Survey on the Use of Formal Languages/Models for the Specification, Verification, and Enforcement of Network Access-lists

Adel El-Atawy
School of Computer Science,
Telecommunication, and Information Systems
DePaul University
Chicago, Illinois 60604
Email: aelatawy@cs.depaul.edu

April 12, 2006

Abstract

Complexity of access-lists and the diversity of their specifications are continuously increasing. Stating the high level requirements as well as verification of the implemented policies became an impossible task if human intervention is required. Also, proving the soundness of these inter-related and confusing policies is very hard without an appropriate framework. Therefore, a formal and canonical specification for security access-lists is highly needed for us to be able to specify requirements, verify correctness and enforce the policy.

In this paper, we present some of the work available in the literature that discusses these problems and propose solutions for having an automated network security policy management.

1 Introduction

Complexity of access-lists and the diversity of their specifications are continuously increasing. Size of access lists (or security policy) can be in order of thousands per security-enforcing device in the system (*e.g.*, firewall, IDS, screening router, ...). Moreover, the interaction between these devices and the complexity of the underlying network can cause any modification to one of the policies to be propagated to all other devices. For example, if a firewall at the edge of the network rejected an incoming flow, then there will be no need to re-consider it in following policies. Also, if a traffic is to be rejected by firewalls in the depth of the network, then allowing it in the first place will only leave the internal network's capacity to be eaten up with useless traffic.

Although these examples seem like serious problems, the more dangerous cases take place when malicious traffic is allowed by mistake into the corporate servers due to lack of coordination between different security devices. For example, if traffic was to be rejected by advanced devices (*i.e.*, devices near the network edge) and the devices in depth has already optimized their policies depending on this fact, then the first device filtering mechanism was disabled as a part of a periodical maintenance or a faulty upgrade (which is very common); the network is endangered by this unaccounted for flows.

In business-based security policies, more complex conditions and restrictions can take place. Examples of such rule; having an employee's access restricted in only a given time of year, or after he performs some specific operation. Or having highly secured documents accessed by only a limited number of viewers at a time, or if permitted by another person with higher access-rights. Or conflict of interest between roles a user can play at the same time, although he can play each role separately in different business sessions. Examples can be quite sophisticated and complex for analysis and enforcement purposes.

Scenarios like these ones are very common and can be very hard to expect or anticipate their occurrences. Thus, an automated robust infrastructure is needed to communicate this information between these security enforcement devices/entities as well as between administrators. Also, such framework will be responsible for the more problematic aspect of network administration; translating high level requirements into actual device policies. With thousands of rules distributed over tens of devices, it becomes an impossible task for a human administrator to tackle. Besides, the standards in which these policies are written are ever changing and each vendor has his own version of policy grammar, and special cases of access and traffic handling.

Therefore, a formal and canonical specification for representing security access-lists is highly needed to be able to specify requirements, verify correctness and enforce the policy. Trials to formalize security requirements and policies started a long time ago. In 1981, [13] showed different models and approaches to specify, model and enforce security policies in basic electronic systems for military purposes. In 1993, [1] proposed a calculus for access control in distributed systems. In 1994, [18] tried to show a general formal definition and architecture for network and systems policies. However, the problems are continuously emerging as systems get more complex, and computational features keep on increasing.

In this paper, we present some of the work available in the literature that discusses these problems and propose solutions for having an automated network security policy management. Modeling security policies was the focus of many researchers with diverse backgrounds, approaches and targets. Each section is designated to a specific approach or an application in which formal model and formal languages are used in network security policies.

2 Background

In this section, some of the basics of network security as well as established models of Security Policy will be presented.

2.1 Network Security and Network Access Lists

Limiting the traffic entering an enterprize or corporate network is a must to maintain a streamlined operation. The number of network-based attacks and intrusion attempts is continuously increasing, and a way to enforce some form of a security policy is mandatory to stop these attacks. Moreover, insider malicious activity is also a threat to the corporate security. Valuable information might be sent from inside-out thus policies should include restrictions on the allowed types of data transmission from entities within. Even in personal and home networks, sensitive personal data might be endangered by outsiders. Also, worms and similar threats has to be stopped from being propagated from the SOHO (small office/home office) to the outside.

Specifying the network security policies (P) is usually specified as a sequence of filtering rules ($R_1, R_2, \ldots, R_n$) feed into an independent security device or host-based software, called a firewall. Each rule consists of two parts; a criteria and an action. Using the notation from [11];

$$R_i := C_i \rightarrow act_i$$

where act_i is one of the possible actions that can be applied on a single packet (*e.g.*, accept, discard, protect, log,...), and C_i is a set of values that must be found in their corresponding fields in the packet for the criteria to be fulfilled.

$$C_i = fv_i^1 \wedge fv_i^2 \wedge \ldots \wedge fv_i^k$$

given that we have k fields that can be used as filtering conditions, and fv_i^j means that in field j the value is the one mentioned in rule i.

Normally, policies work under single-trigger conditions, which means a packet is checked against each rule top-to-bottom until a successful match is found and the action of such a rule is applied. In some devices, as in IPSec, multi-trigger is used, where all actions of all satisfied rules will be applied to the packet.

2.2 Role Based Access Control Model

References: [5], [9], [12] and [17].

To specify access restrictions and permissions, RBAC (Role Based Access Control Model) was suggested in order to facilitate user access to resources. A user is granted (or prohibited) access to a specific resource based on the roles he can play. Within the roles, there exist what is called a role hierarchy which allows some kind of inheritance between the rule rights. [In other words, we can view roles as typing, and role hierarchy is a way to implement sub-typing over the entities defined in the system. However, objects can play different roles, and have multiple types types.]

Of course, some complications arises when there conflict of interest between two roles that are granted to the same subject or user, or when a user is granted more roles than what he is allowed to, or even when the user is granted a role that is beyond his qualifications. These are solved by having what is called Separation of Duty Constraints, cardinality, and prerequisites constraints (will be discussed in later sections).

