

Integrity Constraints in Trust Management (Extended Abstract) *

Sandro Etalle
University of Twente
P.O.Box 217
7500AE Enschede, The Netherlands
s.etalles@utwente.nl

William H. Winsborough
George Mason University
4400 University Drive, MS 4A4 Fairfax, VA
22030, USA
wwinsborough@acm.org

ABSTRACT

We introduce the use, monitoring, and enforcement of integrity constraints in trust management-style authorization systems. We consider what portions of the policy state must be monitored to detect violations of integrity constraints. Then we address the fact that not all participants in a trust management system can be trusted to assist in such monitoring, and show how many integrity constraints can be monitored in a conservative manner so that trusted participants detect and report if the system enters a policy state from which evolution in unmonitored portions of the policy could lead to a constraint violation.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection; D.4.6 [Operating Systems]: Security and Protection—*Access Controls Information Flow Controls*

General Terms

Security, Theory, Languages

Keywords

access control, trust management, distributed system security, integrity

1. INTRODUCTION

Trust management [3] (TM) is an approach to managing authorization in environments where authority emanates from multiple sources. Authorization policy consists of statements issued by many participants, and resource sharing is facilitated by delegating authority from one principal to another.

*This work was partially supported by the BSIK Freeband project I-Share.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'05, June 1–3, 2005, Stockholm, Sweden.
Copyright 2005 ACM 1-59593-045-0/05/0006 ...\$5.00.

A particular authorization is decided by posing a query to the system. An evaluation procedure combines the statements issued by all relevant principals to derive the query's answer. By adding or removing a policy statement, a principal can potentially affect many authorizations of many principals.

One of the difficulties of operating in such a context is that at present no system exists for monitoring unexpected consequences of policy changes made by other principals. Basically, in present TM systems, delegating trust implies losing a great deal of control on the policy involved the delegation. Let us first see three examples of this.

Firstly, resources may become unavailable unexpectedly. Consider for instance a team leader who needs to be informed if members of his team suffer interruption in their authorization for mission-critical resources. If the team's mission involves rapid response, the notification of interruption should not depend on team members attempting to access a critical resource and discovering its unavailability only because the attempt fails. What is needed is that the policy change triggers a procedure that pushes the notification to the team leader.

Secondly: properties such as mutual exclusion cannot be guaranteed. While in the above example, the exceptional state involved someone losing authorization, having someone unexpectedly gain authorization can be just as important to detect. For instance, it should be possible to trigger an action if a principal becomes authorized for two mutually exclusive purposes. Mutual exclusion is an approach often used, for instance in RBAC systems [17], to enforce separation of duty, a classic device aimed at preventing fraud. By ensuring that no individual is authorized to complete all parts of a sensitive task, the technique ensures that only a colluding group could misuse the capability. Because the participants in a trust management system are autonomous, it is in general not possible to prevent a principal being given two authorizations. However, cooperating principals should be able to prevent another principal from gaining two mutually exclusive authorizations under the control of the cooperating group. What is needed is a way to distribute the mutual exclusivity requirement and monitor policy evolution to ensure that control over the key authorizations is not delegated outside the cooperating group.

Thirdly: quality cannot be monitored. Consider the situation in which the principal A states, for instance, that he considers expert anyone that B considers an expert (A delegates to B the definition of "expert"). In addition, A

expects experts to have a PhD degree. Now, A has no way of controlling that all experts added by B actually have doctorates. Of course, A could modify his policy as follows “ A considers expert anyone *holding a PhD* that B considers an expert”. However often it would be preferable for A to know whether a non-PhD had been added to the expert list because it might suggest to A that an exception to A ’s policy is acceptable, or that some other evolution of A ’s policy should take place (perhaps it is time to revoke the trust in B ’s experts). Thus, what A needs is to be able to monitor whether B ever decides that a non-PhD is an expert. Notice that this is what would happen in practice: before delegating to B the definition of expert A would normally put in place a monitoring activity to guarantee that B ’s expert fulfill the quality criteria. Unfortunately, present decentralized TM systems do not allow for such monitoring.

Summarizing, there is a need for a mechanism to monitor a TM system and to reveal when an exceptional state has been entered so that appropriate steps can be taken proactively. Ideally, it would even be possible to enlist the assistance of others in preventing exceptional states from arising. The problem of providing such a monitoring system is aggravated by the fact that changes are made by autonomous principals that may not agree or be trusted to assist in the monitoring.

In this paper we introduce a new trust management construct called a constraint, inspired by integrity constraints in database management systems (see, e.g. [8, 5]), that provides system participants the ability to monitor the evolution of the policy. The author of a constraint receives notification when the constraint is violated. This is achieved by enlisting the assistance of principals to which authority is delegated and triggering constraint checks when those principals make relevant policy changes. The emphasis in this paper is on determining whether a policy change is relevant, or can be ignored.

In addition we also consider the setting in which some principals are not trusted or willing to help monitoring a constraint. As mentioned above, in some environments, it is not appropriate to assume that all principals to whom one delegates authority will assist in monitoring one’s constraints. By providing a sufficiently expressive constraint language, we show how to limit to an arbitrary, specified set those principals that are trusted to cooperate in monitoring a constraint. This is done by allowing a constraint to express a security analysis problem of the kind formulated by Li et al. [14]. Such a constraint quantifies over policy states that are reachable by policy changes made by untrusted principals asking whether a given query holds either in all reachable states (universal quantification) or in some reachable state (existential quantification). By checking such a constraint each time the trusted principals make relevant policy changes, and committing their changes only if the constraint is satisfied, the trusted principals can ensure that a state violating the constraint is never entered, no matter what the untrusted principals do. They are able to do this because the untrusted principals are unable to affect the validity of the constraint.

