Attribute-based Mining Process for the Organization-Based Access Control Model

Ahmad Samer Wazan, Gregory Blanc, Hervé Debar and Joaquin Garcia-Alfaro
Institut Mines-Telecom, Telecom Sudparis
CNRS Samovar UMR 5157, Evry, France
{samer.wazan,gregory.blanc,herve.debar,joaquin.garcia_alfaro}@telecom-sudparis.eu

***Abstract*—Since the late 60's, different security access control models have been proposed. Their rationale is to conceive high level abstract concepts that permit to manage the security policies of organizations efficiently. However, enforcing these models is not a straightforward task, especially when they do not consider the reality of organizations which may have ad-hoc security policies already deployed. Another issue is the vagueness of their abstract concepts. We propose to bridge the gap between the theory of access control models and the reality of organizations by defining an attribute-based mining process that deduce the abstract concepts starting from the attribute level. Additionaly, the attributes allow us to semantically enrich the obtained results. We have selected the Organization-Based Access Control (OrBAC) model as the abstraction objective of our study.**

***Index Terms*—Security, Policy Management, Access Control, Role Mining.**

I. Introduction

A security policy defines all the actions that need to be enforced in order to ensure the security of the assets of an organization. Regarding the information systems of an organization, different security mechanisms can contribute to their protection and safety. In particular, access control mechanisms enforce the permissions of individuals to access resources. An access control system is composed of three entities: access control *policy*, access control *mechanism* and access control *model*. The objective of access control *policies* is to determine whether a *subject* has the right to execute an *action* on an *object*. Access control *policies* are enforced through security *mechanisms*, like Firewalls. Access control *models* are usually used to analyze and evaluate the access control systems. A *model* is a formal presentation of the security policy enforced by the system and is useful for proving theoretical limitations of a system [1].

There exist different security *models* (e.g., RBAC [2], OrBAC [3] and ABAC [4]). The main idea of these *models* is to conceive high level abstract concepts (e.g., Roles, Sessions, Views and Activities) that permit to manipulate the organizations' policies at a higher level. However, applying these models in order to manage the security policies is not a straightforward task, for two reasons:

- *migration problem*: for an organization that has already an ad-hoc security policy deployed on different security mechanisms, it would be difficult to adopt one of the access control models.
  Manual work in order to read the existing rules is a cumbersome task. The difficulty depends on the number of the security rules present in the different security mechanisms.
- *semantic problem*: the semantics of abstract concepts are often vague, leading to different interpretations of these concepts (e.g. roles, views and activities). Indeed, administrators often have hardships to capture the meaning of abstract concepts and designing them may fail to comply with the chosen model. If we consider a firewall, how do we model a subject? Can it be an IP address or would it be better to consider the pair (IP address, port)? What are the actions implemented by the configuration rules? Allow or block packets? Log packets? What is an appropriate role with regards to firewalls? A range of IP addresses?
  The task of enforcing a security model raise many philosophical questions about the nature of concrete entities or how to design abstract entities.

This makes the whole task cumbersome and time-consuming, even when security mechanisms do not have already deployed security policies. There is a need to bridge the gap between the theory of access control models and the reality of organizations. The persistence of this gap constitutes a real obstacle for the organizations that want to adopt one of the access control models. This observation is based on our experience acquired during the implementation of an inference engine for the OrBAC model.

One can consider that handling these problems fall under the domain of role engineering. This is from one side partially correct because our approach aims to deduce abstract concepts (such as roles) from low-level configuration rules. On the other side, our approach is different because it does not start from the couple $< user, permissions >$ which is the pillar of that domain. Our approach proposes to start one level below, which is the attribute level. We consider that the rules of all security mechanisms (such as firewalls, operating systems, databases, etc.) can be modeled as sets of attributes along with decisions.

Traditionally, in the access control area, security mechanisms have been considered as pure access control mechanisms, particularly firewalls. However, this point of view does not permit to model all the functions provided by security mechanisms. We propose to utilize the concept of *obligations* to specify actions to be executed before, during or after access control requests. For example, the *log* action provided by most

stateful firewalls is an *obligation* that should be performed along with the access control decision. Thus, each security mechanism would be modeled as two related matrices: one being the access control matrix which is a set of attribute vectors associated with decisions (accept or deny), the other being the obligation matrix which is also a set of attribute vectors associated with decisions (oblige or dispense).

Once security mechanisms are formally modeled, we propose a mining approach that starts the processing from the attribute level. The process deals with security mechanisms by grouping them into classes (e.g., routers, firewalls, operating systems and database management systems). Members of a security mechanism class share more or less the same functions so that it is possible to express them in a unified manner. Each class is expressed using a specific set of attributes that allow to rewrite the configuration rules. An administrator selects a model that maps the attributes to the concrete entities (subject, action, object, context). An algorithm has been proposed to deduce the abstract concepts (role, activity, view). With contrast to previous algorithms proposed for role mining, our algorithm mines both the users and the permissions so that the obtained concepts reflect exactly the security policy of the organization. Additionaly, we leverage the concept of attributes by performing correlation analysis that permits to attach semantic information to the obtained results.

Paper organization — Section II briefly presents a selection of work in the domain of role mining. Section III presents the OrBAC model and introduces the concepts of *contexts* and *obligations* handled by this model. Section IV shows our approach, based on the definition of security mechanisms in terms of attribute matrices. Section V presents the different stages of our proposed attribute-based mining process and the algorithmic solution to deduce the abstract concepts. Section VI closes the paper with some conclusions and perspectives for future work.

II. Related Works

Our work has close relation with the domain of role engineering. Two basic approaches have been considered in this domain: top-down and bottom-up approaches. In the top-down approach, the roles are defined from business perspective. Several research works [5], [6] , [7] have focused on the top-down approach. However, according to the NIST report, this approach is costly and time-consuming process [8].