A comparison between simple access lists, and RBAC structure can be found in [5]. Features of each are compared and shown to be equivalent in almost all cases. However, the version of the RBAC used in comparison is somehow restricted, so it can be stated that RBAC models are more powerful in modeling higher level policies than access lists.

In [12], a model for flat role-based access control (FRBAC) is given in the **Z** specification notation. The FRBAC is a special case of the RBAC, where there is no hierarchical structure in any of the entities (roles, users, objects . . . are all flat).

2.3 Organizational Based Access Control Model

Main References: [10], [7] and [8].

The above mentioned RBAC is missing some of the necessary features needed for fully modeling the AC requirements. Some of these limitations are; the lack of contextual permissions or prohibition rules, obligation or recommendation rules cannot be modeled, organization-specific rules have no means of being stated. Or-BAC was proposed to solve these limitations and others.

The overall entity relation possibilities can be represented in the following figure.

Within this model, there is a hierarchical for all the entities; organizations, roles, objects, ... Also, the concept of views is presented, where an object can be a member of multiple views, each view can be seen by a different organization. In other words, objects can play views as users can play roles. But the distinction between both comes from the fact that objects are not active entities, they cannot perform actions or activities. Activities are the actual effect that a user can exert, while actions are the conceptual versions of such activities. Each organization can consider an activity to be a specific action. For example, in a hospital system, the activity "consulting" can be considered an action of "reading" for the "archives" department, while the same activity can be considered a "writing" action for the "emergency room".

Another concept is the concept of "a context". A context is how the current circumstances are stated. The "Define" relation is between the organization, its subjects, actions, objects and context. In the same hospital example, context can be "emergency", "normal operation", or "urgency".

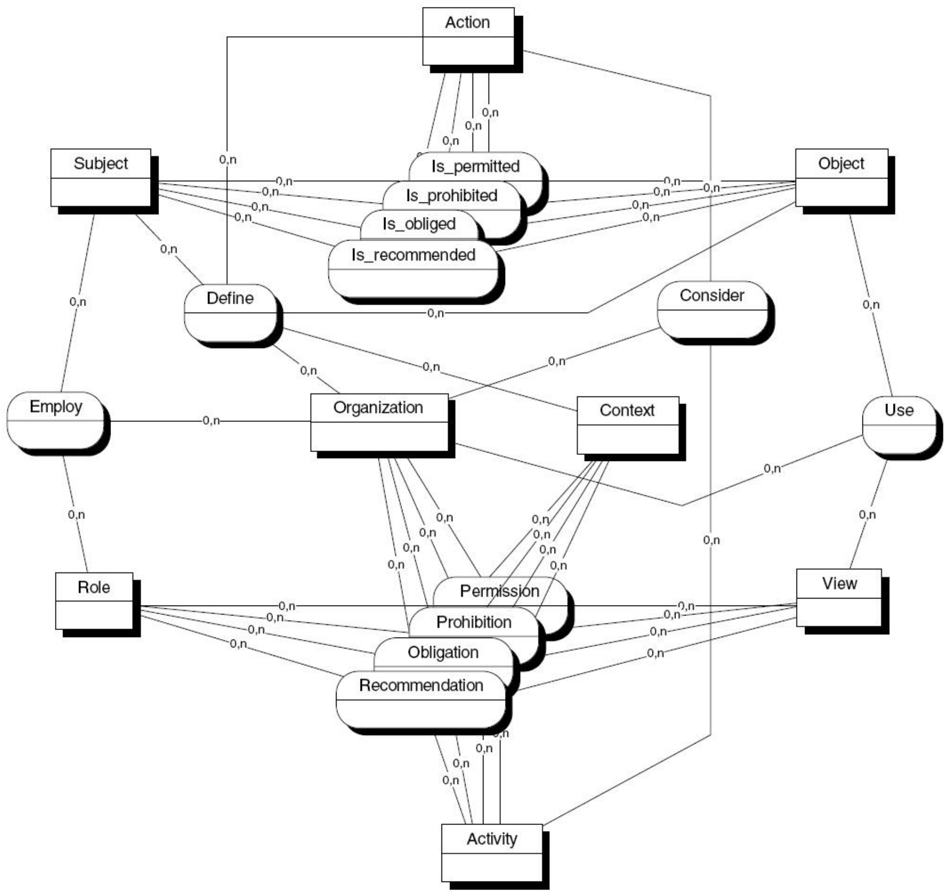


Figure 1: The ORBAC Model

More details about how to model contexts, and the conceptual meaning behind this modeling can be found in another paper [7] by some of the coauthors who originally proposed the OrBAC.

The system includes support for constraints as add-on predicates.

Different variants of the ORBAC were presented later on. For example, the AdOr-BAC (Administrative Organization Based Access Control) [8]. These additions were suggested to enable administrative management of the OrBAC, additions/deletetions of entities, defining relations, ... It can be viewed as a meta-policy for accessing the policy at run-time.

3 UML Version of Security Policy

Main References: [15], [14] and [16].

3.1 RBAC into UML/OCL

Among the first trials to model RBAC completely including its constraints into UML were by [2]. The constraints themselves - previously defined by other researchers - were modeled into the UML using its accompanying Object Constraint Language (OCL). The authors have previously defined another language (RCL2000: Role-based Constraint specification Language 2000) to specify formally the constraints required in a system. However, for these specifications to be used in actual system design, the migration to a model like UML/OCL was a must. The link between UML and OCL themselves were not fully exploited before this work. Examples of writing constraints using OCL are provided in the next section.