The technical contribution in this paper is a method to identify portions of the policy state that must be monitored in order to detect constraint violations. We do this first under the assumption that all principals in the system can be trusted to assist in monitoring the portion of the policy

state under their control. We then relax this assumption by requiring only that a given portion of the policy can be reliably monitored. In this case, monitoring is carried out by using security analysis to assess the possibility of the constraint becoming violated by policy changes that cannot be monitored directly.

Section 2 discusses the TM policy language that we use. Section 3 identifies the portion of the policy state to be monitored for constraint violations, assuming all portions can be monitored. Section 4 shows how to monitor constraints for potential violations when not all parts of the policy state can be monitored directly. Section 5 discusses related work. Section 6 concludes. Due to space constraints, some of the proofs are omitted, and can be found in the appendix of [7].

2. PRELIMINARIES

Trust management [3, 1, 2, 16, 6, 4, 9, 10, 15, 14, 11, 13, 18] is an approach to access control in decentralized distributed systems with access control decisions based on policy statements issued by multiple principals. In trust management systems, statements that are maintained in a distributed manner are often digitally signed to ensure their authenticity and integrity; such statements are sometimes called *credentials* or *certificates*. This section presents the trust management language RT_0 [14], which we use in this paper.

The Language RT_0

A *principal* is a uniquely identified individual or process. Principals are denoted by names starting with an uppercase, typically, A, B, D .

A principal can define a *role*, which is indicated by principal’s name followed by the *role name*, separated by a dot. For instance $A.r$, and $GMU.students$ are roles. For the sake of simplicity we assume that A is the *owner* (or the administrator) of $A.r$, though the results of this papers apply also in the case $A.r$ is owned by some other principal. We use names starting with a lowercase letter (sometimes with subscripts) to indicate role names.

A role denotes a set of principals (the principals that populate it, i.e., the members of the role). To indicate which principals populate a role, RT_0 allows the owning principal to issue four kind of *policy statements*:

- *Simple Member*: $A.r \leftarrow D$
With this statement A asserts that D is a member of $A.r$.
- *Simple Inclusion*: $A.r \leftarrow B.r_1$
With this statement A asserts that $A.r$ includes (all members of) $B.r_1$. This represents a delegation from A to B , as B may add principals to become members of the role $A.r$ by issuing statements defining (and extending) $B.r_1$.
- *Linking Inclusion*: $A.r \leftarrow A.r_1.r_2$
We call $A.r_1.r_2$ a *linked role*. With this statement A asserts that $A.r$ includes $B.r_2$ for every B that is a member of $A.r_1$. This represents a delegation from A to all the members of the role $A.r_1$.
- *Intersection Inclusion*: $A.r \leftarrow B_1.r_1 \cap B_2.r_2$

We call $B_1.r_1 \cap B_2.r_2$ an *intersection*. With this statement A asserts that $A.r$ includes every principal who is a member of both $B_1.r_1$ and $B_2.r_2$. This represents partial delegations from A to B_1 and to B_2 .

For any statement $A.r \leftarrow e$, $A.r$ is called the *head* and e is called the *body* of the statement. We write $head(A.r \leftarrow e) = A.r$. The set of statements having head $A.r$ is called the *definition* of $A.r$.

The definition of RT_0 given here is a slightly simplified (yet expressively equivalent) version of the one given in [14]. A *policy state* (*state* for short, indicated by \mathcal{P}) is a set of policy statements. Given a state \mathcal{P} , we define the following: $\text{Principals}(\mathcal{P})$ is the set of principals in \mathcal{P} , $\text{Names}(\mathcal{P})$ is the set of role names in \mathcal{P} , and $\text{Roles}(\mathcal{P}) = \{A.r \mid A \in \text{Principals}(\mathcal{P}), r \in \text{Names}(\mathcal{P})\}$.

To express constraints, we need one last definition:

Definition 2.1 *Positive roles expressions* are defined by the following grammar:

- sets of principals are positive role expressions,
- roles are positive role expressions,
- union and intersections of positive role expressions are positive role expressions. \square

E.g., $A.r$, $A.r \cup \{A, B\}$ and $A.r \cap B.r_1.r_2$. Positive role expressions, and are denoted by Greek letters, ϕ , λ , and ϱ . A positive role expression containing no roles (but only sets of principals) is called *static*.

Semantics

The semantics of a policy state is defined by translating it into a logic program. The *semantic program*, $SP(\mathcal{P})$, of a state \mathcal{P} , is a Prolog program has one ternary predicate m . Intuitively, $m(A, r, D)$ means that D is a member of the role $A.r$.

Definition 2.2 (Semantic Program) Given a state \mathcal{P} , the *semantic program* $SP(\mathcal{P})$ for it is the logic program defined as follows: (here symbols that start with “?” represent logic variables)

- For each $A.r \leftarrow D \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $m(A, r, D)$
- For each $A.r \leftarrow B.r_1 \in \mathcal{P}$, add to $SP(\mathcal{P})$ the clause $m(A, r, ?Z) :- m(B, r_1, ?Z)$
- For each $A.r \leftarrow A.r_1.r_2 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $m(A, r, ?Z) :- m(A, r_1, ?Y), m(?Y, r_2, ?Z)$
- For each $A.r \leftarrow B_1.r_1 \cap B_2.r_2 \in \mathcal{P}$ add to $SP(\mathcal{P})$ the clause $m(A, r, ?Z) :- m(B_1, r_1, ?Z), m(B_2, r_2, ?Z)$. \square

We can now define the semantics of a role in a state.

