data-type:ipAddress"/>
        ...
    <Resources>
      <Resource>
        <ResourceMatch MatchId="...function:ipAddress-regexp-match">
          <AttributeValue
           DataType="...XMLSchema#string">(111)\.(222)\.(1)\.(17)</AttributeValue>
          <ResourceAttributeDesignator AttributeId="...resource:resource-id">
           DataType="...data-type:ipAddress"/>
        ...
  </Target>
  <Condition>
    <SubjectAttributeDesignator AttributeId="protocol"
     DataType="...XMLSchema#string" />
    ...
</Rule>
```

## 5 Implementation

In order to validate the feasibility of our approach, a prototype tool[2], able to work with two popular firewall filtering languages *Linux Iptables* and *Cisco PIX*, has been developed under the Eclipse[5] environment. Figure 5 summarizes the steps and technological choices we made for the prototype development.

The tool implements the first step of our approach (the injection process) with Xtext [6], an Eclipse plugin for building domain specific languages. As an input

[5] http://www.eclipse.org/
[6] http://www.eclipse.org/Xtext/

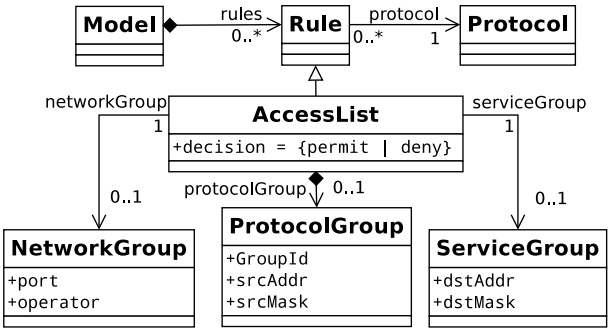


**Fig. 6.** Cisco Metamodel excerpt

to this tool we have written simple yet usable grammars for the two languages supported by our tool. By providing these two grammars the Xtext tool creates for us the corresponding metamodels depicted in Figures 6 and 7 along with the parser and the injector needed to get models out of the configurations files.

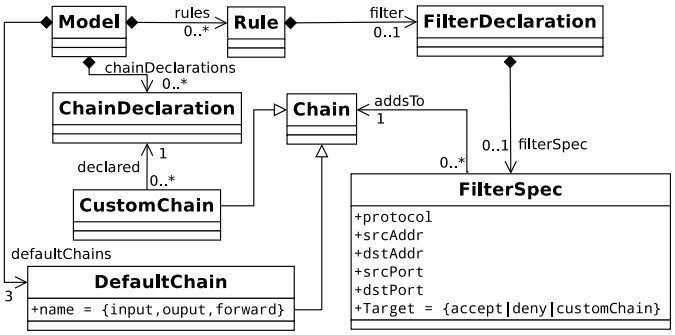


**Fig. 7.** Iptables Metamodel excerpt

The transformations from the PSMs to the PIM along with the detection of anomalies have been written using the model transformation language ATL[12], both in its normal and in-place (for model refining[21]) modes. Same for the PIM's aggregation process. The visualization of the topology relies on Portolan and the translation from our PIM to XACML policies has been developed using Xpand[7], a model-to-text tool.

[7] http://wiki.eclipse.org/Xpand

## 6 Related Work

Several other works tackle the problem of extracting access control policies from network configurations but they either are limited to analyzing one single firewall component or focus on a specific analysis task and thus they do not generate a usable representation of the firewall/s under analysis. Moreover, these latter works require as an additional input the network topology, instead we are able to calculate it as part of the process.

More specifically, [22] proposes a technique that aims to infer the higher-level security policy rules from the rules on firewalls by extracting classes (types) of services, hosts and protocols. However, it takes only one firewall into account for the process. In [15] and [17] a method and tool to discover and test a network security policy is proposed. The configuration files along with the description of the network topology are used to build an internal representation of the policy that can be verified by the user through queries in ad-hoc languages. Unfortunately, no explicit model of the recovered security policy is provided and thus, the extracted policy can not be globally inspected without learning complex query languages. [5] proposes a bi-directional method to enforce and reverse engineer firewall configurations. It promotes the use of an intermediate policy representation but does not provide a model for such representation nor specific processes to perform the enforcement and the discovery tasks.