3.2 RBAC Constraints Visualization using UML

In [16], the (Unified Modeling Language) UML was used as an intermediate step to reach the final goal of a verified policy. Given the RBAC the proposed system maps this into an UML class template. The template can be then instantiated with the values and settings given in the specific system under consideration. One of the great advantages of the use of UML is that the application framework can be incorporated with the system security policy into a single entity.

UML has different families of diagrams; the authors used class diagram template and the object diagram template. The first for modeling the RBAC pattern, the latter for the system constraints. The second template is used to model the system constraints that are provided as (Object constraint language) OCL.

The *Class Diagram Template* was used to model the RBAC (that is considered a pattern) with its entities and relations. The diagram consists of class descriptors with its parameters and single-parameter constraints. To represent the RBAC fully, they needed *User, Role, Session,* and *Permission* classes for the different entities in the RBAC. For example, for the system users, there exist the Users template. Some operations are defined as *CreateSession, AssignedRoles, DeassignedRoles*, ... Other templates are the association templates like *UserAssignment* and *SessionRoles*. Activity or *Operation* templates are constrained in their operation by OCL templates that place pre- and post- conditions on their execution. An example of conditions can be written as:

```
context| User::|CreateSession():(|s:|Session)
  post: result=|s and
  |s.oclIsNew()=true and
  self.|Session → includes(|s)


context|Role::|GrantPermission():(|p:|Permission)
  pre:self.|Permission → excludes(|p)
  post:self.|Permission → includes(|p)
```

The *Object Diagram Template* is used to model the system constraints that are external to the RBAC definition. Normally, OCLs are based on first order logic, which makes them very hard to understand and analyze, especially for the ordinary user. While OCLs can be given as annotations to the RBAC when modeled as class diagram, writing them as Object Diagrams is more formal and precise.

In the paper, they show three types of constraints and specify formally how they can be converted to Object entities in the Object Diagram. These

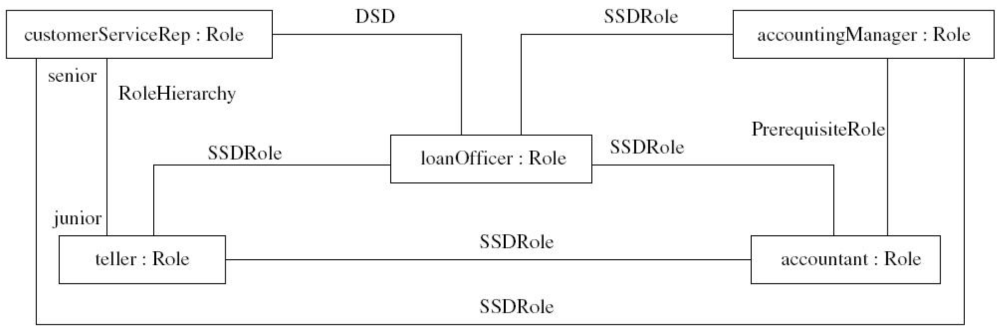


Figure 2: The simple model of the Banking Organization

types are the Separation of Duty Constraints, the Prerequisite Constraints, and the Cardinality Constraints. The first deals with conflict of interest between policy parts. It can be Static Separation of Duty (SSD), which ensures the validity of the current state (*e.g.*, after user is granted permissions for conflicting roles, ...). The other is Dynamic Separation of Duty (DSD), which deals with conflict of interest as well, but within the running session. To see the difference, these are two constraints for each type;

```
context |SSDRole inv:
   |r1.|User → excludesAll(|r2.|User)

context |DSDRole inv:
   |r1.|Session → excludesAll(|r2.|Session)
```

Each constraint limited the activation of conflicting entities. However, the SSD focused on roles given to users, while DSD focuses on enabling roles within a single session. For further details, the paper presents diagrams with detailed explanation for each type of constraints.

The Second type of constraints (Prerequisite Constraints) are obvious from their name. For example, a user cannot be head of technical department unless he had already acquired the role of administrator.

The third type of constraint; Cardinality constraints ensures that there is a limit on the number of roles a user can play. In general, they restrict the number of entities that can be related to each other.