Definition 2.3 (Semantics) Given a state \mathcal{P} , the semantics of a role $A.r$ is defined in terms of atoms entailed by the semantic program:

- $\llbracket A.r \rrbracket_{SP(\mathcal{P})} = \{Z \mid SP(\mathcal{P}) \models m(A, r, Z)\}$ \square

We extend this semantics to positive role expressions in the natural way as follows:

$$\begin{aligned} \llbracket \{D_1, \dots, D_n\} \rrbracket_{SP(\mathcal{P})} &= \{D_1, \dots, D_n\} \\ \llbracket \phi_1 \cup \phi_2 \rrbracket_{SP(\mathcal{P})} &= \llbracket \phi_1 \rrbracket_{SP(\mathcal{P})} \cup \llbracket \phi_2 \rrbracket_{SP(\mathcal{P})} \\ \llbracket \phi_1 \cap \phi_2 \rrbracket_{SP(\mathcal{P})} &= \llbracket \phi_1 \rrbracket_{SP(\mathcal{P})} \cap \llbracket \phi_2 \rrbracket_{SP(\mathcal{P})} \end{aligned}$$

3. CONSTRAINTS

Consider a state \mathcal{P} , which might change in time. We are interested in defining a *constraint*, which intuitively is a *query* that is intended to hold throughout the state changes. To this end, we focus on the class of constraints already considered for the purposes of security analysis in [12]. These constraints express set containment.

Definition 3.1 A *constraint* is an expression of the form $\langle O, \lambda \sqsubseteq \varrho \rangle$, in which O is a principal called the *owner* of the constraint, and λ and ϱ are positive role expressions. \square

The following definition clarifies that \sqsubseteq represents set containment.

Definition 3.2 Let \mathcal{P} be a state and \mathcal{Q} be the constraint $\langle O, \lambda \sqsubseteq \varrho \rangle$, we say that

- \mathcal{P} *satisfies* \mathcal{Q} ($\mathcal{P} \vdash \mathcal{Q}$) iff $\llbracket \lambda \rrbracket_{SP(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{SP(\mathcal{P})}$

(\mathcal{P} *violates* \mathcal{Q} otherwise) \square

Constraints of this form can capture many important and intuitive requirements.

- Consider $\langle O, \{\text{Bob}\} \cap A.r \sqsubseteq \emptyset \rangle$. This constraint captures a safety requirement that Bob must not become a member of $A.r$.
- The constraint $\langle O, \{\text{Alice}\} \sqsubseteq A.r \rangle$ captures the availability requirement that Alice must be authorized for $A.r$.
- The constraint $\langle O, A.\text{manager} \cap B.\text{controller} \sqsubseteq \emptyset \rangle$ captures the mutual exclusivity requirement that no one must be authorized for both $A.\text{manager}$ and $B.\text{controller}$.

Example 3.3 Suppose the Bureau of Alcohol, Tobacco, Firearms and Explosives (*ATF*) operates a database containing information about hazardous materials (*HAZMAT*) for use by emergency response personnel. The *ATF* individually authorizes users so as to retain tight control over the sensitive information contained in the database. It does this by issuing statements such as:

$$ATF.\text{hazmatDB} \leftarrow \text{Rollins} \quad (1)$$

The Emergency Response Center (*Emergency*) wants to ensure that all its hazmat emergency response personnel have access to the database at all times. This is expressed by the constraint

$$\langle \text{Emergency}, \text{Emergency.hazmatPersonnel} \sqsubseteq ATF.\text{hazmatDB} \rangle$$

We assume that *Emergency.hazmatPersonnel* is defined by the collection of statements (2) \dots (8) in Table 1. Suppose

Table 1: Policy State of Example 3.3

$ATF.hazmatDB \leftarrow Rollins$	(1)
$Emergency.hazmatPersonnel \leftarrow Emergency.responsePersonnel \cap ATF.hazmatTraining$	(2)
$Emergency.responsePersonnel \leftarrow Emergency.dept.responsePersonnel$	(3)
$Emergency.dept \leftarrow Fire$	(4)
$Emergency.dept \leftarrow Police$	(5)
$ATF.hazmatTraining \leftarrow Rollins$	(6)
$ATF.hazmatTraining \leftarrow Burke$	(7)
$ATF.hazmatTraining \leftarrow O'Connell$	(8)

Additional Statements

$Police.responsePersonnel \leftarrow Rollins$	(9)
$Police.responsePersonnel \leftarrow Burke$	(10)

The semantics of $\mathcal{P} = \{(1), \dots, (8)\}$ is

$\llbracket ATF.hazmatDB \rrbracket_{SP(\mathcal{P})}$	$=$	$\{Rollins\}$
$\llbracket ATF.hazmatTraining \rrbracket_{SP(\mathcal{P})}$	$=$	$\{Rollins, Burke, O'Connell\}$
$\llbracket Emergency.hazmatPersonnel \rrbracket_{SP(\mathcal{P})}$	$=$	\emptyset
$\llbracket Emergency.responsePersonnel \rrbracket_{SP(\mathcal{P})}$	$=$	\emptyset
$\llbracket Emergency.dept \rrbracket_{SP(\mathcal{P})}$	$=$	$\{Fire, Police\}$

The semantics of $\mathcal{P}' = \mathcal{P} \cup \{(9), (10)\}$ is

$\llbracket ATF.hazmatDB \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Rollins\}$
$\llbracket ATF.hazmatTraining \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Rollins, Burke, O'Connell\}$
$\llbracket Emergency.hazmatPersonnel \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Rollins, Burke\}$
$\llbracket Emergency.responsePersonnel \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Rollins, Burke\}$
$\llbracket Emergency.dept \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Fire, Police\}$
$\llbracket Police.responsePersonnel \rrbracket_{SP(\mathcal{P}')}$	$=$	$\{Rollins, Burke\}$

the following two statements are added:

$Police.responsePersonnel \leftarrow Rollins$	(9)
$Police.responsePersonnel \leftarrow Burke$	(10)

When these statements are added, it must be checked whether they cause violations of the constraint. Credential (9) does not cause a violation, but (10) does, and the Emergency Response Center must be notified accordingly. \square

3.1 Monitoring Constraints

We now see how we can put in place a system for monitoring constraint violations. Let \mathcal{P} be a state, and consider the constraint $\mathcal{Q} = \langle O, \lambda \sqsubseteq \varrho \rangle$. Assuming that \mathcal{P} changes in time, we are interested in monitoring when \mathcal{Q} is violated.