The bottom-up approach defines roles by analyzing the existing permissions using data mining techniques. The term role mining is often used to refer to this approach. Starting from 2003, different works have been proposed [9] [10] [11] [12] [13]. Kuhlmann et al. [9] have adopted IBM Intelligent Miner as the mining engine of the security data. The tool enables to select the data portions to be analyzed, recognize invalid data, iterate the mining processes until an acceptable result is reached, and finally produce role definitions based on the attained grouping. Schlegelmilch and Steffens [10] proposed the ORCA algorithm for role mining. The algorithm performs a cluster analysis on permission assignments to build a hierarchy of permission clusters. However, as the ORCA algorithm does not allow overlapping roles (i.e., a user cannot play multiple roles), Vaidya et al. [11] propose an approach based on subset enumeration, called RoleMiner, which eliminates these limitations. Subsequently, Vaidya et al. [12] have proposed a formal definition for the role mining problem. The salient idea from the given definition is the necessity to minimize the number of the mined roles. Two different algorithms (δ-*approx RMP* and *Minimal Noise RMP*) have been defined to reduce the number of the mined roles.

Two main problems come from the mining techniques that have been adopted by these approaches. Firstly, instead of discovering the roles that are naturally defined as sets of shared permissions among subjects (users) in organizations security policies, they define roles by optimizing combinations of permissions based on different heuristics. Thus, the mined roles often fail to recover the original security policy of the organization [12] and generate roles that are not useful to the organization. To cover this problem, different works have proposed to define a set of metrics used to inform the administrators about the utility of the mined roles in their organizations [14].

Secondly, since these mining techniques do not consider the attributes and start mining from the couples $< user, permissions >$, they are not able to semantically define the obtained roles. By starting one level below (attribute level), we claim the ability to attach semantic information to the mined concepts.

III. Organization-based Access Control Model

Since the late 60's, several access control models have been proposed. Generally, the proposed access control models are classified in three categories: discretionary access control (DAC) [15], mandatory access control [16], [17] and role-based access control (RBAC) [2]. Discretionary access control means that the access rights to an object are managed by its owner, whereas mandatory access control (MAC) means that the access rights are managed by a central authority and not by the owner of the object. RBAC is non-discretionary model, but in the literature it is considered as independent category of access control models [1].

Many extensions to RBAC have been widely proposed in the literature. Organization-Based Access Control (*OrBAC*) model [3], [18] is one of the most interesting models because it gives a high consideration to the concept of *context*. In addition to the *Role* concept, which is an abstract concept used to group subjects which share the same permissions, *OrBAC* proposes two new abstract concepts: *View* and *Activity*. The concept of *View* is used to group objects on which the same permissions are applied whereas the concept of *Activity* is used to group actions that share the same permissions. The model supports *hierarchies*, *delegation* [19] and *conflict detection and resolution* [20]. Additionally, OrBAC supports the concept of *Obligation* [21], which is the set of actions that a subject is required to execute before, during or after access control. Thus,

Obligations can be used to express the requirements that a subject must execute.

In OrBAC, a policy is defined at the abstract level (also known as the *organizational* level) and then expressed at the concrete level by a derivation process. The concrete entities (*subject*, *action*, *object*) are bound by conditions to an access control decision (*permission* or *prohibition*). The conditions that apply to subjects, actions and objects correspond to the facts that, in a given organization (org), a subject (s) is *empowered* into a *role* (r), an action (α) is *considered* to implement an *activity* (a) and an object (o) is *used* in a *view* (v). These relations are represented by the following built-in predicates:

$$Empower(org, s, r) \tag{1}$$

$$Use(org, o, v) \tag{2}$$

$$Consider(org, \alpha, a) \tag{3}$$

The concept of context in the OrBAC model defines when the security rules should be activated. The contexts are specified using the predicate *Hold*:

$$Hold(Org, s, \alpha, o, Ctx) \leftarrow P_1, ..., P_n \tag{4}$$

The organization *Org* considers that the context Ctx holds for a subject s taking an action α on an object o if the set of conditions $P_1, ..., P_n$ are true. For instance, we can define a context $secure\_area\_network$ that activates when a subject s connects from a secure area network:

$$\begin{aligned} &Hold(Trustedbank, s, \alpha, o, secure\_area\_network) \\ &\quad \leftarrow host\_IP(s, ip) \wedge IP\_range(ip, SA) \end{aligned} \tag{5}$$

The context holds for a subject s if it is assigned the IP address ip (by the predicate $host\_IP$) and this IP address falls within a network range designated as a secure area (SA). $IP\_range(ip, range)$ checks the membership of ip to $range$.

The OrBAC derivation process from the abstract level to the concrete level is specified using the following rule:

$$\begin{aligned} &Is\_permitted(Org, s, \alpha, o) \leftarrow \\ &\quad Permission(Org, r, a, v, Ctx) \wedge \\ &Empower(Org, s, r) \wedge Consider(Org, \alpha, a) \\ &\quad \wedge Use(Org, o, v) \wedge Hold(Org, s, \alpha, o, Ctx) \end{aligned} \tag{6}$$

With regards to obligations, Elrakaiby et al. discussed in [21] the use of group obligation actions based on the OrBAC model. Obligation actions may represent prerequisites to gain some privilege (pre-obligations), to satisfy some ongoing or post requirement for resource usage (ongoing and post obligations). Two main types of obligations have been defined: system and user obligations. System obligations are generally enforced by mechanisms implemented in the system. On the other hand, user obligations are actions that subjects are required to take in the future [21]. Since subjects cannot be forced to take actions, user obligations are unenforceable and should be monitored for violation/fulfillment. However, system obligations do not need to be monitored.

R1	AAT1	AAT2	AAT3	AAT4	...	Decision1
R2	AAT1	AAT2	AAT3	AAT4	...	Decision2
R3	AAT1	AAT2	AAT3	AAT4	...	Decision3
...	...	...	...	...	...	...