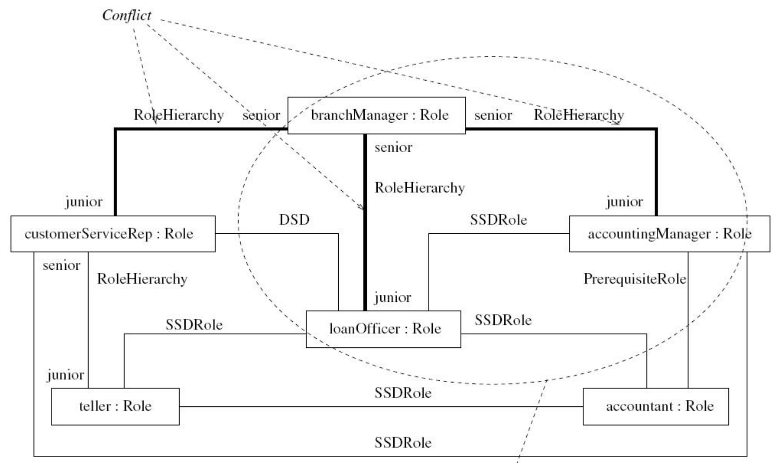


Figure 3: Violation of the SSD Constraint

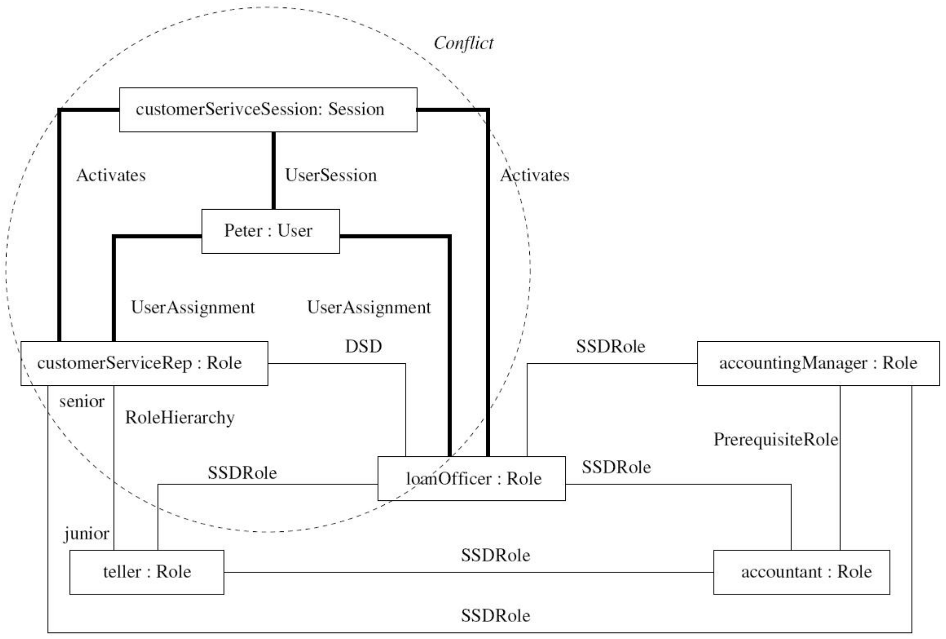


Figure 4: Violation of the DSD Constraint

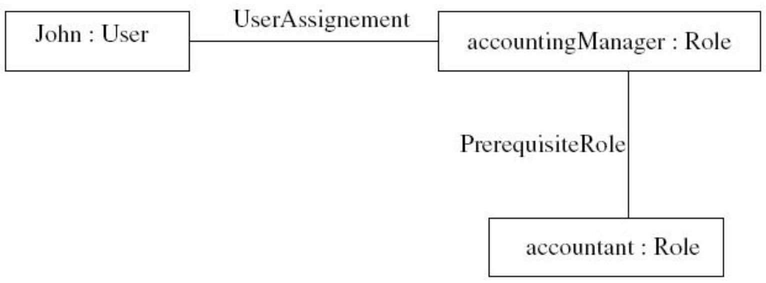


Figure 5: Violation of the Pre-requisite Constraint

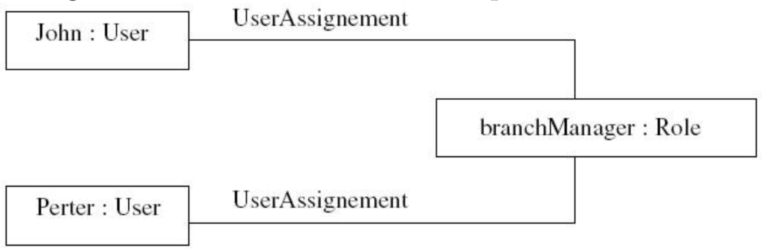


Figure 6: Violation of the Cardinality Constraint

The paper presented a case study on a banking application. It included a Teller, a Customer Service Rep, an accountant, a loan officer as users. Also, it states that some pairs are conflicting as (teller, loan officer), (loan officer, accountant). Other DSD, Prerequisite and cardinality constraints are also modeled.

Some operations were presented to the model, and conflicting patterns were identified visually quite easily. Of course, an automated system could be designed to check for these predefined patterns and pinpoint them efficiently.

3.3 Secure UML

Other work by [14] also used UML as a means of modeling security policies. In [14], a modified UML was proposed that incorporate the security requirements in the system design from the very beginning. They used OCL (Object Constraint Language) as the expression language for defining the constraints. This is not a new idea as it was the goal of the other work discussed in this section. However, they added authorization as a new constraint type to the previously mentioned constraint types and showed actually how to integrate both the system and policy UMLs together.

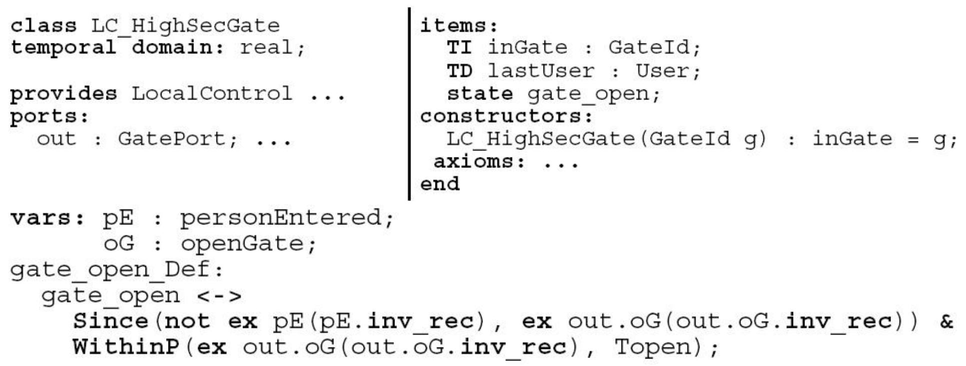
3.4 UML-compatible formal language

Another reference [15], is not focused directly to security policies. However, they represent a UML-compatible formal language that can be used for writing down descriptions for system architecture (that includes the security policy). The paper presents **ArchiTRIO**, a HOL temporal logic language which combines a subset of the UML notation with a precise formal semantics from previously known languages (by the same group).

The main advantage of their work is that it is possible to make a hybrid design; non-critical parts can be presented using the traditional UML notation, and critical aspects of the system can then be strictly and completely represented using their formal language. Both parts are then fully integrated into one single overall design. A user can start with the traditional UML design until the need arises for more clarity and precision, then details are embedded using a formal notation into the design.