Definition 3.4 Let $\mathcal{P} \mapsto \mathcal{P}'$ be a state change from \mathcal{P} to \mathcal{P}' . We say that

- the change *violates* \mathcal{Q} if $\mathcal{P} \vdash \mathcal{Q}$ and $\mathcal{P}' \not\vdash \mathcal{Q}$

Notice that if a change violates the constraint, then there exists D such that $D \notin \llbracket \lambda \rrbracket_{SP(\mathcal{P})} \setminus \llbracket \varrho \rrbracket_{SP(\mathcal{P})}$, while $D \in \llbracket \lambda \rrbracket_{SP(\mathcal{P}')} \setminus \llbracket \varrho \rrbracket_{SP(\mathcal{P}')}$. This remark points out an important feature of containment constraints: that if they are violated then there exists a specific set of principals violating it.

To monitor the system, a feature of *RT* we are going to exploit is its monotonicity: adding a statement to \mathcal{P} cannot cause the set semantics of a role to shrink. Similarly, removing a statement cannot cause the set semantics to grow. Formally, for each role $A.r$ and each statement *stmt*:

$$\begin{aligned} \llbracket A.r \rrbracket_{SP(\mathcal{P})} &\subseteq \llbracket A.r \rrbracket_{SP(\mathcal{P} \cup \{stmt\})} \\ \llbracket A.r \rrbracket_{SP(\mathcal{P})} &\supseteq \llbracket A.r \rrbracket_{SP(\mathcal{P} \setminus \{stmt\})} \end{aligned} \quad (1)$$

Therefore, adding a statement to \mathcal{P} can only augment the set $\llbracket \lambda \rrbracket_{SP(\mathcal{P})}$ and $\llbracket \varrho \rrbracket_{SP(\mathcal{P})}$. Consequently, if we assume that \mathcal{P} initially satisfies $\lambda \sqsubseteq \varrho$, we see the following:

- Adding a statement to \mathcal{P} can yield to a violation of $\lambda \sqsubseteq \varrho$ only if the addition affects $\llbracket \lambda \rrbracket_{SP(\mathcal{P})}$.
- Removing a statement from \mathcal{P} can yield to a violation of $\lambda \sqsubseteq \varrho$ only if the removal affects $\llbracket \varrho \rrbracket_{SP(\mathcal{P})}$.

We now want to further isolate the roles that might influence the satisfaction of a constraint.

Example 3.5 Consider the following set of statements.

$A.r \leftarrow A.r.r$	(2)
$A.r \leftarrow B$	(3)
$B.r \leftarrow C$	(4)
$C.r \leftarrow D.r$	(5)
$E.r \leftarrow F$	(6)

It is easy to see that $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$ is $\{B, C\}$. Notice now that if we add a statement $D.r \leftarrow E$, then $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$ grows to $\{B, C, E, F\}$. Therefore we can say that $D.r$ may *positively* affect $A.r$. We see that $\{A.r, B.r, C.r, D.r\}$ is the set of roles that can positively affect $A.r$. Dually, we can define the set of roles that may affect the *shrinking* of $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$. Here, it is easy to see that the only way of “reducing” the semantics $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$ of $A.r$ is by removing one of the statements (2), (3) or (4). Since these statements define the roles $A.r$ and $B.r$ we can say that $\{A.r, B.r\}$ is the set of roles that can *negatively* affect $A.r$. \square

This section constructs two sets of roles whose definitions determine the membership of a given role $X.u$ in state \mathcal{P} . If the membership of $X.u$ were to grow, some role in one of these sets would have to have a new statement in its definition, and if the membership of $X.u$ were to shrink, some role in the other set would have to have a statement in its definition revoked.

Positive Dependencies

Given a set \mathcal{P} and a role $A.r$ we want to isolate a set $\Gamma_{\mathcal{P}}(A.r)$ of roles we have to monitor, as they might affect the *growth* of $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$.

Definition 3.6 Let $A.r$ be a role and \mathcal{P} be a state; $\Gamma_{\mathcal{P}}(A.r)$ is the least set of roles containing $A.r$ and satisfying the following:

- If $B.r_0 \in \Gamma_{\mathcal{P}}(A.r)$ and $B.r_0 \leftarrow B.r_1 \in \mathcal{P}$, then $B.r_1 \in \Gamma_{\mathcal{P}}(A.r)$.
- If $B.r_0 \in \Gamma_{\mathcal{P}}(A.r)$ and $B.r_0 \leftarrow B.r_1.r_2 \in \mathcal{P}$, then $B.r_1 \in \Gamma_{\mathcal{P}}(A.r)$ and $X.r_2 \in \Gamma_{\mathcal{P}}(A.r)$ for all $X \in \llbracket B.r_1 \rrbracket_{SP(\mathcal{P})}$.
- If $B.r_0 \in \Gamma_{\mathcal{P}}(A.r)$ and $B.r_0 \leftarrow B_1.r_1 \cap \dots \cap B_n.r_n \in \mathcal{P}$, then for each $i \in [1, n]$ $B_i.r_i \in \Gamma_{\mathcal{P}}(A.r)$. \square

The main properties of $\Gamma_{\mathcal{P}}(\cdot)$ we will make use of are summarized in the following lemma, which is proved in [7].