Fig. 1. Access Control Matrix

The obligation rules can take the following form:

$$Obligation(Org, r, a, v, Ctx_a, Ctx_v) \tag{7}$$

Ctx_a and Ctx_v are obligation context expressions. Ctx_a denotes the obligation's activation context and Ctx_v the obligation's violation context. In this paper, we are interested only in the system obligations; in this case the subject entity is always the security mechanism we are studying, and the violation context is always considered false. Thus, there is no need to monitor system obligations.

IV. Modeling a Security Mechanism from Access Control Perspective

A security mechanism is the system that enforces the security policy. In the access control jargon, a security mechanism is referred to as Policy Enforcement Point (PEP) [22]. The policies of security mechanisms can be expressed as a matrix of attributes that are associated with decisions.

Each security mechanism manipulates *rules* that define the *authorizations* of a *subject* (s) to perform an *action* (α) on an *object* (o). In these mechanisms, a request represented as a triplet $< s, \alpha, o >$ is matched to attributes that constitute the conditions to be satisfied in order to take the associated decision. Thus, a security mechanism is a set of rules R, that is a possibly *ordered* list of rules. Each *rule* R_i has the form:

$$\forall s, \forall \alpha, \forall o, (Condition \rightarrow Decision)$$

while, the *Decision* part of the relation is usually reduced to *allow* or *deny* (but other values can apply such as *undefined*), *Condition* can be further modeled as:

$$\begin{aligned} &cond\_subject(s) \wedge cond\_action(\alpha) \wedge \\ &\quad cond\_object(o) \wedge contexts(s, \alpha, o) \end{aligned}$$

where $cond\_subject(s) \wedge cond\_action(\alpha) \wedge cond\_object(o)$ represent the set of entity-related attributes with their associated values. $contexts(s, \alpha, o)$ represents the set of attributes that deal with the context during which the configuration rule is activated. Here, we make the assumption that such logical conditions can be built upon entity attributes, such that each concrete entity c is an attribute vector $(c_1, ..., c_n)$. Each attribute is a property expressed as a name:value pair that can be associated with concrete entities(i.e. subjects, actions and objects) as well as the contexts. Generally, the attributes express capacity of the PEPs to implement access control related functions.

It is not totally true to consider each PEP as a pure access control mechanism; there exist associated functions that are dependent of the access control functions. The access control-dependent functions are executed before, during or after the

O1	OAT1	OAT2	OAT1	OAT4	…	Decision1
O2	OAT1	OAT2	OAT2	OAT4	…	Decision2
O3	OAT1	OAT3	OAT3	OAT4	…	Decision3
…	…	…	…	…	…	…

Fig. 2. Obligation Matrix

access control requests. We consider these kinds of functions as *obligations*, where each *obligation* is associated with three entities: subject, action and object. The subject entity is always the security mechanism (e.g. firewall).

As a consequence, each PEP can be represented using two matrices:

- the access control matrix (Fig. 1) is composed of the set of attributes that give arise to concrete entities used in the access control rules: subject, action and object. The associated decisions are usually : allow, deny.
- the obligation matrix (Fig. 2) is composed of set of attributes that give arise to concrete entities used for defining the obligations associated with the access control rules. The associated decisions are usually: oblige or dispense.

The relation between the matrices can be defined using the activation contexts that define the moment at which the obligations should be applied.

The matrix approach (Figures 1 and 2) implements the concept of attribute-based entity definition. Therefore, these entities are defined for each line of the matrix (corresponding to a single rule) as subsets of the attributes. Formalizing the mapping between concrete entities and rule attributes relies on a model. However, following one's point of view, the model may differ, though some mappings are semantically consistent while others are not. Actually, an expert can define a set of constraints over the mappings to express mutual exclusion or association relationships between the attributes. Other environmental phenomena can be taken into account to further constrain or loosen up relationships between attributes, as well as the possible mappings. This leads to a finite but possibly large number of models to express concrete entities using PEP rule attributes.

In the remainder of the paper, we will assume that an administrator, responsible for the PEP mining process, decides on the most adequate mapping to model the concrete entities of the PEP policy.

V. Attribute-based Mining Process for PEP Policies

Applying access control models to manage the security policies of organizations having security policies already deployed is not an easy task. In most cases, the policies of organizations are defined in an ad-hoc manner. Two main parameters characterize the difficulty of the task: the *number of rules*, and the *number of attributes* present in each rule. Thus, a manual work in order to read the existent rules and to translate them towards the abstract concepts of any access control model is difficult to realize. Additionally, the semantics of abstract concepts of access control models are often vague, leading to different interpretations of these concepts.

We handle these problems by proposing to infer from low level existing rules the high level concepts of the OrBAC model. The bottom-up inference process deals with PEPs by grouping them into classes (e.g., firewalls, operating systems, MPLS routers, etc.). Our idea is that each class of PEPs shares more or less the same functions so that the same inference strategies can be applied to each class. For example, all the operating systems should provide functions that enable users to implement access control actions (read, write and execute) on the files, but they use different syntax to implement these functions.

The proposed bottom-up inference process relies on the architecture depicted in Figure 3. For a given PEP whose class is known, we can apply the following process:

1) *parsing*: using the common language dictionary, the PEP configuration rules are parsed into attribute vectors;
2) *data preprocessing*: in this step, the data of the PEP is prepared before deducing the concrete entities;
3) *modelization of concrete entities*: by grouping attributes into concrete concepts according to a given paradigm, we generate models of concrete entities that maps configuration selector values to concrete entity attributes. The output is the set of concrete entities, authorizations, obligations and contexts;
4) *discovery of abstract entities*: once concrete entities have been discovered, we infer abstract entities, authorizations and obligations by grouping concrete entities that share exactly the same permissions.