The paper shows a running example of a building with three clearance levels, and users have access rights and there are physical gates that are controlled by the overall security policy. In order to model the section with high clearance with its complex logic (more than UML can support), the authors show how using their temporal-logic-based language they can specify the policy conditions. At first the system is designed simply using UML; showing how sections are connected physically, and how the less secure sections interact, open/close their gates, conditions on incoming personnel, ... Then a class written specifically for the extra constraints is embedded into the object of the high security section. In the example they specify the condition on which the gate opens; it must be opened within the last TOpen seconds, and it must be opened and not used yet by the user opened it, once he enters it closes.

Future work mentioned for this paper included studying the possibility of encoding the proposed language into HOL (Higher Order Logic) of one of the available theorem provers such as PVS. Also, they plan to translate there classes into automata-based formalization in order to exploit model checking techniques.



```
class LC_HighSecGate
temporal domain: real;

provides LocalControl ...
ports:
  out : GatePort; ...
```

```
items:
  TI inGate : GateId;
  TD lastUser : User;
  state gate_open;
constructors:
  LC_HighSecGate(GateId g) : inGate = g;
 axioms: ...
end
```

```
vars: pE : personEntered;
      oG : openGate;
gate_open_Def:
  gate_open <->
    Since(not ex pE(pE.inv_rec), ex out.oG(out.oG.inv_rec)) &
    WithinP(ex out.oG(out.oG.inv_rec), Topen);
```

Figure 7: ArchiTRIO: An example of a class and one of its axioms

4 XML Translation of Security Policies

Main References: [6] and [19].

4.1 Policy Specification/Generation using XML/XSL

In [6], the authors propose using XML to specify a version of the Or-BAC (actually a reduced version) that is specific for network policies. Then this XML version can later be translated to any specific firewall language using the appropriate XSL.

The use of Or-BAC to start extracting the policy from is due to the fact that Or-BAC is the superset of RBAC and, clearly, firewall filtering policies. This ensures that no generated policy will be violating the global view of the organization's policy.

Another point of interest in their work is that they tried to specify the policy ONLY as permissions (no prohibition, obligation, or recommendation rules). This will make the order among the rules in the policy irrelevant, which is a huge plus as most of firewall rule anomalies occur from having the wrong order between the rules. Also, it worth mentioning that they model the firewall as an organization in its own, of course a hierarchy or network devices is present to govern all of them together.

The translation from the XML to the final policy rules takes place in two steps. First, the XML is translated to an intermediate policy (via XSL), where the policy can be checked for correctness and being cycle free, ...

Secondly, another XSL translation generates the final text that can be feed to the firewall.

4.2 Access Control Systems in XACML/RW

In [19], a verifiable formal language is used to specify the policy. RW can be formally verifiable in the sense that we can always answer this question: "Can agent x do action a to object o within t time from event e?". The language RW (stands for read write) models the right of agents to perform the two basic operation read and write. The Access Control System S is represented as a tuple $< A, P, r, w >$, where A is a set of agents, P is the space of propositional variables, r and w are mappings from $PxA \rightarrow L(P)$. Agent a can read/write a variable p if and only if $r(p, a) = true$ or $w(p, a) = true$, respectively. The logical condition in each element of the space PxA specifies if a can read (or write) the variable p.

The paper presents an example of a scientific conference with conditions on whom can see the authors information, set the reviewers, change the submitted paper, ... So given the following rules, conditions will be created, followed by the appropriate logical conditions.

1. PC members and authors of papers are known to everybody. Authors of papers cannot be changed.
2. The PC chair appoints the PC members. A PC member can resign his membership.
3. The PC chair can assign a paper to a PC member for reviewing, except if he is one of its authors.
4. All PC members, except the author(s) of a paper can know who are the reviewers for this paper.
5. The reviewer of a paper can assign the paper to be sub-reviewed by an agent who is not an author of the paper and has not been assigned the same paper by another reviewer.
6. A reviewer of a paper p can resign, unless he has already appointed a sub-reviewer for the paper.

7. Sub-reviewers are known to all PC members who are not authors of the respective papers.

8. A sub-reviewer can resign, unless he has already submitted his review.

$$\mathtt{r}(\mathtt{author}(p,a),x) \rightleftharpoons \mathtt{true},\ \mathtt{r}(\mathtt{pcmember}(a),x) \rightleftharpoons \mathtt{true}$$

$$\mathtt{w}(\mathtt{pcmember}(a),x) \rightleftharpoons \mathtt{chair}(x) \vee \mathtt{pcmember}(a)$$

$$\mathtt{r}(\mathtt{reviewer}(p,a),x) \rightleftharpoons \mathtt{pcmember}(x) \wedge \neg\mathtt{author}(p,x)$$

$$\mathtt{w}(\mathtt{reviewer}(p,a),x) \rightleftharpoons \left(\begin{array}{l} \left(\begin{array}{l} (\mathtt{chair}(x) \\ \wedge\mathtt{pcmember}(a)) \\ \wedge \\ (\neg\mathtt{author}(p,a) \\ \wedge\neg\mathtt{reviewer}(p,a)) \end{array} \right) \\ \vee \left(\begin{array}{l} \mathtt{pcmember}(a) \\ \wedge x = a \\ \wedge\mathtt{reviewer}(p,a) \\ \wedge\neg(\exists\ b \in \mathtt{A}) \\ \mathtt{sub}(p,a,b) \end{array} \right) \end{array} \right)$$

$$\mathtt{r}(\mathtt{sub}(p,a,b),x) \rightleftharpoons \left(\begin{array}{l} (\mathtt{pcmember}(x) \\ \wedge\neg\mathtt{author}(p,x)) \\ \vee x = b \end{array} \right)$$