Lemma 3.7 Let $\mathcal{P}' = \mathcal{P} \cup \{stmt\}$, where $head(stmt) \notin \Gamma_{\mathcal{P}}(A.r)$, then

- (a) $\llbracket A.r \rrbracket_{SP(\mathcal{P})} = \llbracket A.r \rrbracket_{SP(\mathcal{P}')}$, and
- (b) $\Gamma_{\mathcal{P}}(A.r) = \Gamma_{\mathcal{P}'}(A.r)$.

Moreover, if \mathcal{P}' is obtained from \mathcal{P} by (a) adding zero or more statements whose head is not in $\Gamma_{\mathcal{P}}(A.r)$, and (b) removing zero or more statements, then

- (c) $\llbracket A.r \rrbracket_{SP(\mathcal{P})} \supseteq \llbracket A.r \rrbracket_{SP(\mathcal{P}')}$, and
- (d) $\Gamma_{\mathcal{P}}(A.r) \supseteq \Gamma_{\mathcal{P}'}(A.r)$. \square

Example 3.8

- Returning to Example 3.3, the left-hand side of the constraint $Emergency.hazmatPersonnel \sqsubseteq ATF.hazmatDB$ is $Emergency.hazmatPersonnel$. So

$$\Gamma_{\mathcal{P}}(Emergency.hazmatPersonnel) = \left\{ \begin{array}{l} Emergency.hazmatPersonnel, \\ Emergency.responsePersonnel, \\ ATF.hazmatTraining, \\ Emergency.dept, \\ Fire.responsePersonnel, \\ Police.responsePersonnel \end{array} \right\}$$

is the set of roles for which addition of new statements must be monitored.

- Consider the policy state in Example 3.5. Then $\Gamma_{\mathcal{P}}(A.r) = \{A.r, B.r, C.r, D.r\}$.

- Suppose \mathcal{P} contains only the statement $\{A.r_0 \leftarrow A.r_1.r_2\}$. Then $\Gamma_{\mathcal{P}}(A.r_0) = \{A.r_0, A.r_1\}$, and $\llbracket A.r_0 \rrbracket_{SP(\mathcal{P})} = \emptyset$. Now, if we add a new statement $A.r_1 \leftarrow B$ to \mathcal{P} (obtaining \mathcal{P}') then $\llbracket A.r_0 \rrbracket_{SP(\mathcal{P}')} = \emptyset$ is still the empty set, while $\Gamma_{\mathcal{P}}(A.r_0)$ is now $\{A.r_0, A.r_1, B.r_2\}$. \square

For efficiency reasons, we would like $\Gamma_{\mathcal{P}}(A.r)$ to be as small as possible, while maintaining the properties stated in Lemma 3.7. There are two reasons why $\Gamma_{\mathcal{P}}(A.r)$ is non-minimal: the first reason is that an intersection inclusion can act as a filter. For instance, if $A.r \leftarrow B_1.r_1 \cap B_2.r_2 \in \mathcal{P}$ and $\llbracket B_1.r_1 \rrbracket_{SP(\mathcal{P})} = \emptyset$, there is no point in adding $B_2.r_2$ to $\Gamma_{\mathcal{P}}(A.r)$ as any change to $B_2.r_2$ will not affect the membership to $A.r$. The second reason concerns linked roles: if $A.r \leftarrow A.r_1.r_2 \in \mathcal{P}$ and there exists no role $B.r_2$ such that for some D , $D \in \llbracket B.r_2 \rrbracket_{SP(\mathcal{P})} \setminus \llbracket A.r \rrbracket_{SP(\mathcal{P})}$, then we could avoid adding $A.r_1$ to $\Gamma_{\mathcal{P}}(A.r)$, as any addition to $B_2.r_2$ would not affect the membership to $A.r$. However, refining the definition $\Gamma_{\mathcal{P}}(A.r)$ to take these factors into consideration would make its definition more complex than seems practical.

Negative Dependencies

Now, we need to isolate the dual of $\Gamma_{\mathcal{P}}(A.r)$, i.e., a set of roles that might cause $\llbracket A.r \rrbracket_{SP(\mathcal{P})}$ to shrink. To this end, we say that Σ is a \mathcal{P} -support of D for $A.r$ if the roles in Σ carry enough information to demonstrate that $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P})}$. We denote by $\mathcal{P}|_{\Sigma}$ the restriction of \mathcal{P} to the roles in Σ , $\mathcal{P}|_{\Sigma} = \{stmt \in \mathcal{P} | head(stmt) \in \Sigma\}$

Definition 3.9 Let $A.r$ be a role, D be a principal, \mathcal{P} be a set of statements and Σ be a set of roles.

- We say that Σ is a \mathcal{P} -support of D for $A.r$ if $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P}|_{\Sigma})}$.
- For $L \subseteq \text{Principals}(\mathcal{P})$, we say that Σ is a \mathcal{P} -support of L for $A.r$ if $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P}|_{\Sigma})}$ for every $D \in L$.
- We say that Σ is a \mathcal{P} -support for $A.r$ if and only if it is a \mathcal{P} -support of every $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P})}$. \square

Example 3.10

- (i) Consider again the policy state in Example 3.5. Any set containing $\{A.r, B.r\}$ as a subset is a support for $A.r$.
- (ii) In case of redundancies, minimal support might not be unique. Consider

$$\begin{array}{l} A.r \leftarrow B.r \\ A.r \leftarrow C.r \\ B.r \leftarrow F \\ C.r \leftarrow F \end{array}$$

Here, both $\{A.r, B.r\}$ and $\{A.r, C.r\}$ are support for $A.r$. \square

We can now state the counterpart of Lemma 3.7.