Some other works provide a metamodel for representing firewall configurations like [18], [24] and [6]. Nevertheless, a reverse engineering process to populate those models from existing configuration files is not provided, and in our opinion, the abstraction of the models level is still too close to the implementation details, therefore limiting their usability.

## 7 Conclusions and Future Work

We have presented a model-driven reverse engineering approach to extract network access-control policies from the network information included in the network's firewall configuration files. As a result of the process, a platform-independent access control model is created. Apart from facilitating the comprehension and analysis of the network policies to security experts, this model can also be the basis for further applications like the visualization of the (implicit) network topology or the generation of an equivalent XACML-like model ready to be processed by specialized security reasoning tools.

As a future work we plan to extend our approach to take into account other network elements that may take part in the enforcement of a security policy like routers, VPN tunnels and intrusion detection systems. We consider also that giving precise semantics to the proposed metamodel concepts constitutes a necessary next step. Moreover, given that the own XACML defines extensibility mechanisms for this standard, we believe it would be useful to work on a network-specific extension that facilitates the representation and analysis of this kind of firewall access-control policies. Finally, we plan to apply our approach on real network configurations to test the scalability of the approach.

## References


1. *Building secure software: how to avoid security problems the right way.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
2. Firewall Reverse Engineering project web site. `http://www.emn.fr/z-info/atlanmod/index.php/Firewall_Reverse_Engineering`, 2013.
3. M. Alam, M. Hafner, and R. Breu. Constraint based role based access control in the sectet-framework: A model-driven approach. *J. Comput. Secur.*, 16(2):223–260, Apr. 2008.
4. Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, Nov. 2004.
5. M. Bishop and S. Peisert. Your security policy is what?? Technical report, 2006.
6. A. D. Brucker, L. Brügger, P. Kearney, and B. Wolff. Verified firewall policy transformations for test-case generation. In *Third International Conference on Software Testing, Verification, and Validation (ICST)*, pages 345–354. IEEE Computer Society, Los Alamitos, CA, USA, 2010.
7. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.
8. J. Garcia-Alfaro, N. Boulahia-Cuppens, and F. Cuppens. Complete analysis of configuration rules to guarantee reliable network security policies. *Int. J. Inf. Secur.*, 7(2):103–122, Mar. 2008.
9. J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Aggregating and deploying network access control policies. volume 0, pages 532–542, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
10. J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Management of exceptions on access control policies. In *SEC*, volume 232 of *IFIP*, pages 97–108. Springer, 2007.
11. G. Hughes and T. Bultan. Automated verification of access control policies using a sat solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–520, Oct. 2008.
12. F. Jouault and I. Kurtev. Transforming models with atl. In *MoDELS Satellite Events*, pages 128–138, 2005.
13. H. Lockhart, B. Parducci, and A. Anderson. OASIS XACML TC, 2013.
14. S. Martínez, J. Cabot, J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. A model-driven approach for the extraction of network access-control policies. In *Proceedings of the Workshop on Model-Driven Security*, MDsec '12, pages 5:1–5:6. ACM, 2012.
15. A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 177–, Washington, DC, USA, 2000. IEEE Computer Society.
16. T. Mouelhi, F. Fleurey, B. Baudry, and Y. Traon. A model-based framework for security policy specification, deployment and testing. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 537–552, Berlin, Heidelberg, 2008. Springer-Verlag.
17. T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. The margrave tool for firewall analysis. In *Proceedings of the 24th international conference on Large installation system administration*, LISA'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

18. S. Pozo, R. Gasca, A. Reina-Quintero, and A. Varela-Vaca. Confiddent: A model-driven consistent and non-redundant layer-3 firewall acl design, development and maintenance framework. *Journal of Systems and Software*, 85(2):425 – 457, 2012.
19. R. Russell. Linux 2.4 packet filtering howto. http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html, 2002.
20. R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM.
21. M. Tisi, S. Martínez, F. Jouault, and J. Cabot. Refining Models with Rule-based Model Transformations. Rapport de recherche RR-7582, INRIA, 2011.
22. A. Tongaonkar, N. Inamdar, and R. Sekar. Inferring higher level policies from firewall rules. In *Proceedings of the 21st conference on Large Installation System Administration Conference*, LISA'07, pages 2:1–2:10, Berkeley, CA, USA, 2007. USENIX Association.
23. E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 561–569, Washington, DC, USA, 2005. IEEE Computer Society.
24. V. Zaliva. Platform-independent firewall policy representation. *CoRR*, abs/0805.1886, 2008.