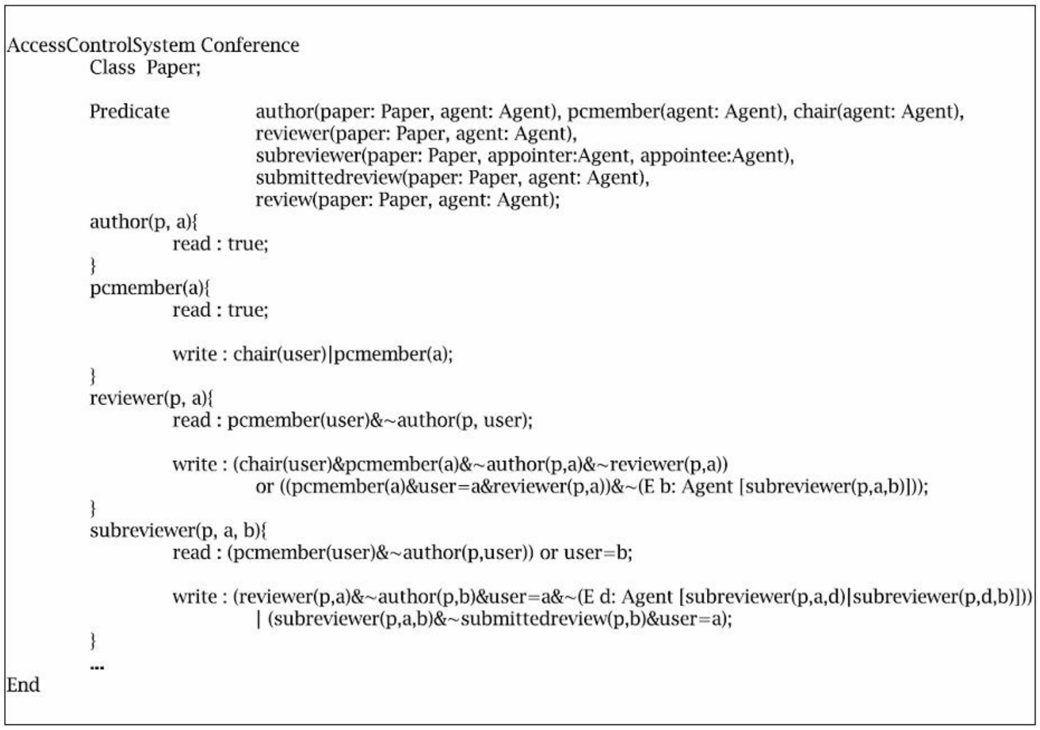
$$\mathtt{w}(\mathtt{sub}(p,a,b),x) \rightleftharpoons \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} (\mathtt{reviewer}(p,a) \\ \wedge\neg\mathtt{author}(p,b)) \\ \wedge x = a \end{array} \right) \\ \wedge\neg(\exists\ d \in \mathtt{A}) \\ \left(\begin{array}{l} \mathtt{sub}(p,a,d) \\ \vee\mathtt{sub}(p,d,b) \end{array} \right) \end{array} \right) \\ \vee \left(\begin{array}{l} (\mathtt{sub}(p,a,b) \\ \wedge x = b) \\ \wedge\neg\mathtt{submit}(p,b) \end{array} \right) \end{array} \right)$$

Figure 8: The permission mapping r and w for the given example

After obtaining the logical conditions needed, the program in RW can be written as in Fig 9. As an example, the *pcmember*(a) predicate is translated into XACML.

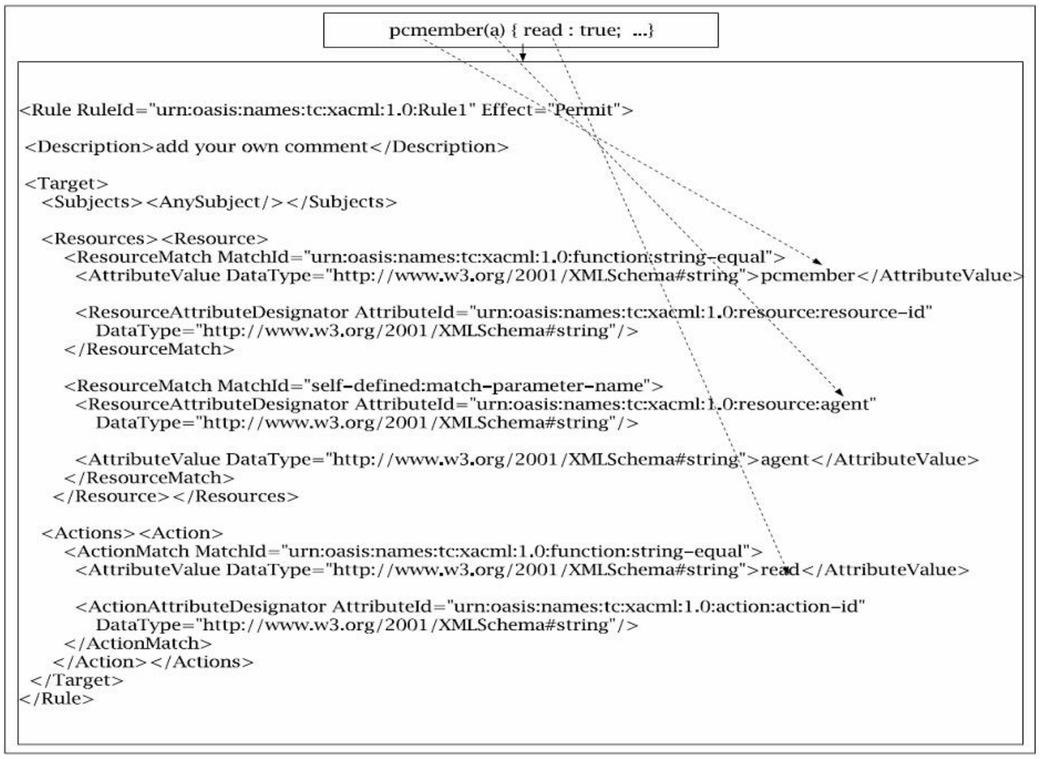
5 Ordered Binary Decision Diagram Formalization