Lemma 3.11 Let $A.r$ be a role, D be a principal, \mathcal{P} be a state and Σ be a \mathcal{P} -support of D for $A.r$. Then

1. $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P})}$

Moreover, if \mathcal{P}' is obtained from \mathcal{P} by (a) removing zero or more statements whose head is not in Σ , and (b) adding zero or more statements, then

2. Σ is a \mathcal{P}' -support for $A.r$, and therefore
3. $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P}')}$

Proof. Point 1 follows immediately from the fact that, by monotonicity, $\llbracket A.r \rrbracket_{SP(\mathcal{P})} \supseteq \llbracket A.r \rrbracket_{SP(\mathcal{P}|\Sigma)}$. For points 2 and 3, by the construction of \mathcal{P}' we have that $\mathcal{P}|\Sigma \subseteq \mathcal{P}'$, so the results follows from the definition of support and the fact that the semantics is monotonic. \square

To build a \mathcal{P} -support of D for $A.r$ one basically has to collect all the roles used to prove that $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P})}$. In [7] we give an algorithm to compute minimal \mathcal{P} -support while evaluating role membership.

Putting Things Together

We can now prove the result we were aiming at. Suppose we need to deploy the integrity constraint $\mathcal{Q} = \lambda \sqsubseteq \varrho$ on \mathcal{P} . The first step we need to take is to check if \mathcal{P} satisfies \mathcal{Q} . This is can be done as follows:

1. First, $\llbracket \lambda \rrbracket_{SP(\mathcal{P})}$ is computed.
2. Then, for each $D \in \llbracket \lambda \rrbracket_{SP(\mathcal{P})}$, we check that $D \in \llbracket \varrho \rrbracket_{SP(\mathcal{P})}$.

In step 2, while checking that $D \in \llbracket \lambda \rrbracket_{SP(\mathcal{P})}$ it is usually possible to build for free a \mathcal{P} -support of D in ϱ . Once we have checked that \mathcal{P} satisfies \mathcal{Q} , we want to make sure that changes to \mathcal{P} do not cause a violation of \mathcal{Q} . For this we have the following.

Theorem 3.12 (Main) Assume that \mathcal{P} satisfies the constraint $\langle O, \lambda \sqsubseteq \varrho \rangle$. Let Σ be a \mathcal{P} -support of $\llbracket \lambda \rrbracket_{SP(\mathcal{P})}$ for ϱ , and let $\mathcal{P} \mapsto \mathcal{P}'$ be a (possibly multistep) change from \mathcal{P} to \mathcal{P}' . If

- (i) $\forall stmt \in \mathcal{P}' \setminus \mathcal{P}, head(stmt) \notin \Gamma_{\mathcal{P}}(\lambda)$, and
- (ii) $\forall stmt \in \mathcal{P} \setminus \mathcal{P}', head(stmt) \notin \Sigma$

Then \mathcal{P}' satisfies the constraint $\langle O, \lambda \sqsubseteq \varrho \rangle$ as well.

Proof.

Take any $D \in \llbracket \lambda \rrbracket_{SP(\mathcal{P}')}$
 By Lemma 3.7, $D \in \llbracket \lambda \rrbracket_{SP(\mathcal{P})}$
 Since by assumption, $\mathcal{P} \vdash \lambda \sqsubseteq \varrho$, $D \in \llbracket \varrho \rrbracket_{SP(\mathcal{P})}$
 By Lemma 3.11, $D \in \llbracket \varrho \rrbracket_{SP(\mathcal{P}')}$

Hence the thesis. \square

Theorem 3.12 also shows that, as long as the changes to \mathcal{P} satisfy (i) and (ii), we do not have to recompute the set $\Gamma_{\mathcal{P}}(\lambda)$ or the support Σ . Technically, this is due to the fact that changes satisfying (i) and (ii) do not affect Σ (by Lemma 3.11, Σ is still a support of ϱ), and can only reduce the set $\Gamma_{\mathcal{P}}(\lambda)$ (by Lemma 3.7). When statements defining roles in $\Gamma_{\mathcal{P}}(\lambda)$ are issued, (i) is violated, and when statements defining roles in Σ are revoked, (ii) is violated. At these times, the constraint must be checked and the sets $\Gamma_{\mathcal{P}}(\lambda)$ and Σ must be recomputed.

The theorem indicates how a system for monitoring constraints should be deployed: the first step (mentioned above) is to check that \mathcal{P} satisfies $\lambda \sqsubseteq \varrho$. While doing this, we can build an appropriate Σ . Secondly, we have to build $\Gamma_{\mathcal{P}}(\lambda)$.

Thirdly, we need to put in place *monitoring* of the roles in Σ and in $\Gamma_{\mathcal{P}}(\lambda)$ such that each time a statement defining a role in $\Gamma_{\mathcal{P}}(\lambda)$ (resp. Σ) is added to (resp. deleted from) \mathcal{P} , the constraint owner is warned. When the constraint owner receives a warning he has to (a) check whether the constraint still holds, and (b) recompute $\Gamma_{\mathcal{P}}(\lambda)$ and Σ .

