

Analysis of Privacy and Security Policies

ELISA BERTINO³, CAROLYN BRODIE², SERAPHIN CALO²,
LORRIE CRANOR¹, CLARE-MARIE KARAT², JOHN KARAT²,
NINGHUI LI³, DAN LIN³, JORGE LOBO²,
QUN NI³, PRATHIMA RAO³ AND XIPING WANG²

Carnegie Mellon University¹, IBM, T.J. Watson Research Center², Purdue University³

Abstract

Policy analysis techniques have usually been developed independently of applications or they have been tailored to policies with specific purposes, e.g. they have been used to analyze access control policies, system management policies or privacy policies. There are analysis techniques to detect redundancy and incompleteness of policies. There are also techniques to detect modality conflicts such as obligations that cannot be fulfilled because of the lack of authorizations, and techniques to detect circular dependencies of obligations. The intent of this paper is to present a survey of these and other analysis techniques that have been developed independently for the analysis of security and privacy policies, and to show how these techniques can be harmonized to analyze the interactions between these two types of policies. Working with health care examples we will show how these different techniques can be applied to improve the quality of policies.

1 Introduction

The authoring and implementation of security and privacy policies is usually a distributed process. In a medium size organization there might be global privacy and security policies that apply across the organization while individual departments may have their own policies. Each department implementing their policies might also be responsible for the local implementation of the organization-wide policies. Departments interact and provide services to each other or combine services to serve other departments or entities external to the organization. All these interactions between different parts of an organization create an incredible challenge for the management of security and privacy policies. Maintaining consistency across different parts of the organization which can all be creating and modifying policies necessitates the development of tools and methodologies to manage the policies life cycle. In a companion paper [18] we describe a general framework for security and policy management. In this paper we will look in detail at a fundamental component of the framework: policy analysis.

Policy analysis techniques have usually been developed independently of applications or they have been tailored to policies with specific purposes, e.g. they have been used to analyze access control policies, system management policies or privacy policies. There are analysis techniques to detect redundancy and incompleteness of policies. There are also techniques to detect modality conflicts such as obligations that cannot be fulfilled because of the lack of authorizations, and techniques to detect circular dependencies of obligations. The intent of this paper is to present a survey of these and other analysis techniques that have been developed independently for the analysis of security and privacy policies and show how these techniques can be harmonized to analyze the interactions between these two types of policies. Working with health care examples we will show how these different techniques can be applied to improve the quality of policies.

The meaning of security or privacy policies can significantly differ from situation to situation, and the languages in which these policies are expressed can also vary significantly. Our interest in this paper is to look at security and privacy policies as they are applied and implemented in IT systems. In particular, we will focus on access control policies and obligations. An access control policy determines who is authorized to have access to what data or resources and under what circumstances. These are examples of access control policies:

- *Security*: Radiologists can access image processing applications and tools.
- *Privacy*: Radiologists can collect patient medical information for the purpose of treatment.

There are many classes of obligations, but in this paper we will consider obligations that are needed to get or maintain access to data or resources. In many situations obligations can be interpreted as conditions that must be satisfied to enable the access. But in contrast to a regular condition that merely checks the state of the environment where the policy is evaluated (e.g., the time of the day), the obligation involves an action that changes the environment. Thus, we will consider an obligation to be an action or process that must be performed at certain times under certain conditions in order to gain or maintain access to data or resources. These are examples of access control policies with obligations:

- *Security*: Hospital staff must change passwords every 90 days to maintain access to the computer system.
- *Privacy*: Doctors can collect medical information of a minor for the purpose of treatment after obtaining a written authorization from his/her legal guardian.

In the security example to get access to computer systems the obligation is to change the password every 90 days. In the privacy example, to collect medical information the obligation is to obtain the guardian authorization.

Before we start our discussion on analysis we will introduce in Section 2 an elementary notion of access control policy and build on this notion to describe the different analysis techniques. We will discuss in Section 3 basic analysis techniques for access control policies. We will talk about how to discover ineffective policies, potential inconsistencies and redundancies. We then move to obligations in Section 4. The first part of this section extends the notion of access control from Section 2 to incorporate obligations. In the second part we discuss analysis techniques for obligations. In Section 5 we take a different look at policy analysis. In this section we explore two methodologies for finding similarities among policies. The first method takes a broad view of policies and tries to cluster policies based on structural similarities. The second method takes a narrow view and can point out similarities and differences based on the specific input/output behavior of the policies. In Section 6 we look at the problem of policy distribution. This section discusses a general architecture in which an access control request is decided by more than a single party. It describes analysis methodologies to decompose policies for distributed evaluation. In the last section, Section 7, we have some concluding remarks including some references to policy analyses not covered in this paper.

2 Access control policies

We first start by assuming that there is a generic computer system that possesses data or resources that need to be protected from unauthorized accesses. Policies are defined to apply to this system.

Definition 1 An *access control policy (rule)* is a tuple of the form

$$(Subjects, Action, Resources, Purpose, Condition)$$

The *subjects* term identifies a subset of all the entities (e.g. an individual or process) that can potentially invoke an action in the system. The *action* is any operation (e.g. deleting a file) that can be applied to a resource in the system. The *resources* term identifies a subset of all entities for which access can be restricted (e.g. data, or computer resources like Web-servers or database servers). The *purpose* is an optional argument selected from a pre-defined set of purposes that subjects may have for executing an action. If the purpose is not used in the rule it will be replaced by *null*. The *condition* is a boolean expression (i.e. a predicate) that might involve the *attributes* of the other four arguments plus system *attributes*. We call the set of available attributes the *context* of evaluation.

Subjects, action, and resources are classical components of access control policies that specify who can access what and invoke which operation. Conditions are introduced to realize more fine-grained policies. The condition checks for properties of the context with no intended side effects. For example, it can further constrain the scope of the target of an access control policy. If a side effect is required we will consider that an obligation and special considerations must be made. We will discuss obligations in detail in Section 4. The purpose for which an access request is undertaken is indispensable to privacy policies.

We can provide two different readings to a rule (S, A, R, P, C) :

- Any member of the set of Subjects S is authorized to execute action A on any member of the set of resources R for the purpose P if the condition C holds.
- No member of the set of Subjects S is authorized to execute action A on any member of the set of resources R for the purpose P if the condition C holds.

The first reading identifies the rule as a positive authorization while the second identifies it as a negative authorization. To disambiguate the reading we will write $+(S, A, R, P, C)$ for positive authorizations and $-(S, A, R, P, C)$ for a negative authorization when needed.

The two examples in the introduction are both positive authorizations. The security policy example includes two rules. In the first rule S = the radiologists, A = access, R = the image applications, there is no purpose and the condition is the boolean constant *true*. The second rule only changes in the set of resources involved, R = image tools. The privacy example has S = the radiologists, A = collect, R = patient medical information, P = treatment and the condition is the boolean constant *true*.

Typical environmental attributes that will be used in the condition are time and, in mobile systems, location. Other attributes could refer to properties of the subjects and the resources. For example, if the resource is patient data an attribute could be the owner of the data and properties of the owner such as date of birth and marital status.