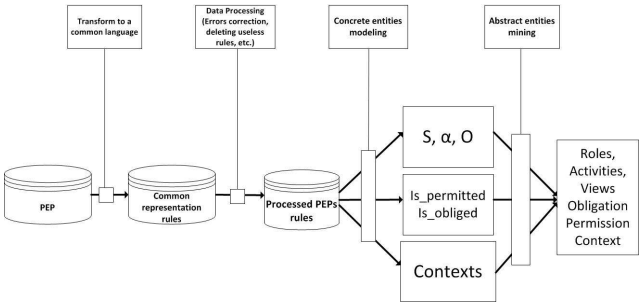

Fig. 3. Attribute-based Mining Process

A. Parsing Configuration Rules

The common language is a key element of the inference process. It enables giving a common representation of the policies of different PEPs, provided they share common functions. PEPs are categorized by classes. A class denotes a set of common functions implemented by all members of the class, such as firewalls, operating systems, enterprise directories, etc. Figure 4 is an excerpt of a firewall common language dictionary.

Using this dictionary, we can parse rules from different firewall vendors, provided we can translate the specific firewall

#	Attributes	Values
1	Source and Destination IP Address	IP address, Wildcard, Block
2	Interface	Identifier
3	Interface direction	In, out
4	Protocol	TCP, UDP, ICMP, Number, Wildcard
5	Source and Destination Port	Number, Range: [p1,p2], Identifier, Wildcard
6	ICMP Type	Number, Identifier
7	Non-AC Action	Log, Mark, Nat, ...
8	Decision	Allow, Deny, Oblige, Dispense
9	Connection State	New, Established, Related, Invalid

Fig. 4. Firewall common language

language to the firewall class common language (this translation task is out of the scope of this work). For example, let us consider the same rule written in both `pf` language:

$$\underbrace{pass}_{8}\ \underbrace{in}_{3}\ \underbrace{log}_{7}\ \underbrace{on\ \$ext\_if}_{2}\ \underbrace{proto\ tcp}_{4}\ \underbrace{to\ !\ <firewall>}_{1}\ \underbrace{port\ ssh}_{5} \quad (8)$$

and `iptables` language:

$$iptables\ \underbrace{-A\ INPUT}_{3}\ \underbrace{-i\ \$ext\_inf}_{2}\ \underbrace{-p\ tcp}_{4}\ \underbrace{-d\ !\ <firewall>}_{1}\ \underbrace{–dport\ 22}_{5}\ \underbrace{-j\ LOG}_{7} \quad (9)$$

This rule logs packets incoming on the external interface and bound to hosts on port 22, excluding the firewall itself. The numbers correspond to the parsing index of the common language dictionary in Figure 4. According to this dictionary, the rule is interpreted as follows:

$$src\_ip(s_{ip}, any) \wedge src\_port(s_p, any) \wedge if\_dir(dir, in) \wedge if(inf, \$ext\_inf) \wedge proto(p, tcp) \wedge \neg dst\_ip(d_{ip}, <firewall>) \wedge dst\_port(d_p, 22) \wedge action(a, log) \rightarrow is\_permitted$$

The expression varies from language to language, mostly by the syntax. A notable difference is the decision (8) which is explicite for `pf` and is not visible in the `iptables` rule. Actually, the rule in `iptables` only features the non-AC action (7), which is a non-terminal target, i.e., the packet matching the rule will be later processed by another rule. Here, we assume the packet will be eventually processed by an `ACCEPT` target, which is similar to `pf`'s `pass`.

It should be noted that the expressiveness of the common languages depends on the shared functions between the PEPs belonging to the same class. For example, if an organization uses three different types of firewalls, one of which does not support the audit function, the common language should not contain attributes representing the audit functionality. However, if an organization uses three firewalls from the same type, e.g. `iptables`, then the common language should be more expressive (encompassing all functions implemented in `iptables`).

In other words, the expressiveness of a common language must be adaptive to the organization's environment. The more homogeneous the environment, the more expressive the language will be. Thus, the transformation step must be executed in a semi-automatic manner to give an environment-adaptive common language. In this step, the functions of known PEPs (e.g., CISCO PIX, `iptables`, `pf` for the firewall class) are stored by the process so that the common language can be generated automatically. However, an administrator of an organization must be authorized to declare the functions of unknown PEPs (e.g., a custom-made firewall) in order to generate a common language appropriate to the administrator's environment.

The uncommon functions can be managed at the level of each PEP. The attributes relative to the uncommon functions are extracted and attached to the rule just before applying it to the concerned PEP. We represent this transformation as follows:

$$is\_permitted(s, a, o) \wedge additional\_attributes(s, \alpha, o) \rightarrow is\_permitted(s\prime, \alpha\prime, o\prime)$$

The main idea is that the triplet $< s, \alpha, o >$ constitutes a key, used to look up the additional attributes and apply the transformation. It should be noted that some of the uncommon functions can be "emulated" by the other PEPs. For example, the possibility for a firewall to represent a range of ports in its rules should not be considered as uncommon since it can be emulated by other firewalls using their available syntax.

The output is a set of attribute vectors representing each a configuration rule.

B. Data Preprocessing

The objective of this step is to prepare the data of the PEP before deducing the concrete entities. This mainly includes the correction of errors. PEPs can have several anomalies in their configuration rules. This situation results from errors made during successive configuration of a PEP, especially when different administrators are involved in its management over the years. If not corrected, these errors could compromise the security of organizations.

Three main types of errors can be detected:

- Conflict errors: these errors are raised when two opposed decisions are detected for the same configuration rule. For example, when a firewall associates both $allow$ and $deny$ decisions to the same traffic.
- Shadowing errors: A configuration rule R_i is shadowed in a set of configuration rules R when such a rule never applies because all the information that the rule R_i matches are already matched by another set of rules [23], [24].
- Redundancy errors: A configuration rule R_i is redundant in a set of configuration rules R when the following conditions hold: (1) R_i is not shadowed by any other rule or set of rules; (2) when removing R_i from R, the security policy does not change.

These errors must be detected and corrected. There exist in the literature many proposals to resolve these problems [25]. We consider that developing new algorithms is out of the scope of this paper.