Example 3.13

- Returning to Example 3.3, to monitor $\langle Emergency, Emergency.hazmatPersonnel \sqsubseteq ATF.hazmatDB \rangle$, we must monitor revocation of definitions of roles in some \mathcal{P} -support of each member of $\llbracket Emergency.hazmatPersonnel \rrbracket_{SP(\mathcal{P})}$ for $ATF.hazmatDB$. In this example, $\Sigma = \{ATF.hazmatDB\}$ is a \mathcal{P} -support of each such member for $ATF.hazmatDB$. We must also monitor additions to $\Gamma_{\mathcal{P}}(Emergency.hazmatPersonnel)$, as discussed in Example 3.8. If new statements are added defining other roles, no action has to be taken. Similarly, if statement (10), $Police.responsePersonnel \leftarrow Burke$, were removed, no action would be necessary because $Police.responsePersonnel$ is not in Σ .
- Consider now Example 3.10 (ii), together with the query $\{F\} \sqsubseteq A.r$. To apply Theorem 3.12, we have to choose one support of F for $A.r$ (the two candidate support are $\{A.r, B.r\}$ and $\{A.r, C.r\}$) and monitor the roles in it. Suppose we choose $\Sigma = \{A.r, B.r\}$. Suppose we now remove the statement $B.r \leftarrow F$. This does not yield to a violation of the constraint, but we do have to recompute Σ , which now becomes $\{A.r, C.r\}$.
- Finally, it is also instructive to see that a change in $\Gamma_{\mathcal{P}}(\lambda)$ might require recomputing Σ , even if it does not entail a violation of the constraint. Let \mathcal{P} be the following set of statements:

$$\begin{aligned} A.r &\leftarrow E \\ B.r &\leftarrow C.r \\ B.r &\leftarrow D.r \\ C.r &\leftarrow E \\ D.r &\leftarrow F \end{aligned}$$

together with the constraint $A.r \sqsubseteq B.r$. This constraint is satisfied and to monitor its evolution we have to monitor the roles in $\Gamma_{\mathcal{P}}(A.r) = \{A.r\}$ and $\Sigma = \{B.r, C.r\}$. Now if we add the statement $A.r \leftarrow F$ then the constraint owner is warned that a change in $\Gamma_{\mathcal{P}}(\lambda)$ has occurred. The constraint owner can check that the constraint is still satisfied in $\mathcal{P}' = \mathcal{P} \cup \{A.r \leftarrow F\}$; however Σ has to be recomputed to take into account that it should be a \mathcal{P}' -support of F too. The new Σ is $\{B.r, C.r, D.r\}$. \square

3.2 Alternative Support Definition

We have defined the \mathcal{P} -support Σ to be a set of roles. Alternatively, we could have defined Σ to be a set of *credentials*.

Definition 3.14 (Alternative definition of support)

Let $A.r$ be a role, D be a principal, \mathcal{P} be a set of statements and $\Sigma \subseteq \mathcal{P}$ be a set of credentials

- We say that Σ is a \mathcal{P} -support of D for $A.r$ if $D \in \llbracket A.r \rrbracket_{SP(\Sigma)}$.
- For $L \subseteq \text{Principals}(\mathcal{P})$, we say that Σ is a \mathcal{P} -support of L for $A.r$ if $D \in \llbracket A.r \rrbracket_{SP(\Sigma)}$ for every $D \in L$.
- We say that Σ is a \mathcal{P} -support for $A.r$ if and only if it is a \mathcal{P} -support of every $D \in \llbracket A.r \rrbracket_{SP(\mathcal{P})}$. \square

Monitoring constraint using this definition requires more machinery than using Definition 3.9, but it could yield to a more efficient implementation. With this definition one monitors the *credentials* and not the *roles* which might affect the right hand side of the constraint. Therefore, to apply this definition one needs a mechanism for monitoring every single credential of Σ (which might be difficult).

Theorem 3.15 (Main - with alternative definition)

Assume that \mathcal{P} satisfies the constraint $\langle O, \lambda \sqsubseteq \varrho \rangle$. Let Σ be a \mathcal{P} -support of $\llbracket \lambda \rrbracket_{SP(\mathcal{P})}$ for ϱ (according to Definition 3.14), and let $\mathcal{P} \mapsto \mathcal{P}'$ be a (possibly multistep) change from \mathcal{P} to \mathcal{P}' . If

- (i) $\forall stmt \in \mathcal{P}' \setminus \mathcal{P}, head(stmt) \notin \Gamma_{\mathcal{P}}(\lambda)$, and
- (ii) $\forall stmt \in \mathcal{P} \setminus \mathcal{P}', stmt \notin \Sigma$

Then \mathcal{P}' satisfies $\langle O, \lambda \sqsubseteq \varrho \rangle$ as well. \square

The advantage of Definition 3.14, is that the hypothesis of Theorem 3.15 hold more often than those of Theorem 3.12. In other words, using Definition 3.14 one has to check whether the query still holds and to recalculate $\Gamma_{\mathcal{P}}(\lambda)$ and Σ less often than with Definition 3.9.

4. MONITORING WHEN NOT ALL PARTICIPANTS ARE TRUSTED TO HELP

The previous section showed how principals in a trust management system can monitor integrity constraints by monitoring changes in the definitions of certain roles. This section considers the problem of monitoring integrity constraints when not all principals in the system agree to assist in monitoring their roles. The idea is to make the assumption that the owners of a certain set of roles are trusted to monitor new statements added to their definitions. We call these the growth-trusted roles and denote them by \mathcal{G} . Similarly, the owners of a set of shrink-trusted roles, denoted \mathcal{S} , are trusted to monitor statements removed from their definitions. The owners of these roles are trusted to test whether changes made to untrusted roles could violate the constraint and, if so, to signal that potential violation. We call the pair $\mathcal{R} = (\mathcal{G}, \mathcal{S})$ a *role monitor* because it indicates the roles that can be monitored with respect to growth and shrinkage.