There are many access control systems that only handle positive authorizations. In these systems a *negative closed world* is assumed and every action that is not explicitly authorized is denied. Although it is not common practice, a system can assume a *positive closed world* and allow only negative authorizations and every action that is not explicitly denied is authorized. Many systems separate the policy evaluation from the final policy decision. In such a case one can mix positive and negative authorizations, so that the evaluation of a set of policy rules on an action can result in permit, deny, both or neither and an independent strategy is used to pick the final result. Writing policies using a pure closed world assumption is complex and not necessarily more secure. The prototypical example of a closed world system is SELinux [25] which uses a negative closed world assumption. In a functional SELinux system the number of policy rules runs over the tens of thousands with no systematic mechanism that can show the soundness (only allowed accesses are permitted) or completeness (all the allowed accesses are permitted) of the policies. The perceived advantage of closed word systems is twofold. One is that in a closed world it would be easier to maintain the principle of least privilege and two, that policy administrators cannot write inconsistent rules, i.e., a set of rules in which an action is allowed by a positive authorization and simultaneously denied by a negative authorization.

Different policy analysis techniques have been developed to ameliorate policy errors (i.e., incompleteness and inconsistency). We will discuss some of them in the following sections.

3 Basic policy analysis model

We will follow the methodology of the EXAM system [22] to characterize the type of analyses we would like to do to access control policies. In EXAM, analysis queries are classified in three categories: *policy metadata queries*, *policy content queries* and *policy effect queries*.

A *policy metadata query* would be a query concerning metadata about the policies. Information such as creation and modification dates, authors and location of the policy will be obtained by making metadata queries. Other metadata retrieved using metadata queries are the available subjects, actions and resources in the system, the list of purposes and other system attributes that can be utilized in the condition of a policy rule.

A *policy content query* directly examines the content of policies. Queries such as how many positive or negative authorizations are in the system, and queries for selecting rules that mention a certain attribute value, or subject or resource can be considered policy content queries. These two categories are the traditional system administration types of queries. Although an access control policy management system must support some subset of these types of queries and they are not necessarily trivial to implement we will not discuss them more in this paper and instead will discuss the *policy effect queries* in detail.¹

¹Correctly tracking changes for auditing queries may have legal implications and it is an area of active research. The appropriate

A *policy effect query* is a query about the outcomes policy rules may produce and the interaction of among policy rules due to their outcomes. In EXAM, policy effect queries are subdivided into two classes: queries about single policy rules and queries about multiple policy rules. Next we will look at these types of queries in detail.

3.1 Queries on single policy rules

There is only one type of query that we will describe in this section: *property verification queries*. An example of a property verification is: given a set of access control requests, is there at least one of these requests for which the policy will make a decision? Other queries may want to know if a policy applies to all the requests or none.

In order to get a better grasp on how difficult answering these queries are we need a more precise specification of access control requests and policy rules. We assume we have a set of objects \mathcal{O} partitioned into types. The types will be system dependent but typically there will be simple types like booleans, integers and characters and more complex types like strings, sets and trees. We assume we have available the standard arithmetic and boolean operations that we can apply to the simple objects. An object o of a complex type may have a set of associated attributes $\{a_1, \dots, a_n\}$ and values of these attributes are also objects of a fixed type. We will use the standard dot notation, $o.a_i$ to refer to the value of attribute a_i of the object o .

We identify four special sets of objects, the set of subjects \mathcal{S} , the set of actions \mathcal{A} , the set of resources \mathcal{R} and the set of purposes \mathcal{P} . All four sets are finite. The intersection of \mathcal{S} and \mathcal{R} is not necessarily empty, i.e. there might be objects that can be a subject and a resource.

We also assume that the system provides a set \mathcal{F}_b of boolean functions such that a function $f_b \in \mathcal{F}_b$ can take one or more objects as parameters (i.e. $f_b(o_1, \dots, o_n)$), and return either *true* or *false*. In addition *true* and *false* are members of \mathcal{F}_b . They can be interpreted as constant functions (the only functions with zero parameters) with the standard meaning.

A request is a triple (s, a, r, p) , where $s \in \mathcal{S}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$ and $p \in \mathcal{P} \cup \{null\}$. In a policy rule (S, A, R, P, C) , $S \subseteq \mathcal{S}$, $A \in \mathcal{A}$, $R \subseteq \mathcal{R}$, $P \in \mathcal{P} \cup \{null\}$ and C is a boolean expression that combines instances of the system pre-defined boolean functions using the boolean operations *and* (denoted by \wedge), *or* (denoted by \vee) and *not* (denoted by \neg or *not*). In C some of the parameters in the boolean functions can be S , A , R or P .

Given a condition C of a policy rule P_r and a request (s, a, r, p) , we will denote by $C[s, a, r, p]$ the condition C' that results from replacing every occurrence of S, A, R, P by s, a, r, p respectively in C .

The *evaluation of a positive authorization rule* $P_r = +(S, A, R, P, C)$ on a request (s, a, r, p) , denoted by $eval(P_r, (s, a, r, p))$, will return *permit* if $s \in S$, $a = A$, $r \in R$, P is *null* or equal to p and the evaluation of the condition $C[s, a, r, p]$ is true; otherwise it will return *non_applicable*.

Similarly, the *evaluation of a negative authorization rule* $P_r = -(S, A, R, P, C)$ on a request (s, a, r, p) , denoted by $eval(P_r, (s, a, r, p))$, will return *deny* if $s \in S$, $a = A$, $r \in R$, P is *null* or equal to p and the evaluation of the condition $C[s, a, r, p]$ is true; otherwise it will return *non_applicable*.

A *property verification query* for a policy rule P_r is a tuple of the form (C_{P_r}, O_{P_r}, Q) , where C_P is a query condition where the subject set S , the action A , the resource set R and the purpose P of P_r can occur; O_P is one of *permit*, *deny* or *non_applicable* and Q is one of *none*, *some* or *all*.

A property verification query (C_{P_r}, O_{P_r}, Q) for a policy rule P_r is true if one of the following conditions holds:

1. Q is *none* and there does not exist a request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_r, (s, a, r, p))$ returns O_{P_r}
2. Q is *some* and there is at least one request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_r, (s, a, r, p))$ returns O_{P_r}
3. Q is *all* and for every request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_r, (s, a, r, p))$ returns O_{P_r}

We will refer to these queries as *none* queries, *some* queries, and *all* queries respectively.

The most basic property that we can test using a property verification query is whether a policy rule P_r applies to any input at all, i.e., does this policy affect any subject? If P_r is a positive authorization $+(S, A, R, P, C)$ we

query language and the appropriate way to display answers to queries is also an active human computer interaction research area [19, 17, 29].

can get this answer by asking the query (*true, permit, some*). If the answer to this query is negative, a negative closed world assumption is used and this is the only rule in the policy every request by any subject against any target will be denied. A similar query can be used for negative authorizations having the opposite effect in a positive closed world, i.e. every access will be permitted. We will call this class of policies *ineffective*.

It is unlikely that a security or privacy policy is intentionally written to behave this way and the analysis should try to provide an explanation to identify the ineffectiveness. This can happen for three reasons: (i) the set S of subjects is empty; (ii) the set R of resources is empty or (iii) these sets are not empty but there is no request (s, a, r, p) that satisfies the condition $C[s, a, r, p]$. Hence to answer this type of query we need to detect (i), (ii) and (iii).