OrBAC concrete entities									
Subject		**Action**					**Object**		**Context**
-s	–sport	-p	tcp udp icmp	-i	-o	-m state	-d	–dport	-m time
Source		**Protocol**		**Network Interfaces**		**Connection State**	**Destination**		Periodic
iptables rules fields									

TABLE I
A MODEL OF CONCRETE ENTITIES FOR A FIREWALL INSTANCE (IPTABLES)

```
iptables -A FORWARD -p udp -s 192.168.1.0/24
-d 192.168.0.0/24 -m state --state ESTABLISHED,RELATED
-m time --timestart 08:30 --timestop 18:00
--days Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

Fig. 5. An iptables rule

C. Modeling Concrete Entities

As mentioned previously in Section IV, the modelization of the concrete entities is done by mapping the selectors of a PEP language to the concrete entities. A given model relies on a given mapping of the selectors as attributes of the concrete entities. The model may follow a paradigm: a fixed association between a selector and an entity attribute, that constrains the other associations by mutual exclusion or dependency relations. These relations may be explicitely described in a specification language.

For example, considering the class of firewalls, we may start our modelization by considering either of the following paradigms (the list is not exhaustive):

- the action is a service: the entity action is constituted of the protocol and destination port attributes;
- the object is a service: the entity object is constituted of the protocol and destination port attributes;
- the subject is a packet: the entity subject is constituted of the source port, protocol and other header attributes;
- the subject is the originating host: the entity subject is constituted of the source IP address and port.

Following a model, each parsed configuration rule will generate attribute vectors that will be mapped to the concrete entities according to the paradigm and the constraint rules. Therefore, for a given ruleset, there exist many modelizations of the concrete entities. The administrator is responsible for the choice of the model that best fits his needs and/or views.

Table I is an excerpt of the table mapping firewall selector values (here for the sole `iptables` instance) to OrBAC concrete entities (subject, action, object). This is indeed one model among many. One drawback is that this model freezes the mapping for each configuration rule. In the future, we will propose several ways to loosen up the models and generalize the expression of models through the constraints imposed on attributes as a formal expression of the paradigms.

As an example of application, let us consider the `iptables` rule in Figure 5 that allows UDP connections originating from the 192.168.1.0 network to the 192.168.0.0 network only during working hours and week days.

According to Table I, the concrete entities modeled from the above rule (Fig. 5) are as follows:

subject: $ip(s_i, 192.168.1.0/24) \wedge port(s_i, ANY)$
action: $proto(\alpha_j, UDP) \wedge$
$(state(\alpha_j, ESTABLISHED) \vee state(\alpha_j, RELATED))$
object: $ip(o_k, 192.168.0.0/24) \wedge port(o_k, ANY)$

The attributes beyond `-m time` are mapped to the context entity. This is actually a temporal context that constraints the expression of the permission to week working hours. The context $week\_working\_hours$ can be expressed as follows:

$$week\_working\_hours = after\_time(08:00) \& before\_time(18:00) \& \overline{(on\_day(saturday) \& on\_day(sunday)}$$

D. Expressing Additional Requirements

Most security policies describe access control requirements to be enforced by PEPs. But their scope is more general and encompass additional requirements bound to usage, system state or other environmental properties, as well as dissemination of target objects and effects of decisions over the subjects and objects [26].

In the context of our approach, we lack much information on the deployment context and we assume no written policy since our goal is to infer the policy of an organization from the configurations of its deployed PEPs. Much of the framework components defined in related works are useless, and we shall not consider anything but requirements bound to the system (PEP) we are studying, and its configuration.

In general, PEP configuration rules do not only express access control policies at a low-level but also specify some additional requirements to be satisfied in order to get access. These requirements specify actions to be taken by the system which may or may not modify the attributes of subjects and objects.

Our approach does not ignore such behavior and processes these requirements along the related authorizations, i.e., authorizations that feature subjects and objects whose attributes have been updated by the requirements. We propose to use the concept of system obligations (as specified in [21]) presented in Section III. Therefore, the *Obligation* (as expressed in Eq. 7) can be written as follows at the concrete level:

$$Is\_obliged(PEP, ob\alpha, obo) \tag{10}$$

The PEP is *obliged* (resp., *dispensed*) to execute the action $ob\alpha$, which is not an access action, on the object obo, which may be constituted from entity attributes of the related permission $Is\_permitted(s, \alpha, o)$ (resp., the related prohibition $Is\_prohibited(s, \alpha, o)$). The relationship between the obligation and the permission is not formally defined. This is why we use the context of activation (as expressed in Eq. 7) to

bind the two predicates. The context of activation is therefore based on the request of the permission, the triplet $< s, \alpha, o >$:

$$Is\_permitted(s, \alpha, o) \Rightarrow Is\_obliged(PEP, ob\alpha, obo) \quad (11)$$

$$Ctx_a \leftarrow Conditions(s, \alpha, o) \quad (12)$$

As a practical example, we can consider the `pf` rule (see Eq. 8) once again. This rule not only expresses a permission as stated previously, but also specifies the additional requirement of logging the packets allowed by the permission. Hence the following obligation associated to the permission derived from the `pf` rule:

$$Is\_obliged(FW, LOG, < allowed\_packets >) \quad (13)$$

With $< allowed\_packets >$ being expressed as the following combination of attributes:

$$src\_ip(s_{ip}, any) \wedge src\_port(s_p, any) \wedge if\_dir(dir, in) \wedge$$
$$if(inf, \$ext\_inf) \wedge proto(p, tcp) \wedge$$