Main References: [3] and [11].



```
AccessControlSystem Conference
        Class  Paper;

        Predicate           author(paper: Paper, agent: Agent), pcmember(agent: Agent), chair(agent: Agent),
                            reviewer(paper: Paper, agent: Agent),
                            subreviewer(paper: Paper, appointer:Agent, appointee:Agent),
                            submittedreview(paper: Paper, agent: Agent),
                            review(paper: Paper, agent: Agent);
        author(p, a){
                read : true;
        }
        pcmember(a){
                read : true;

                write : chair(user)|pcmember(a);
        }
        reviewer(p, a){
                read : pcmember(user)&~author(p, user);

                write : (chair(user)&pcmember(a)&~author(p,a)&~reviewer(p,a))
                        or ((pcmember(a)&user=a&reviewer(p,a))&~(E b: Agent [subreviewer(p,a,b)]));
        }
        subreviewer(p, a, b){
                read : (pcmember(user)&~author(p,user)) or user=b;

                write : (reviewer(p,a)&~author(p,b)&user=a&~(E d: Agent [subreviewer(p,a,d)|subreviewer(p,d,b)]))
                        | (subreviewer(p,a,b)&~submittedreview(p,b)&user=a);
        }
        ...
End
```

Figure 9: The class in RW that specifies the conference policy



```
<Rule RuleId="urn:oasis:names:tc:xacml:1.0:Rule1" Effect="Permit">

 <Description>add your own comment</Description>

 <Target>
   <Subjects><AnySubject/></Subjects>

   <Resources><Resource>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">pcmember</AttributeValue>

        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ResourceMatch>

      <ResourceMatch MatchId="self-defined:match-parameter-name">
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:agent"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>

        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">agent</AttributeValue>
      </ResourceMatch>
    </Resource></Resources>

   <Actions><Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>

        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action></Actions>
 </Target>
</Rule>
```

Figure 10: An XACML translation of one predicate from the RW conference class

In [3], our research team has worked on modeling firewall policies as Boolean expression managed as OBDDs (Ordered Binary Decision Diagrams) in order to check the policy for inconsistencies between its rules, and between rules in different policies on more than one firewall in the network. Using this representation many of the rule anomalies can be identified very efficiently. Redundant and shadowed rules can be pinpointed by comparing the policy represented as a single Boolean expression before and after a rule is removed. An equality means a redundant or shadowed rule. The same technique (with a little of detail) is applied for inter-firewall anomaly detection. Also, the same idea was extended to identify errors in configuring rules for IPSec devices [11]. In IPSec, the number of different actions the device can apply for any single packet is larger than in the case of firewalls (in Firewalls only one action: accept/deny is applied, in IPSec: accept, deny, encrypt, authenticate, encrypt/authenticate, . . .).

The use of Boolean expressions to represent the criteria section of access-list and security rules is not new. It was used before as a way to unify the representation of rules regardless of the specifics of any rule (within the same policy). For example,

$$\begin{aligned}\texttt{if } proto = TCP\ AND\ src = 11\texttt{then deny} \Rightarrow \\ \texttt{if } x_1x_3\overline{x_0x_2x_4..x_7} \wedge y_0y_1y_3\overline{y_2y_4..y_7} = \texttt{truethen deny}\end{aligned}$$

$$\begin{aligned}\texttt{if } proto = ICMP\ AND\ dst = 3\texttt{then deny} \Rightarrow \\ \texttt{if } x_1\overline{x_0x_2..x_7} \wedge z_0z_1\overline{z_2..z_7} = \texttt{truethen deny}\end{aligned}$$

Using the formulation presented we can check if both criteria apply to the exact same traffic, or do their domain overlap; all by simple Boolean operations.

The whole policy can be represented concisely into a single expression for future operations. Assuming we use a single trigger model in which any incoming packet will be matched against the policy, rule-by-rule, till a match takes place (the packet satisfy the rule's criterion), then the action of the rule is applied. No further checking against successive rules is performed. Under this assumption, the "policy domain" with an action a (P_a) is represented as a single expression as follows;

$$P_a = \bigvee_{i \in index(a)} (\neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_{i-1} \wedge C_i)$$

where $index(a)$ is the indices of the rules having a as their action.

$$index(a) = \{i | R_i = C_i \rightarrow a\}$$

6 Other Work

6.1 Efficient Comparison of Enterprize Privacy Policies

Main Reference: [4].

This paper presents an efficient algorithm that compares two privacy policies and checks if one of them refines the other. In other words, Refinement is equivalence with a different level of abstraction. For an enterprize, policy refinement means that there is a given policy that this enterprize has to a adhere to, and policy refinement is to generate a policy that conform with this given policy and adds extra requirements or customizations to the enterprize. So, one of the policies is more abstract, and relaxed than the one that refines it.

IBM (with which three of the four authors are affiliated) has proposed Enterprize Privacy Authorization Language (EPAL) to be used for these purposes. It is an XML specification and is still open for suggestions and standardization. As in Or-RBAC roles, organizations, users, data are modeled in Hierarchies with inheritable Obligations and Conditions.

EPAL is given by a vocabulary, a list of authorization rules, a global condition, and a default ruling. The vocabulary defines element hierarchies for data, purposes, users, and actions, as well as the obligation model and the condition vocabulary. Data, users and actions are as in most access control policies, and purposes are an important additional hierarchy for the purpose binding of collected data. The vocabulary itself is a tuple: $Voc = (UH, DH, PH, AH, Var, OM)$ where UH, DH, PH, and AH are hierarchies called user, data, purpose, and action hierarchy, respectively, Var is a condition vocabulary, and OM an obligation model. The condition vocabulary is defined by a set of variables and their corresponding scope $Var = (V, Scope)$.