We have not specified any properties of the arguments S and R in a policy rule. We only know that they must be finite because they are subsets of \mathcal{S} and \mathcal{R} . In practice, many times subjects are organized in types or roles and role hierarchies. Similarly resources are organized in types and type hierarchies. Membership verification in these sets can be done in linear time with respect to the size of the set and in most cases can be done in sublinear or constant time. On the other hand, if we allow arbitrary boolean expressions in the condition, even if the evaluation of the system pre-defined boolean functions is fast (i.e. constant time), detecting (iii) is an NP-hard problem since the problem boils down to a satisfiability problem. In practice, however, the number of object attributes involved in a condition and the condition itself are small and testing satisfiability is possible. Next we need to consider the boolean functions in \mathcal{F}_b . Agrawal et al. [2] identify four categories that frequently occur in policies:

- **Category 1:** One variable equality constraints.

$x = c$, where x is a variable and c is a constant.

- **Category 2:** One variable inequality constraints.

$x \triangleright c$, where x is a variable, c is a constant, and $\triangleright \in \{<, \leq, >, \geq\}$.

- **Category 3:** Regular expression constraints.

$s \in L(r)$ or $s \notin L(r)$, where s is a string variable, and $L(r)$ is the language generated by regular expression r .

- **Category 4:** Real valued linear constraints.

$\sum_{i=1}^n a_i x_i \triangleright c$, where x_i is variable, a_i, c_i are constants, and $\triangleright \in \{=, <, \leq, >, \geq\}$.

Agrawal et al. [2] discuss polynomial time algorithms and a system implementation to test for satisfiability of conjunctions of formulas that combine the four categories. The implementation was done to do policy analysis in a policy management system called PMAC [1]. Bandara et al. [5] has an implementation that covers categories 1, 2 and 3 for the Ponder language [10]. Brodie et al. [7] has analysis methods at the structural level, i.e., they parse policies written in structure natural language and verify that all the components of a policy are present and they are written in the appropriate form.

Note that handling the query condition in a property verification query (C_{P_r}, O_{P_r}, Q) is as simple as adding the query condition C_{P_r} to the condition of the policy.

For example, consider a hospital's access control policy P regarding a patient's medical records. Policy P allows access to medical records if the patient has consented and if the subject who needs access is either a nurse or a primary physician. P can be denoted by:

$P1 : (*, *, MedicalRecords, Consent=yes \wedge (Role = Nurse \vee Role = PrimaryPhysician), \emptyset)$

If a patient would like to ensure that under no circumstances is access to his medical records allowed without his consent, he can use the following property verification query $Q1$ on P :

$Q1 : (r = MedicalRecords \wedge Consent \neq yes, permit, none)$

Similarly, if he would like to know if his primary physician has access to his medical records, he can use query $Q2$ on P :

$Q2 : (r = MedicalRecords \wedge Role = PrimaryPhysician, permit, some)$

3.2 Queries on sets of policy rules

In contrast to having a single rule a policy set can mix positive and negative authorizations. Thus, several rules can apply to a single request returning opposite results, i.e. some permit the request and some deny it. Different implementations handle this in different ways. Some systems assign priorities to the rules like in PMAC [1], others give priorities to the results [8], others mix the two [26] and yet others consider it an inconsistency [5]. The queries described in this section are intended to help policy administrators discover the interaction among rules so that they can address them appropriately according to the specific methods available in the system that they use.

Queries over policy sets can be defined either to find commonalities or differences among the sets. We extend the syntax of property verification queries to handle these two cases. Queries will be of the form $(C_{P_s}, (O_{P_1}, O_{P_2}), Q)$ where both O_{P_1} and O_{P_2} are taken from the set $\{permit, deny, non_applicable\}$. The answer to this type of query when we are comparing a pair of policy sets (P_1, P_2) is yes or true if one of the following conditions holds:

1. Q is *none* and there does not exist a request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_1, (s, a, r, p)) = O_{P_1}$, $eval(P_2, (s, a, r, p)) = O_{P_2}$.
2. Q is *some* and there is at least one request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_1, (s, a, r, p)) = O_{P_1}$ and $eval(P_2, (s, a, r, p)) = O_{P_2}$.
3. Q is *all* and for every request (s, a, r, p) such that $C_P[s, a, r, p]$ is true and $eval(P_1, (s, a, r, p)) = O_{P_1}$ and $eval(P_2, (s, a, r, p)) = O_{P_2}$.

Similar to the single rule queries, we will refer to these queries as *none* queries, *some* queries, and *all* queries respectively.

If in the query $O_{P_1} = O_{P_2}$ we call the query *common property* query. If $O_{P_1} \neq O_{P_2}$ we call the query *discrimination* query. If the answer to a discrimination query is yes, this implies there are positive and negative authorization rules that are in *conflict*, and either a conflict resolution strategy needs to be applied if the sets are mixed.

These queries could be very helpful during the harmonization of security and privacy policies. Assume, for example, that P_1 represents a privacy policy and P_2 represents a security policy and we want to verify that the access that will be permitted by the privacy policy is no larger than the permitted access allowed by the security policy. Then, we just need to ask the query $(true, (permit, deny), none)$. This is assuming that non applicable requests in the security policy are "don't-care" requests; otherwise we also need to check that the query $(true, (permit, non_applicable), none)$ is also true. We can also check that a policy is contained in a second policy by checking that both $(true, (permit, deny), none)$ and $(true, (deny, permit), none)$ hold, and to show that the containment is proper at least one of $(true, (non_applicable, permit), some)$ and $(true, (non_applicable, deny), some)$ must also hold. The policy containment relation is also known as *dominance*. The policy that contains the second policy is said to dominate the second policy since joining the second policy to the set makes no difference [2] to the access control.

To see how these queries can be implemented we just observe that to check if there is a request (s, a, r, p) for which the two policy rules $(S_1, A_1, R_1, P_1, C_1)$ and $(S_2, A_2, R_2, P_2, C_2)$ apply then it must be the case that $s \in S_1 \cap S_2$, $a = A_1 = A_2$, $r \in R_1 \cap R_2$, $p = P_1 = P_2$ and $C_1[s, a, r, p] \wedge C_2[s, a, r, p]$. Again assuming that intersection of sets of subjects and sets of resources are easy to compute the hard question is the satisfiability of $C_1[s, a, r, p] \wedge C_2[s, a, r, p]$. The same methods used for property verification queries can be used here. This answers the *some* query. The *none* query is answered positively by trying to answer the *some* query and failing; otherwise it is false. The *all* query can be answered using a *none* query by observing that if two policies apply to the same set of requests then if we take the "complement" of one of the policies the resulting pair of policies must have no intersection. Hence, to check that $(S_1, A_1, R_1, P_1, C_1)$ and $(S_2, A_2, R_2, P_2, C_2)$ apply to the same set of requests we need to check that there is no request that applies neither to $(S_1 - S_2, A_1, R_1 - R_2, P_1, C_1)$ nor to $(S_1, A_1, R_1, P_1, C_1 \vee not(C_2))$.

The implementation in [2] covers all these case for PMAC in the four categories with polynomial algorithms when the condition has only ands. Halpern and Weissman [14] use a fragment of multi-sorted first order logic

as the language to represent access control policies. They call this fragment Lithium and show that an efficient implementation exists to evaluate policies in this class and treat analysis as logical entailment of formulas. Kolovski et al. [20] formalize XACML policies by using description logic, another subset of first order logic, and then employ logic-based analysis tools for policy analysis. Bandara et al. [5] uses logic programs to represent policies and abduction logic programming techniques to do analysis. The system described in Brodie et al. [7], also does redundancy analysis of policies that have been specified in structured natural language.