$$\neg dst\_ip(d_{ip}, <\textit{firewall}>) \wedge dst\_port(d_p, 22)$$

E. Inferring Abstract Entities

As mentioned earlier, *Roles* (resp., *Views* and *Activities*) are sets of subjects (resp., objects and actions) that share the same permissions. Sandhu et al. [2] demonstrate that the role concept is close to the group concept. In fact, the role is a set of subjects on one side and a set of permissions on the other, whereas groups are defined as a set of subjects only.

Per analogy with RBAC, OrBAC features a role concept defined for subjects but also a role concept in relation to actions, which is the concept of *Activity* as well as a role for objects, the concept of *View* [18]. Thus, deducing a *Role* (resp., an *Activity* and a *View*) consists in identifying the subjects (resp. actions and objects) that share the same set of permissions.

The permissions from the perspective of roles are based on the actions and objects, they can be defined as follows:
$PER_{role} = \{(decision_k, o_j, \alpha_l)_{s_i} : \forall decision_k \in \{Accept, Deny\}, \forall o_j \in O, \forall \alpha_l \in A, \forall s_i \in S\}$

The permissions from the perspective of views are based on the subjects and actions, they can be defined as follows:
$PER_{view} = \{(decision_k, s_i, \alpha_l)_{o_j} : \forall decision_k \in \{Accept, Deny\}, \forall s_i \in S, \forall \alpha_l \in A, \forall o_j \in O\}$

The permissions from the perspective of activities are based on the subjects and objects, they can be defined as follows:

TABLE II
SAMPLE SET OF PERMISSIONS

R1	s_1	α_1	o_1	accept	R9	s_2	α_2	o_3	accept
R2	s_1	α_1	o_2	accept	R10	s_2	α_3	o_2	accept
R3	s_1	α_1	o_3	accept	R11	s_2	α_3	o_3	accept
R4	s_1	α_2	o_1	accept	R11	s_3	α_1	o_1	accept
R5	s_1	α_3	o_1	accept	R13	s_3	α_1	o_2	accept
R6	s_2	α_1	o_1	accept	R14	s_3	α_1	o_3	accept
R7	s_2	α_1	o_2	accept	R15	s_3	α_2	o_1	accept
R8	s_2	α_2	o_2	accept	R16	s_3	α_3	o_1	accept

$PER_{activity} = \{(decision_k, s_i, o_j)_{\alpha_l} : \forall decision_k \in \{Accept, Deny\}, \forall s_i \in S, \forall o_j \in O, \forall \alpha_l \in A\}$

Algorithm `GetAbstractEntities` summarizes our proposed process to infer the abstract entities. The main task of the algorithm is to identify the set of permissions of each concrete entity (i.e., subjects, actions or objects) and deduce abstract entities by detecting concrete entities that strictly share the same set of permissions.

Algorithm 1 $A \leftarrow$ `GetAbstractEntities`(C, PER)

Input: C /* set of concrete entities */
Input: PER /* set of permissions */
Output: A /* set of obtained abstract entities */
$PERM_{c_i} \leftarrow \emptyset$
$A \leftarrow \emptyset$
for all $c_i \in C$ **do**
  $new\_abstract\_entity \leftarrow$ **true**
  for all $per_k \in PER$ **do**
    if $per_k.c_k == c_i$ **then**
      $PERM_{c_i} \leftarrow per_k$
    end if
  end for
  for all $a_j \in A$ **do**
    if $PERM_{c_i} == a_j.PERMS$ **then**
      $a_j.CONCRETE \leftarrow a_j.CONCRETE \cup c_i$
      $new\_abstract\_entity \leftarrow$ **false**
    end if
  end for
  if $new\_abstract\_entity$ **then**
    $a_{length(A)+1}.PERMS = PERM_{c_i}$
    $a_{length(A)+1}.CONCRETE = c_i$
    $A \leftarrow A \cup a_{length(A)+1}$
  end if
end for
return A

Suppose an organization with the set of permissions represented in Table II. The corresponding set of concrete entities contains three subjects (i.e., s_1, s_2, s_3); and the set of permissions with respect to each concrete entity contains the following permissions[1]:

$$s_1 :: \{\alpha_1 o_1, \alpha_1 o_2, \alpha_1 o_3, \alpha_2 o_1, \alpha_3 o_1\}$$
$$s_2 :: \{\alpha_1 o_1, \alpha_1 o_2, \alpha_2 o_2, \alpha_2 o_3, \alpha_3 o_2, \alpha_3 o_3\}$$
$$s_3 :: \{\alpha_1 o_1, \alpha_1 o_2, \alpha_1 o_3, \alpha_2 o_1, \alpha_3 o_1\}$$

Notice that subjects s_1 and s_3 share exactly the same permissions. Therefore, by applying Algorithm `GetAbstractEntities` with the set of subjects and their permissions as input parameters, we obtain as output a set containing two roles, i.e., one role grouping the permissions associated to subjects s_1 and s_3, and another role containing the permissions associated to subject s_2.

Suppose now the set of permissions in Table II with respect to actions α_1, α_2, and α_3:

[1]We omitted the decision from the permissions since all permissions cause the same decision in Table II. This should make the example easier to read.

$\alpha_1 :: \{s_1o_1, s_1o_2, s_1o_3, s_2o_1, s_2o_2, s_3o_1, s_3o_2, s_3o_3\}$
$\alpha_2 :: \{s_1o_1, s_2o_2, s_2o_3, s_3o_1\}$
$\alpha_3 :: \{s_1o_1, s_2o_2, s_2o_3, s_3o_1\}$

Actions α_2 and α_3 share exactly the same permissions. Therefore, by applying Algorithm `GetAbstractEntities` with the aforementioned set of actions and permissions, we obtain as output a set containing two activities: one activity grouping the permissions associated to action α_1, and another activity containing the permissions associated to actions α_2 and α_3.

Similarly, from the set of permissions in Table II containing rules with respect to objects o_1, o_2, and o_3, we have the following permissions:

$o_1 :: \{s_1\alpha_1, s_1\alpha_2, s_1\alpha_3, s_2\alpha_1, s_3\alpha_1, s_3\alpha_2, s_3\alpha_3\}$
$o_2 :: \{s_1\alpha_1, s_2\alpha_1, s_2\alpha_2, s_2\alpha_3, s_3\alpha_1\}$
$o_3 :: \{s_1\alpha_1, s_2\alpha_2, s_2\alpha_3, s_3\alpha_1\}$

We can see that none of the objects share permissions. Therefore, by applying Algorithm `GetAbstractEntities` with the aforementioned set of objects and permissions, we obtain as output a set containing three views, i.e., one view per object.

If we group now the inferred abstract entities (roles, activities, and views) as follows:

$R = \{r_1 = \{s_1, s_3\}, r_2 = \{s_2\}\}$
$A = \{a_1 = \{\alpha_1\}, a_2 = \{\alpha_2, \alpha_3\}\}$
$V = \{v_1 = \{o_1\}, v_2 = \{o_2\}, v_3 = \{o_3\}\}$

We can finally derive the following abstract authorizations (displayed as triplets):
(r_1, a_1, v_1), (r_1, a_1, v_2), (r_1, a_2, v_1), (r_2, a_1, v_1), (r_2, a_1, v_2), (r_2, a_2, v_2), (r_1, a_1, v_3), (r_2, a_2, v_3)

F. Complexity

With regard to the complexity of Algorithm `GetAbstractEntities`, Theorems 1 and 2 provide its space and time consumption boundaries. Figure 6 depicts some simulated results, based on [24], [27], to complement the complexity analysis.