6.2 Universal Network Traffic Classification Policy

Main Reference: My work, not yet published.

Most of the current research work on the analysis of firewall, IPSec, ... policies include implicitly the filtering/classification algorithm within the analysis technique itself. Thus, generalizing the analysis technique to other security devices other than the one initially thought of is very hard and needs many modifications (if even possible). For example, modeling single and multi trigger in the same policy format is quite challenging. It shows a drastic change in the inter-rule dependency and in the way the filtering algorithm processes the rules themselves.

So, I proposed a way to write any policy (currently, firewalls, IPsec and IDSs are supported easily), as list of rules, and an associated order: $P =< r_1, o_1 >, < r_2, o_2 >, \ldots < r_n, o_n >,$. Each rule is a tuple that recursively might contain another policy (another list of rules with their order). Any single rule r is defined as:

$$r :=< C, a, i, P >$$

where C is the condition of matching, a is an action from the repository of possible actions, i is an "ignore action" Boolean flag, and P is another policy (a list of rules) to be processed if the packet matched this rule.

To process a packet, all rules are checked in order (*i.e.*, 1,2,... $|P|$) till a rule has its Condition satisfied. The action is applied to the packet (if the i flag is *true* the new packet is discarded and the original is used instead) and then recursively the same is repeated using the Policy mentioned within the matched rule. Analyzing the correctness of the policy using this representation and simple algorithm is the same for whatever origin the policy was taken from.

For firewalls, the root policy has all its rules without a sub-policy (the policies mentioned within each rule is an empty list). When a match occurs, the action is applied (either "no-operation", or "forward packet to correct interface"), and then the algorithm halts.

For IPSec policies with multi-trigger, each rule has all the other rules with higher order (*i.e.*, coming next) in its own sub-policy. The size of the policy this way (if rules are actually repeated) is $O(n^2)$ such that the original was of size n. However, this explosion in size is justified and it can not be higher than this whatever the classification algorithm was (from all the possible ideas we thought of in classification algorithms, n^2 is the highest possible).

7 Conclusion

As long as the topic is not totally exhausted and the motivation of finding better solutions is still pushing, the future work is still to collect more of the research production available and find more ways to exploit the huge matrix of formal tools into the vacant spots in between.

Having a complete model that will involve every possible aspect of security policies might be possible, but then the enforcement of such a system will incur an overhead that will make it far more expensive than going with less security with older models. Having simpler models that are practically feasible to deploy is a must. The solution might be in thinking about simpler individual systems (*e.g.*, firewalls, IDS sensors, file system access rights, digital media copyright enforcement, ...) than trying - from there - to generalize such models to bundle more than one application at the same time.

Another approach is going deep into theory and think of new models that satisfy the high level requirements. However, we have to be patient and not keep searching for applications for these evolving models, let them ripe a little first before we twist them to fit our ill-thought-of applications.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993. 3

[2] G. Ahn and M. E. Shin. Role-based authorization constraints specification using object constraint language. In *WETICE '01: Proceedings of the 10th IEEE Intl. Workshops on Enabling Technologies*, pages 157–162, Washington, DC, USA, 2001. IEEE Computer Society. 7

[3] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM*, 2004. 15, 18

[4] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. Efficient comparison of enterprise privacy policies. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 375–382, New York, NY, USA, 2004. ACM Press. 19

[5] J. Barkley. Comparing simple role based access control models and access control lists. In *RBAC '97: Proceedings of the 2nd ACM workshop on Role-based access control*, pages 127–132, New York, NY, USA, 1997. ACM Press. 4, 5

[6] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Mige. A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust,Toulouse, France*, 2004. 13

[7] F. Cuppens and A. Mi&#232;ge. Modelling contexts in the or-bac model. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 416, Washington, DC, USA, 2003. IEEE Computer Society. 5, 7

[8] F. Cuppens and A. Miège. Administration model for or-bac. In *OTM Workshops*, pages 754–768, 2003. 5, 7

[9] D.Ferraiolo, J.Cugini, , and D.R.Kuhn. Role based access control (rbac): Features and motivations. In *Proceedings of Computer Security Applications Conference*, pages 241–248, December 1995. 4

[10] A. El-Kalam, S.Benferhat, A. Miège, R. El-Baida, F. Cuppens, C. Saurel, P. Balbiani, Y.Deswarte, and G. Trouessin. Organization based access control. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 120, Washington, DC, USA, 2003. 5

[11] H. H. Hamed, E. S. Al-Shaer, and W. Marrero. Modeling and verification of ipsec and vpn security policies. In *ICNP*, pages 259–278, 2005. 4, 15, 18

[12] E. J. Khayat and A. E. Abdallah. A formal model for flat role-based access control. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*. 4, 5

[13] C. E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, 1981. 3

[14] T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *UML 2002 - The Unified Modeling Language : 5th International Conference*. 7, 11

[15] M. Pradella, M.Rossi, and D. Mandrioli. A uml-compatible formal language for system architecture description. In *SDL 2005: Model Driven Systems Design: 12th International SDL Forum*, volume 3530 of *Lecture Notes in Computer Science*, pages 234–246. Springer Berlin / Heidelberg, June 2005. 7, 12

[16] I. Ray, N. Li, R. France, and D. Kim. Using uml to visualize role-based access control constraints. In *SACMAT '04: Proceedings of the 9th ACM symposium on Access control models and technologies*, pages 115–124, New York, NY, USA, 2004. ACM Press. 7

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996. 4

[18] R. Wies. Policies in network and systems management - formal definition and architecture. *Network and Systems Management*, 2(1):63–83, 1994. 3

[19] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems in xacml. In *FMSE '04: Proceedings of the 2004 ACM*

workshop on Formal methods in security engineering, pages 56–65, New York, NY, USA, 2004. ACM Press. 13, 14