4 Obligations

As mentioned in the introduction we will look at obligations as they relate to both security and privacy access control policies. Obligations arise in access control when a subject promises to fulfill certain obligations in order to gain access to some target. In the implementation an obligation can always be associated with an action. Take, for example, the obligation from the introduction: *Hospital staff must change passwords every 90 days to maintain access to the computer system.* Here there is an obligation for any user within a hospital staff role. The user must update his/her password once every 90 days. The obligation should be activated once a user is granted a hospital staff role.

To analyze obligations that apply to access control we need a rational notion to describe them. Before we introduce a more precise definition of obligation we summarize the following features that motivate the definitions below:

- Obligations usually have some specific temporal constraints. Some obligations should be fulfilled before an access is allowed and the result from the obligation fulfillment may affect the decision about the access request. We call these kinds of obligations *pre-obligations*. Other obligations should be fulfilled after the action in the access request is performed. We call these kinds of obligations *post-obligations*. Intuitively, there should be some time interval allocated for each obligation. Otherwise, a policy enforcement engine does not know when it can start evaluating policy conditions in pre-obligations, and subjects in a post-obligation can legally avoid obligations by simply saying “I will do it in the future”. In some cases, temporal constraints require obligations to be fulfilled repeatedly until some condition becomes true.
- A subject’s obligation may result from another subject’s action, i.e., the subject of an obligation may be different from the subject who caused the obligation. For instance, when a doctor discloses medical information of a child to third parties (e.g. another doctor), the third parties may be required to fulfill similar obligations to those of the doctor. In some situations, the subject of an obligation may be the system itself, e.g., logging access history.
- Some obligations may be conditional, that is, a conditional obligation is only required to be fulfilled if some related condition becomes true. For instance, the Children’s Online Privacy Protection Act says that “Before collecting, using or disclosing personal information from a child, an operator must obtain verifiable parental consent from the child’s parent.” Here, the pre-obligation “obtain verifiable parental consent” needs to be conditional because once the consent has been obtained the pre-obligation should not be executed again.

We will closely follow the notation in [27]. There, the notion of temporal constraints is based on a simple notion of time domain lifted from the set of integers, \mathbb{Z} , and the standard order \leq on \mathbb{Z} . For this context, elements of \mathbb{Z} are referred to as a *time instants*. In what follows, given $t, t' \in \mathbb{Z}$, $[t, t']$ denotes $t' - t + 1$ consecutive time instants. The following definition from [27] introduces a terse yet flexible definition of temporal constraints that will let us capture a large class of obligations.

Definition 2 A *temporal constraint* tc is a tuple (t_s, t_e, cnt) , where $t_s, t_e \in \mathbb{Z}$, and $cnt \in \mathbb{N}^*$. tc denotes a sequence of time intervals defined as follows:

- $[t_s, t_e], [t_e + 1, 2t_e - t_s + 1], \dots, [t_s + (cnt - 1)(t_e - t_s + 1), t_e + (cnt - 1)(t_e - t_s + 1)]$ if $t_e \geq t_s \geq 0$;
- $[t_s - (cnt - 1)(t_e - t_s + 1), t_e - (cnt - 1)(t_e - t_s + 1)], \dots, [2t_s - t_e + 1, t_s - 1], [t_s, t_e]$ if $0 \geq t_e \geq t_s$.

For instance, a temporal constraint (3,7,3) represents a sequence of time intervals: [3,7], [8,12], [13,17]. For a time interval $[t_s, t_e]$, the time instants of the time interval are $t_s, t_s + 1, \dots, t_e - 1, t_e$. For example, the time instants in [3,7] are 3, 4, 5, 6, 7. The notation assumes that time is relative to an action request. We identify two special time instants that are related to a predetermined anchor action: 1) a time instant, referred to as *decision time*, that is equal to the time instant at which the decision of granting the permission to execute the action is made; 2) a time instant, referred to as *completion time*, that is the time instant of the completion of the action execution. For a time instant t , if $t > 0$, then t represents the number of time instants after a *completion time*. If $t < 0$, then t represents $|t|$ time instants before a *decision time*. For a temporal constraint (t_s, t_e, cnt) ², if $t_e \leq 0$, the temporal constraint indicates that the obligation under consideration is a *pre-obligation* and should be fulfilled during the time interval between the request time and the decision time. If $t_s \geq 0$, the temporal constraint indicates that the associated obligation is an *post-obligation* that should be fulfilled sometime after completion time. As a special case, $t_e = t_s = 0$ indicates a post-obligation that should be fulfilled right after the action has been executed.

The distinction between the time before granting permission and the time after the execution of the action gives $t = 0$ a different meaning depending on whether t represents a starting time t_s or an ending time t_e . In classical access control policies without obligations, the time at which an action request occurs, the time at which to evaluate the condition of applicable permission assignments, the time at which to authorize the action, and the time when the action is performed, are considered to be the same. In practice however these times are different especially when obligations and temporal constraints are introduced.

Definition 3 An *obligation* is a tuple of the form
 (*Subjects, Action, Resources, Condition, Temporal Constraint*)

The *subjects, action, resources, purpose, and condition* are defined as usual. A temporal constraint is defined according to Definition 2. Although the notion of an obligation is similar to the notion of an access control policy, the semantics is quite different. An access control policy describes that a subject is permitted to perform some action on some resource if some condition is satisfied while an obligation describes that a subject is obliged to perform some action on some resource under some temporal constraints if some condition is satisfied.

Definition 4 An access control policy with obligations is a tuple of the form
 (*Subjects, Action, Resources, Purpose, Condition, Obligations*)

The term *Obligations* denotes a set of obligations in the sense of Definition 3. Obligations associated with an access control policy should not be interpreted as giving blank authorizations to the subjects of the obligations to execute the action imposed by the obligations. Authorization policies should exist; otherwise subjects will not be able to fulfill their obligations. Let's take one of the policies from the introduction:

- *Privacy*: Doctors can collect medical information of a minor for the purpose of treatment after obtaining a written authorization from his/her legal guardian.

<i>Subject:</i>	<i>doctor,</i>
<i>Action:</i>	<i>collect,</i>
<i>Resources:</i>	<i>{medicalinfo(User)},</i>
<i>Purpose:</i>	<i>treatment,</i>
<i>Condition:</i>	<i>writtenAuthorization = yes \wedge age(User)=minor,</i>
<i>Obligations:</i>	{
<i>Subject</i> ₁ :	<i>self,</i>
<i>Action</i> ₁ :	<i>obtain,</i>
<i>Resources</i> ₁ :	<i>{writtenAuthorization, guardian(User)},</i>
<i>Condition</i> ₁ :	<i>writtenAuthorization = na,</i>
<i>Constraint</i> ₁ :	<i>(-7,0,1)</i>
	}

²It is obvious that $t_s \leq t_e$ must hold.

This policy has a single obligation. The obligation has the same subject as the authorization, i.e. the doctor, and it will need to be executed only if the value of the attribute *writtenAuthorization* is not-available (*na*), and it must be executed in last 7 unites of time (these could be days) prior to collecting the medical information of a minor.

While it increases the expressiveness of the notion of access control policies, the introduction of obligations in access control policies inevitably causes new problems because of the interaction between obligations and authorizations, and the interaction between obligations.