Theorem 1 *Let PER be the set of permissions associated to the security mechanism. Let n be the cardinality of PER. Then, the space consumption complexity of Algorithm `GetAbstractEntities` is of $\mathrm{O}(n)$.*

Proof If we assume the worst case scenario, in which none of the concrete entities in PER share common permissions, then the output set A returned by Algorithm `GetAbstractEntities` is a set of abstract entities whose cardinality equals the number of concrete entities. Same applies for the auxiliary set $PERMS_{c_i}$. At the same time, in such a worst case scenario, each permission in PER holds a different set of concrete entities, i.e., the cardinality of C equals the cardinality of PER. Therefore, let n be the cardinality of PER, the space consumption complexity of Algorithm `GetAbstractEntities` is upper bounded by a linear combination of n. □

Theorem 2 *Let PER be the set of permissions associated to the security mechanism. Let n be the cardinality of PER. Then, the time consumption complexity of Algorithm `GetAbstractEntities` is of $\mathrm{O}(n^2)$.*

Proof The time complexity of Algorithm `GetAbstractEntities` is bounded by the nested iterations associated to Lines 6, 8, and 13 of the algorithm. This corresponds to an exhaustive search of concrete entities in C sharing permissions in PER. Let n be the length of set PER, and let m be the length of set C, then it is straighforward that the time complexity of Algorithm `GetAbstractEntities` is upper bounded by the following expression:

$$(m \times n) + \frac{(m-1)(m)}{2}$$

Assuming the worst case scenario, in which m equals n, i.e., rules in PER are completely disjoint, and holding independent concrete entities per rule, then the above expression can be simplified as follows:

$$n^2 + \frac{(n-1)(n)}{2}$$

Therefore, the time consumption complexity of Algorithm `GetAbstractEntities` is upper bounded by a linear combination of n^2. □

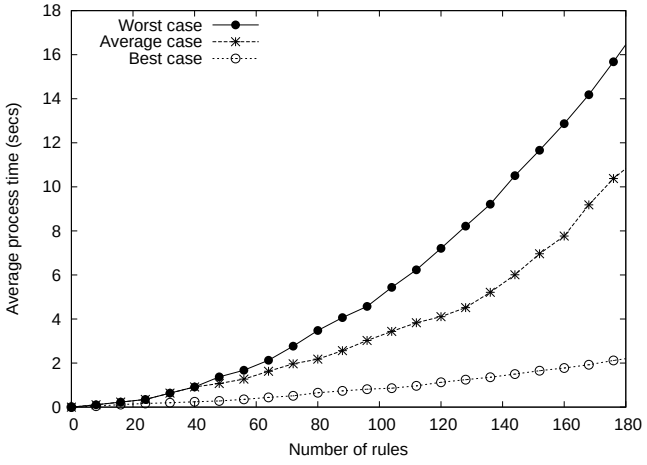

Fig. 6. Processing time evaluation of executing Algorithm `GetAbstractEntities` upon three different rule set classes. Best case assumes rule sets containing a ratio one to ten rules per triplets of concrete entities, i.e., one triplet of concrete entities for every ten rules. Average case assumes rule sets containing a ratio five to ten rules per triplets of concrete entities. Worst case assumes rule sets containing a ratio one to one rules per triplets of concrete entities, i.e., rule sets that contain as much concrete entities as rules.

G. Beyond Traditional Mining: Semantic Enrichment for OrBAC Entities

Traditional role mining approaches often fail to generate meaningful roles. Since mined roles lack semantic data, administrators refuse to deploy roles that they can not understand [13]. So far, the process has outputted *unlabeled* natural roles, i.e., roles that faithfully reflect the PEP policy but which are not named. To further help administrators adopt our proposal, we can leverage the knowledge we have of the processed PEPs to generate additional data that will semantically enrich the discovered results, both concrete and abstract entities. This knowledge can be expressed in the form of heuristics that will allow the process to infer semantic attributes for the mined entities. Heuristics to generate specific semantic information can be developed at the level of each PEP or between several PEPs. Additionally, external knowledge resources can be used to spread information over the entities discovered during the process (subjects, actions, objects, roles, activities, views, contexts, authorizations, obligations).

As an example, let us consider a setting with two PEPs, a firewall and an enterprise directory (LDAP) covering the same domain. For each PEP, the administrator has to select one model among many. These models will be applied to the parsing of the PEP configurations into concrete entities. Once the knowledge is structured, we can derive more knowledge by developing PEP-specific heuristics. Regarding the firewall, we can interpretate information found in configuration files about the existing sub-networks or VLANs. Also, it is commonly known that IP ranges starting in $10.*$ or $192.168.*$ are reserved for private use, thus indicating local networks activities. A subject whose IP address falls into an identified network can then be characterized. By correlating this information with the enterprise directory information, a firewall subject may gain additional semantic attributes. In case the LDAP's subject has a DN (distinguished name) extended with alternative name extensions, a rule can join the LDAP's subject to the firewall's subject by their common IP address. This leads to the fusion of both subjects into a composite subject, and by extension, the fusion of two PEP models. Otherwise, browsing an external resource such as a DHCP configuration may characterize the firewall's subject IP address with organizational information such as the company department to which the IP addess has been allocated. This can be further semantically qualified by adjoining LDAP's OU (organizational unit) attribute information and its inherited attributes' information.

Although designing heuristics to feed this process is specific to each PEP class, this process of enrichment and correlation can be generalized at several levels: attribute level, concrete level and abstract level. Such process has the potential to enhance natural roles we mined during the attribute-based mining process, and should be integrated to it.

VI. Conclusion and Future Works

Almost all the known existing access control models adopt the top-down approach for the designing and the management of security policies. However, the top-down approach is costly and time-consuming process. Additionally, adopting such approach by the access control models create an huge obstacle for organizations to migrate their existing security policies towards one of the access control models.