4.1 Analysis of obligations

In the proposed obligation notion, the execution of an obligation can trigger the execution of another obligation. A subject that performs an obligation also needs an authorization to execute the action required by the obligation. The access control policy that provides such authorization may require the execution of some other obligations. The new obligations, in turn, may require the execution of more obligations. We refer to such a phenomenon as *cascading* of obligations.

Let $+(s_1, a_1, r_1, p_1, c_1, (s_2, a_2, r_2, c_2, tc_2))$ and $+(s_2, a_2, r_2, p_2, c_2, (s_1, a_1, r_1, c_1, tc_1))$ be two positive access control policies each with an obligation. Assume now subject s_1 wants to perform a_1 on resource r_1 , the execution of actions is the following:

1. Based on the first policy, subject s_2 needs to perform a_2 on resource r_2 to let subject s_1 perform a_1 on resource r_1 .
2. Based on the second policy, subject s_1 needs to perform a_1 on resource r_1 to let subject s_2 perform a_2 on resource r_2 .

As we can see, the interaction between obligations and policies causes an infinite action execution. There are other cases that the interaction between obligations and authorizations. Let $+(s_1, a_1, r_1, p_1, c_1, (s_2, a_2, r_2, c_2, tc_2))$ be a positive access control policy with an obligation. Other cases that required analysis are:

1. $c_1 \wedge c_2$ is not satisfiable. In this case, the obligation can never be fulfilled and thus is ineffective.
2. There is no access control policy that authorizes s_2 to perform a_2 on r_2 .
3. The obligation in $+(s_1, a_1, r_1, p_1, c_1, (s_2, a_2, r_2, c_2, tc_2))$ is a cascading obligation of itself.

For cases 1 and 2, we can apply the basic policy analysis techniques discussed in previous sections. The basic idea to detect case 3 is to trace the cascading obligations of an access control policy to try to find circular dependencies. Algorithm 1 below returns true if a policy p depends on itself.

Algorithm 1 Condition 3 detection

```

1: function C3DETECTION( $p, cb$ )            $\triangleright p$ : the policy to be checked;  $cb$ : the list of policies, initial value is  $p$ 
2:    $pl \leftarrow$  policies authorizing the actions in the obligations in  $p$ 
3:   while  $pl \neq \text{null}$  do
4:      $p' \leftarrow \text{removehead}(pl)$ 
5:     if  $p' \in cb$  then
6:       return true
7:     end if
8:     if C3Detection( $p', \text{addtail}(cb, p')$ ) then
9:       return true
10:    end if
11:  end while
12:  return false
13: end function

```

We can think of Algorithm 1 as traversing a directed graph in which the nodes are the access control policies and there is an arc from a node to another if an action in the obligations of the first node is the action in the authorization of the second node. The algorithm uses depth-first-search to find cycles.

4.2 Dominance analysis in obligations

In XACML, P-RBAC [28], and E-P3P [3], given a request, obligations in all applicable access control policies are returned. Therefore, we can expect that some action requests can lead to a large number of obligations, especially if there are ill-written policies. Therefore reducing the number of obligations to be executed may have significant practical impact. Obviously, the reduction should not decrease the duties required by the original policies, but we can imagine that many of these obligations are similar to each other since they are associated with similar permissions. In [27] it is observed that if the appropriate notion of obligation containment is defined, we may safely remove some obligations. For instance, given two post-obligations, one requiring that a privacy notice be sent to both the child and a parent within one week, and another requiring the same privacy notice be sent to the parent within two weeks, if both of them are in the post-obligation set returned upon a user action request, it is reasonable that the user only needs to fulfill the former obligation because the duty represented in the latter is “dominated” by the former. Thus, we will extend the concept of *dominance* to cover obligations. There are two factors affecting *dominance analysis of obligations*: the temporal constraints and the obligation conditions.

Definition 5 Let $tc_1 = (t_{s1}, t_{e1}, cnt_1)$ and $tc_2 = (t_{s2}, t_{e2}, cnt_2)$, $t_{s1}, t_{e1}, t_{s2}, t_{e2} \in \mathbb{Z}$ and $cnt_1, cnt_2 \in \mathbb{N}^*$, be two temporal constraints. tc_2 covers tc_1 , written as $tc_2 \triangleright tc_1$, if and only if one of the following conditions hold:

- $t_{s1} \geq t_{s2} \geq 0$ and $(t_{e1} - t_{s1}) \geq (t_{e2} - t_{s2})$ and $cnt_2 \geq cnt_1$.
- $t_{e2} \leq t_{e1} \leq 0$ and $(t_{e1} - t_{s1}) \leq (t_{e2} - t_{s2})$ and $cnt_2 \geq cnt_1$.

For instance, given the post-obligation temporal constraints (0, 5, 2) and (0, 4, 3), it is obvious that (0, 4, 3) covers (0, 5, 2) because (0, 4, 3) requires that the fulfillment of a corresponding obligation be started right after an action is performed, and completed in 5 days. Moreover this obligation fulfillment cycle should be repeated 3 times. On the other hand, (0, 5, 2) requires that the fulfillment of a corresponding obligation be completed in 6 days and only to be repeated 2 times. Post-obligations are duties; therefore the smaller the repeating number is, the less strict the temporal constraint is. A stricter post-obligation dominates a less strict post-obligation.

Given the pre-obligation temporal constraints (-5,0,2) and (-3,0,1), (-5,0,2) covers (-3,0,1) because only one chance, one 4-day time period, is given to fulfill a pre-obligation by (-3,0,1) and two chances, two 6-day time periods, are given to fulfill a pre-obligation by (-5,0,2). In order to understand the dominance of a temporal constraint in a pre-obligation, we have to first realize the difference between pre-obligations and post-obligations. Unlike post-obligations, pre-obligations are actions that must be fulfilled before the requested action can be allowed and *the result of the fulfillment can determine whether the requested action is executable or not*³. The smaller the count number in a pre-obligation temporal constraint is, the stricter the temporal constraint is. A less strict pre-obligation dominates a stricter pre-obligation. In other words, the bigger the count number in a pre-obligation temporal constraint is, the more chances are that the condition of the access control policy be satisfied.

Definition 6 Let $ob_1 = (c_1, s_1, a_1, r_1, tc_1)$ and $ob_2 = (c_2, s_2, a_2, r_2, tc_2)$ be obligations. ob_2 dominates ob_1 iff c_1 implies c_2 , $s_1 = s_2$, $a_1 = a_2$, $r_1 = r_2$, and $tc_2 \triangleright tc_1$.

An example of dominance relation between two obligations is the following:

$$\begin{aligned} ob_1 &= (*, \text{lab_service}, \text{send}, \text{privacy_notice}(\text{patient}), (0,182,\infty)) \\ ob_2 &= (*, \text{lab_service}, \text{send}, \text{privacy_notice}(\text{patient}), (0,365,\infty)) \end{aligned}$$

In this example the laboratory in a hospital is required to send privacy notices to their customers periodically. Assume that time is measured in days in the temporal constraints. Since obligation ob_1 requires to be fulfilled every half a year and obligation ob_2 only requires once a year, ob_1 dominates ob_2 .

Based on the definition, the coverage analysis of obligations is straightforward.

³If a pre-obligation has no effect on the condition associated with an access control policy, the pre-obligation should be a post-obligation.

5 Policy similarity

In this section we look at analysis from a different perspective. We are looking now at policies coming from different authors and perhaps different organizations and we would like to discover similarities. The sharing of resources such as services, data and knowledge in collaborative applications often gives rise to scenarios in which the access control policies of collaborating parties must be compared. For example, a party P may decide to release some data to a party P' only if the access control policies of P' are very much the same as its own access control policies. Similar requirement arises in a scenario where several alternative resources and services, each governed by its own independently-administered access control policies, may have to be selected and combined in a complex service. In order to maximize the number of requests that can be satisfied by the complex service, and also satisfy the access control policies of each participating resource and service, one would like to select a combination of resources and services characterized by access control policies that are very much similar.

Policy similarity can be defined as the *characterization of the relationships among the sets of requests respectively authorized by a set of policies*. The goal of policy similarity analysis is to provide such a characterization. The result of a policy similarity analysis between a set of policies can be used to derive for all requests of the form (s, a, r, p) the corresponding effects of the policies in the set and thus to compare the behavior of policies in the set with respect to all possible requests. Such an analysis can be used to answer the *common property* and *discrimination* queries which could be helpful during the harmonization of security and privacy policies. It can also be used for change impact analysis where an administrator may want to verify the effect of changes to current policies [12] or find differences among rules.

We will discuss two types of approaches [24, 22] to determine similarity between policies. One approach uses the notion of a *policy similarity measure* between policies and is a quick but less precise technique based on principles from the information retrieval field [24]. Another approach is precise but computationally intensive and uses Multi-Terminal Binary Decision Diagrams (MTBDD) based techniques [22].

5.1 Broad policy similarities

In this approach, a *similarity score*, a value in the interval $[0, 1]$, is computed between two policies to quantify the degree of similarity between them. Higher(lower) values of similarity between two policies indicate that there are more(less) number of requests for which the policies have the same effect. The similarity computation is simple and quick and is based on techniques from the information retrieval field.

The similarity measure takes into account the various elements of the policies. Given two policies, the algorithm for computing similarity score first groups the same type of elements in the two input policies, and evaluates their similarity. Then the scores obtained for the different elements of the policies are combined according to a weighted (usually linear) combination in order to produce an overall similarity score. Weights are used so that one can emphasize scores obtained due to one or more specific elements when computing the similarity. For example, one may want to find policies most similar with respect to the subjects to which they apply to. Techniques like dictionary lookup and ontology matching can also be incorporated when computing the similarity scores to bridge the semantic gap arising due to the use of different vocabularies in expressing access control policies. Such techniques are particularly useful when harmonizing policies across administrative domains where each domain may use a different vocabulary for names and values appearing in their policies.

Policy similarity measure can be used to compare many classes of policies. Here we briefly show similarity measures can be defined for two very different languages: XACML [26] and P3P [9].

- An XACML rule collects the *(subject, resource, action)* into a single element called a target in addition to having the condition and an effect (i.e. positive or negative authorization). An XACML policy consists of a set of these rules with a “global” target. A policy rule applies if both the global target and the target of the rule matched (i.e. the intersection is not empty). Each of these policy elements is abstracted into a list of attribute (name, value) pairs. A score between two elements is computed by comparing the list of such pairs and accumulating their scores. A score between two pairs is computed by comparing the corresponding name and value components. Different computations are used for comparing values belonging to numerical and categorical attribute types. In particular, for comparing categorical values associated with domain

specific ontologies a score that reflects the distance in the hierarchy between two concepts is used. For comparing numerical values a score that reflects closeness in the values being compared is used.

- A P3P policy is mainly composed of statements that describe the data and category of information collected along with how the information may be used, how the information may be shared and the associated data retention policies. Thus a P3P policy can be abstracted into a list of tuples each containing the data, category, purpose, recipient and retention elements. Each of the elements can contain values belonging to a pre-defined set specified in the P3P specification. A similarity score between two P3P policies is derived by computing the similarity score between pairs of tuples corresponding to the two policies. A score between a pair of tuples is calculated by comparing the individual components of the two tuples.

Although this approach cannot be used to specifically enumerate the precise differences among policies with respect to specific requests it can be used as a quick filter approach to prune dissimilar policies before using more precise similarity analyzers which are computationally intensive. This is particularly helpful when a large number of policies need to be compared. Such similarity measure between policies can also serve as distance functions when performing data mining techniques like clustering on policy sets.

5.2 Narrow policy similarities

The narrow similarity analysis uses a MTBDD [13] representation of a policy. A policy MTBDD is a directed acyclic graph whose internal nodes represent Boolean predicates corresponding to the S, A, R, C, P components of a policy and whose terminals can be one of $\{permit, deny, not_applicable\}$ representing the effect of the policy on requests that satisfy the Boolean predicates along the path from root to the terminals. In effect, the paths represent the rules of a policy. Policy MTBDDs of the individual policies being compared for similarity are combined to derive a single MTBDD in which each terminal corresponds to a n-tuple $\langle e_1, e_2, \dots, e_n \rangle$ where e_i is the effect of policy P_i on all requests that satisfy the Boolean predicates along the path from root to the terminal. By traversing paths leading to specified terminals in the combined MTBDD, one can derive the set of requests which have common or different authorizations in the given set of policies. For example, one can derive all requests permitted by both policies by traversing all paths leading to the terminal $\langle permit, permit \rangle$ in the combined MTBDD. An XACML policy can be transformed to an equivalent Boolean expression which can then be used for MTBDD construction and analysis.

For example, consider a scenario when a patient would like to transfer his medical records from hospital X to hospital Y each of which has its own policy, $P1$ and $P2$ respectively, governing access to a patient's medical record. Policy $P1$ allows access to medical records if the patient has consented and if the subject who needs access is either a nurse or a primary physician. Policy $P2$ allows access to medical records if the patient has consented or a surgery has been scheduled and if the subject is either a surgeon or a nurse. The two policies $P1$ and $P2$ can be stated more succinctly as follows:

$$\begin{aligned}
 P1 &: (*, *, MedicalRecords, Consent=yes \wedge (Role = Nurse \vee Role = PrimaryPhysician), \emptyset) \\
 P2 &: (*, *, MedicalRecords, (Consent = yes \vee Surgery = Scheduled) \wedge (Role = Surgeon \vee Role = Nurse), \emptyset)
 \end{aligned}$$

Before transferring the medical records, the patient would like to ensure that hospital Y's security policy offers similar level of security as that of hospital X. A similarity analysis between $P1$ and $P2$ can be used to ensure such requirement. Figure 1 shows the MTBDDs corresponding to $P1$, $P2$ and their combination. Paths leading to the P-P terminal in the combined MTBDD highlight the similarity between the two policies while the remaining paths show the differences. The patient can use this information before deciding to transfer his medical records.

Two systems use MTBDDs to do similarity analysis. Margrave [12] has been developed for the analysis of role-based access-control policies written in XACML. EXAM [22] incorporates in a single system similarity comparisons based on similarity scores and similarity analysis based on MTBDDs both for XACML policies. Note that this method can be used to answer many of the queries introduced in Section 3.

Margrave implementation handles Boolean expression formulas built using Category 1 type expressions. The EXAM system [22] handles the four classes.