We propose to take the opposite direction of common access control models by proposing a attribute based mining process. The process has the advantage that it makes no assumption on the status of policies within the organizations and is therefore adaptive to any access control model. The main idea of the process is to handle PEPs by classes so that the same translation strategies can be applied to each member of the class. We start by parsing the existing configurations rules, processing them and then deducing the concrete entities. We have lastly designed an algorithm to perform a concrete-to-abstract deduction by browsing the permissions of the concrete entities. The complexity computation shows that this algorithm has a linear space complexity and a quadratic time complexity.

Our attribute mining process produces only *unlabeled* abstract and concrete entities. The administrators of organizations could be reluctant to adopt such process because they are not able to understand the obtained results. We have proposed to handle this issue by proposing heuristics that can help to conclude semantic data and associate them with the concrete and abstract entities resulted from the process. The heuristics depend on the type of PEPs and can be developed at the level of each PEP or between several PEPs.

Our approach outputs natural roles, activities and views that reflect the current status of the security policies. Thus, we are not concerned by computing the semantic quality of the mined entities. However, the abstract entities rely strongly on the attribute to concrete entities model chosen by the administrator. Therefore, we plan to develop metrics to assess the best model, i.e., how meaningful are the mapping of rule attributes to concrete entities.

Finally, we plan to implement our approach as a framework of tools that will carry out the whole process from parsing configuration rules to building PEP class dictionaries to mining entities.

References

[1] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn, "Assessment of access control systems," National Institute of Standards and Technology, Gaithersburg, MD, NIST Interagency Report 7316, September 2006.

[2] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[3] A. Abou-El-Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin, "Organization Based Access Control," in *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003, pp. 120–131, lake Come, Italy.

[4] E. Yuan and J. Tong, "Attributed Based Access Control (ABAC) for Web Services," in *Proceedings of the IEEE International Conference on Web Services*, ser. ICWS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 561–569. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2005.25

[5] E. J. Coyne, "Role engineering," in *ACM Workshop on Role-Based Access Control (RBAC)*, 1995.

[6] R. W. H. Roeckle, G. Schimpf, "Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization," in *ACM Workshop on Role-Based Access Control (RBAC)*, 2000.

[7] S. C. D. Shin, G.J. Ahn and S. Jin, "On modeling system-centric information for role engineering," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2003.

[8] A. O. M.P. Gallagher and B. Kropp, "The economic impact of role-based access control," National Institute of Standards and Technology, Gaithersburg, MD, NIST Interagency Report Technical Report Planning Report 02-1, 2002.

[9] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining - revealing business roles for security administration using data mining technology," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2003, p. 179186.

[10] J. Schlegelmilch and U. Steffens, "Role mining with ORCA," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2005, p. 168176.

[11] J. Vaidya, V. Atluri, and J. Warner, "Roleminer: Mining roles using subset enumeration," in *ACM Conference on Computer and Communications Security (CCS)*, 2006, p. 144153.

[12] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: Finding a minimal descriptive set of roles," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2007.

[13] A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan, "Fast exact and heuristic methods for role minimization problems," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.

[14] A. Colantonio, R. D. Pietro, and N. V. Verde, "A business-driven decomposition methodology for role mining," in *Computers & Security*, 2012.

[15] C. S. Jordan, *Guide to Understanding Discretionary Access Control in Trusted Systems*. DIANE Publishing, 1987.

[16] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," DTIC Document, Tech. Rep., 1976.

[17] B. W. Lampson, "Protection," in *5th Princeton Symposium on Information Sciences and Systems*, March 1971, pp. 437–443.

[18] F. Cuppens and N. Cuppens-Boulahia, "Modeling Contextual Security Policies," *International Journal of Information Security*, vol. 7, no. 4, pp. 285–305, 2008.

[19] M. Ben Ghorbel, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula, "A delegation model for extended RBAC," *International journal of information security*, May 2010.

[20] F. Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel, "High level conflict management strategies in advanced access control models," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 186, pp. 3–26, July 2007.

[21] Y. Elrakaiby, F. Cuppens, and N. Cuppens-Boulahia, "Formal enforcement and management of obligation policies," *Data Knowl. Eng.*, vol. 71, no. 1, pp. 127–147, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2011.09.001

[22] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, "AAA Authorization Framework," RFC 2904 (Informational), Internet Engineering Task Force, Aug. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2904.txt

[23] J. Garcia-Alfaro, N. Boulahia-Cuppens, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *International Journal of Information Security*, vol. 7, no. 2, pp. 103–122, 2008.

[24] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, "MIRAGE: a management tool for the analysis and deployment of network security policies," in *3rd International Workshop on Autonomous and Spontaneous Security (SETOP 2010)*. Springer, 2011, pp. 203–215.

[25] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot, "Management of stateful firewall misconfiguration," *Computers & Security*, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404813000217

[26] J. Park and R. Sandhu, "The $UCON_{ABC}$ Usage Control Model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, Feb. 2004. [Online]. Available: http://doi.acm.org/10.1145/984334.984339

[27] "MIRAGE, An Audit Tool for the Analysis of Security Policies." [Online]. Available: http://crimplatinum.org/mirage/