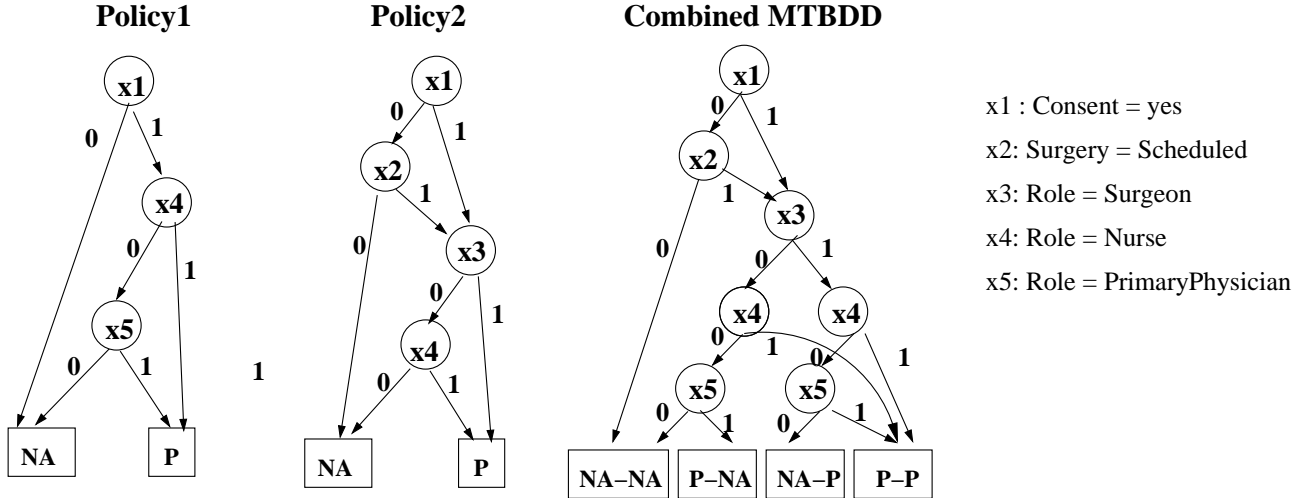


Figure 1: Policy MTBDDs

6 Policy decomposition and distribution

In many situations organizations have several collaborating parties each responsible for managing and protecting resources entrusted to them. Access control decisions thus become a collaborative activity where there is a need to enforce a global policy among collaborating parties without compromising the autonomy or confidentiality requirements of the involved parties. Some of the information may be sensitive and there may not be a unique party having all the necessary information to take an access control decision.

Take a hypothetical example of a hospital with multiple departments including a financial department and a laboratory service department. Each department is responsible for managing access to its resources and each department stores confidential data concerning its specific operations. Consider a (simplified) global policy

(Manager, buy, MedicalSupplyY, -, Balance > 10M \wedge MedicalExamX = "scheduled")

which specifies that a manager of the laboratory service can buy a very expensive medical supply Y that is needed for a medical exam X if there are enough funds in the department budget and the medical exam X has been scheduled for a patient. Suppose that information about exams and patient records is managed by the laboratory service department and the information about the funds is managed by the financial department, and that each department is not able, for confidentiality reasons, to release its own information to others. Here, the access to medical records is protected for privacy considerations to the patients while access to financial information is protected according to security policies of the health organization. Further assume that global policy requires for any requests to be approved by the head of each department and the approval be recorded, through the execution of obligation actions. In such scenario a centralized enforcement of the policy is not feasible. Rather a more suitable approach would consist of "distributing" the policy to the two departments asking each of them to return its own "local" decision. The local decisions are then combined to generate the global decision.

The aim is then to analyze and decompose the policy into sub-policies that can be sent to the different parties to enforce. The basic idea is to analyze the global policy and identify the portions in the global policy that need the information available at each local party. Local policies are constructed and kept in each site. When an access decision is needed the relevant parts of the request are sent to the different parties involved, independent decisions are returned and combined to derive the access control decision as a whole. [23] describes an architecture and process to decompose policies. We will describe both in the following sections.

6.1 Architecture

To support collaborative access control, the system architecture of [23] consists of a central policy enforcement point (PEP) and multiple policy decision points (PDP) (e.g. Figure 2). This differs from the typical access control policy framework where a single PEP and a single PDP are considered. In such a system, a global policy

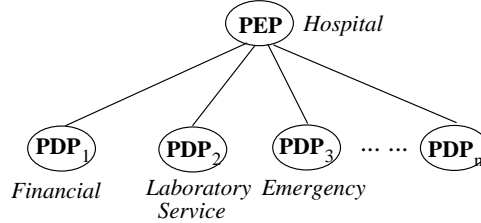


Figure 2: System Architecture

is decomposed into local policies for each PDP according to the availability/sensitivity requirements of each party and user specified constraints, and then local policies are sent to the corresponding PDPs. Once the global policy has been decomposed, it will be encrypted and stored in a secure store. That means that the global policy will no longer be used for the subsequent request evaluation. Instead, only the non sensitive information of each global policy is kept as plain text in a policy table maintained by the central PEP. In the previous example, if the action “Buy” and the resource “MedicalSupplyY” are non sensitive information, they will be kept as plain text. The evaluation tasks of other sensitive information are distributed to local PDPs. For example, the threshold amount to permit the operation without any other requirement might be confidential and kept by the financial department.

When a request is issued, the central PEP first checks whether the request matches the public information stored at the PEP. If there is a match, the central PEP will check the policy table and distribute the request to the corresponding local PDPs. A local policy may contain predicates on both sensitive and non sensitive attributes. For non sensitive attributes, the local PDP looks for the attribute values in the request. For sensitive attributes, the local PDP accesses its local database and resolves the attribute values regardless of whether the request includes such values or not. In some situations, additional information like the requester ID may be needed to help local PDPs resolve the values of the sensitive attributes. The responses of local PDPs are collected and returned to the coordinator where the final decision is made.

The approach assumes that each participating party is trusted and local PDPs will correctly enforce the portions of the global policy that have been sent to them. In the case parties other than the central PEP are compromised by an attacker the information stored at these parties might be leaked while the information stored at other parties is still safe. Similarly, only the request evaluation involving the hacked parties may not be carried out properly. If the central PEP has been attacked, it is hard for the attacker to know the original global policies as they are encrypted. To summarize, the system architecture is designed to prevent the attackers from knowing predicates on sensitive attributes in local policies as well as sensitive information stored at local PDPs.

6.2 Policy analysis and decomposition algorithm

The analysis and decomposition consists of four main steps. First, each rule in the global policy must be translated into a compound boolean expression over policy attributes. A compound boolean expression is composed of atomic boolean expressions combined using the logical operations \vee and \wedge . This expression combines all five components of a policy. The second step is to decide what portion of the policy needs to be assigned to which PDP. Each atomic boolean expression is labeled with the ID of the local PDP that contains the attribute information in the atomic boolean expression, and then atomic boolean expressions are grouped based on their labels. The third step is to generate an actual local policy according to each group of *correlated* atomic boolean expressions. The last step is to compute the effects of original rule from the corresponding local policies.

According to the algorithm, the example policy will be decomposed into two policies P_1 and P_2 as follows.

- P_1 : (*Manager, Buy, MedicalSupplyY,-, MedicalExamX="scheduled"*)
- P_2 : (*Manager, Buy, MedicalSupplyY,-, Balance >10M*)

P_1 will be sent to the laboratory service department and P_2 to the financial department. Only when P_1 and P_2 both yield the permit decision, the global policy will yield permit.

7 Final remarks

There is a large amount of work related to policy analysis. Our goal with this paper has been to show a large spectrum of techniques so that the reader can grasp the many different angles from which the problem of analysis can be attacked. Many of these techniques were originally presented either for access control in a security setting or for privacy. We have not made such a distinction to demonstrate that in practice security and privacy are intermingled and the same approaches can be applied to both types of policies.

There is a general class of analysis problems for which we do not have examples in this paper but it deserves special attention. These are analyses of policies that depend on dynamic changes in the system where the policies are enforced. We have touched slightly on one example of this type of analysis when we introduced the concept of cascading obligations. Algorithm 1 overestimates the dependencies. It might be the case that a dependency path from an obligation o_1 to a cascading obligation o_2 can never be traversed because the execution of an action in the path may invalidate the condition of another obligation later in the path. Hence, although o_1 can be equal to o_2 the system never runs into this circular dependency. There is a complementary problem with obligations and this is that access required for o_2 may depend on the fulfillment of all the obligations between o_1 and o_2 . For example, a hospital may have a policy that states that any case that might lead to an open heart surgery must be reviewed by one of the senior cardiologists. Another policy may say that a physician that is not treating a patient needs to get the patient's consent in order to get access to his or her medical records. The first obligation cannot be fulfilled without the second obligation. These dependencies can be complicated to detect. Irwin et al. [15] defined a secure state as the state where any obligation violation can only be due to the lack of diligence of subjects. A good property for a system to have is to allow a set of policies and execution environments that only move through secure states. Irwin et al. show that this is a computationally hard problem. Some ways of addressing the problem are also discussed in [15]. More about the analysis of obligations can be found in [11] and the references therein.

In the same class we have problems related to dynamic separation of duty and Chinese walls. These are temporal properties that forbid subject access to resources depending on the behavioral history of the subject. A trivial example is that a doctor cannot give a second opinion on a case in which she was involved earlier. These constraints can expand over long transactions and workflows. Analyses of this problem are also computationally hard in general. Some approaches for dealing with these problems can be found in [21] and [30].

In general analysis will be limited if we are not able to incorporate at least partially the behavior of computer systems into the analysis. Consider, for example, a policy which states that the primary health-care provider of a patient is permitted to access the patients medical record, but that nurses in the first year of training are denied such access. These two policy rules are in potential conflict, assuming that subjects are organized in a hierarchical structure and nurses in training are also considered to be primary health-care providers according to the subject hierarchy. In this situation a nurse would both be permitted and denied access to his or her patients records. However, the medical records system may be engineered in such a way as to prevent, not as a consequence of policy rules but as a result of the behavior of the system, nurses in training from becoming primary care providers, and thus the conflict will not arise. It is only by analyzing policies in conjunction with the laws of system evolution that a correct analysis is obtained. Incorporating system dynamics into the analysis of policies is explored in [30] and [4].

Acknowledgments

The authors would like to acknowledge the support of an IBM Open Collaborative Research Award to Elisa Bertino at Purdue University and Lorrie Cranor at CMU which has partly supported the work reported here and

supported the collaboration between teams at IBM Research, CMU and Purdue.

References

- [1] D. Agrawal, K-W. Lee, and J. Lobo. Policy-based management of networked computing systems. *IEEE Communications*, 43(10):69–75, 2005.
- [2] Dakshi Agrawal, James Giles, Kang-Won Lee, and Jorge Lobo. Policy ratification. In *POLICY*, pages 223–232. IEEE Computer Society, 2005.
- [3] Michael Backes, Markus Dürmuth, and Rainer Steinwandt. An algebra for composing enterprise privacy policies. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer, 2004.
- [4] Arosha Bandara, Seraphin Calo, Robert Craven, Jorge Lobo, Emil Lupu, Jiefei Ma, Alessandra Russo, and Morris Sloman. An expressive policy analysis framework with enhanced system dynamicity. Technical Report, Department of Computing, Imperial College London, 2008.
- [5] Arosha K. Bandara, Emil Lupu, and Alessandra Russo. Using event calculus to formalise policy specification and analysis. In *POLICY*, pages 26–35. IEEE Computer Society, 2003.
- [6] Joachim Biskup and Javier Lopez, editors. *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, volume 4734 of *Lecture Notes in Computer Science*. Springer, 2007.
- [7] C. Brodie, D. George, C. Karat, and J. Karat. Methods of usable policy analysis, 2008. Under review.
- [8] Jan Chomicki, Jorge Lobo, and Shamim A. Naqvi. Conflict resolution using logic programming. *IEEE Trans. Knowl. Data Eng.*, 15(1):244–249, 2003.
- [9] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The platform for privacy preferences 1.0 (p3p1.0) specification. *W3C Recommendation*, 2002.
- [10] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In Morris Sloman, Jorge Lobo, and Emil Lupu, editors, *POLICY*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2001.
- [11] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Obligations and their interaction with programs. In Biskup and Lopez [6], pages 375–389.
- [12] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *ICSE*, pages 196–205. ACM, 2005.
- [13] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data-structure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.
- [14] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *CSFW*, pages 187–201. IEEE Computer Society, 2003.
- [15] Keith Irwin, Ting Yu, and William H. Winsborough. On the modeling and analysis of obligations. In Juels et al. [16], pages 134–143.
- [16] Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors. *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*. ACM, 2006.

- [17] Clare-Marie Karat, John Karat, Carolyn Brodie, and Jinjuan Feng. Evaluating interfaces for privacy policy rule authoring. In Rebecca E. Grinter, Tom Rodden, Paul M. Aoki, Edward Cutrell, Robin Jeffries, and Gary M. Olson, editors, *CHI*, pages 83–92. ACM, 2006.
- [18] J. Karat, C-M. Karat, , E. Bertino, N. Li, Q. Ni, C. Brodie, J. Lobo, S. Calo, L. Cranor, P. Kumaraguru, and R. Reeder. A policy framework for security and privacy management. *Submitted to IBM System Journal special issue on harmonizing security and privacy*.
- [19] John Karat, Clare-Marie Karat, Carolyn Brodie, and Jinjuan Feng. Privacy in information technology: designing to enable privacy policy management in organizations. *Int. J. Hum.-Comput. Stud.*, 63(1-2):153–174, 2005.
- [20] Vladimir Kolovski, James A. Hendler, and Bijan Parsia. Analyzing web access control policies. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 677–686. ACM, 2007.
- [21] Ninghui Li and Qihua Wang. Beyond separation of duty: an algebra for specifying high-level security policies. In Juels et al. [16], pages 356–369.
- [22] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Exam - a comprehensive environment for analysis of access control policies. In *Technical Report, Dept of Computer Science, Purdue University, USA*, 2007.
- [23] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy decomposition for collaborative access control. In *To Appear in Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.
- [24] D. Lin, P. Rao, E. Bertino, and J. Lobo. An approach to evaluate policy similarity. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1 – 10, 2007.
- [25] Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association.
- [26] T. Moses. Extensible access control markup language (XACML) version 2.0. *Technical report, OASIS*, 2005.
- [27] Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *To Appear in Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.
- [28] Qun Ni, Dan Lin, Elisa Bertino, and Jorge Lobo. Conditional privacy-aware role based access control. In Biskup and Lopez [6], pages 72–89.
- [29] Robert W. Reeder, Clare-Marie Karat, John Karat, and Carolyn Brodie. Usability challenges in security and privacy policy-authoring interfaces. In Maria Cecília Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, *INTERACT (2)*, volume 4663 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2007.
- [30] Qihua Wang and Ninghui Li. Satisfiability and resiliency in workflow systems. In Biskup and Lopez [6], pages 90–105.