Definition 4.1 (Reachable) In the presence of a role monitor \mathcal{R} , we say that \mathcal{P}' is \mathcal{R} -reachable from \mathcal{P} if \mathcal{P}' can be obtained from \mathcal{P} without adding any statements defining roles in \mathcal{G} or removing any statements defining roles in \mathcal{S} . That is to say, $\{stmt \in \mathcal{P}' \mid head(stmt) \in \mathcal{G}\} \subseteq \mathcal{P}$ and $\{stmt \in \mathcal{P} \mid head(stmt) \in \mathcal{S}\} \subseteq \mathcal{P}'$. \square

The problem we address is to monitor whether the system ever enters a state \mathcal{P} from which some reachable \mathcal{P}' violates $\lambda \sqsubseteq \varrho$. This problem is closely related to the security analysis problem [12], which also is defined in terms of a role monitor $\mathcal{R} = (\mathcal{G}, \mathcal{S})$, although in that context it is called a restriction rule. In security analysis, the definitions of roles in \mathcal{G} are assumed not to grow and those of roles in \mathcal{S} , not to shrink; the security analysis problem is to determine whether other changes to the policy state could cause a constraint to become violated. In [12] it was shown that this problem is decidable (**coNEXP**) for RT_0 over the class of constraints we consider here, and that it is polynomial for an important subclass of those constraints. What has not been shown before, and what we show in this section, is how to identify subsets of \mathcal{G} and \mathcal{S} that need to be monitored so that security analysis can be used to maintain integrity constraints.

In the rest of this section, we introduce alternative semantics that can be used to answer questions about policy states that are reachable through changes to the definitions of untrusted roles. We then formalize sets of roles that must be monitored and show that monitoring these roles is sufficient. Finally, we provide a method for monitoring integrity constraints when not all principals in the system are trusted to assist the process.

Alternative Semantics

We now recall two non-standard semantics for a policy state \mathcal{P} and role monitor \mathcal{R} . These were introduced [12] for computing the lower and upper bounds on role memberships under the assumption that the definition of roles in \mathcal{G} do not grow and the definition of roles in \mathcal{S} do not shrink. We first recall the lower-bound program for a state \mathcal{P} and a restriction \mathcal{R} ; this program enables one to compute the lower-bounds of every role.

Definition 4.2 (Lower-Bound Program [12]) Given \mathcal{P} and \mathcal{R} , the *lower-bound program* for them, $LB(\mathcal{P}, \mathcal{R})$, is constructed as follows:

- (b1) For each $A.r \leftarrow D$ in $\mathcal{P}|_{\mathcal{R}}$, add $lb(A, r, D)$
- (b2) For each $A.r \leftarrow B.r_1$ in $\mathcal{P}|_{\mathcal{R}}$, add $lb(A, r, ?Z) :- lb(B, r_1, ?Z)$
- (b3) For each $A.r \leftarrow A.r_1.r_2$ in $\mathcal{P}|_{\mathcal{R}}$, add $lb(A, r, ?Z) :- lb(A, r_1, ?Y), lb(?Y, r_2, ?Z)$
- (b4) For each $A.r \leftarrow B_1.r_1 \cap B_2.r_2$ in $\mathcal{P}|_{\mathcal{R}}$, add $lb(A, r, ?Z) :- lb(B_1, r_1, ?Z), lb(B_2, r_2, ?Z)$. \square

We now recall the upper-bound program for a state \mathcal{P} and a role monitor \mathcal{R} . This program enables one to simulate the upper-bound of any role.

Definition 4.3 (Upper-Bound Program [12]) Given \mathcal{P} and $\mathcal{R} = (\mathcal{G}, \mathcal{S})$, their upper-bound program, $UB(\mathcal{P}, \mathcal{R})$, is constructed as follows. (\top is a special principal symbol not occurring in \mathcal{P} , \mathcal{R} , or any query \mathcal{Q} .)

- (u) Add $ub(\top, ?r, ?Z)$
- (u0) For each $A.r \in \text{Roles}(\mathcal{P}) \setminus \mathcal{G}$, add $ub(A, r, ?Z)$

- (u1) For each $A.r \leftarrow D$ in \mathcal{P} , add
 $ub(A, r, D)$
- (u2) For each $A.r \leftarrow B.r_1$ in \mathcal{P} , add
 $ub(A, r, ?Z) :- ub(B, r_1, ?Z)$
- (u3) For each $A.r \leftarrow A.r_1.r_2$ in \mathcal{P} , add
 $ub(A, r, ?Z) :- ub(A, r_1, ?Y), ub(?Y, r_2, ?Z)$
- (u4) For each $A.r \leftarrow B_1.r_1 \cap B_2.r_2$ in \mathcal{P} , add
 $ub(A, r, ?Z) :- ub(B_1, r_1, ?Z), ub(B_2, r_2, ?Z)$ \square

The rules (u1) to (u4) follow from the meanings of the four types of statements and are similar to the semantic program construction in Definition 2.2. The rule (u0) means that for any role $A.r$ not in \mathcal{G} , the upper-bound of $A.r$ contains every principal. The rule (u) means that for any role name r , the upper-bound of $\top.r$ contains every principal. This is so because \top does not appear in \mathcal{G} . The rule (u) is needed because given $A.r \leftarrow A.r_1.r_2$, where $A.r \in \mathcal{G}$ and $A.r_1 \notin \mathcal{G}$, we should ensure that the upper-bound of $A.r$ contains every principal. We define:

$$\llbracket A.r \rrbracket_{UB(\mathcal{P})} = \{Z \mid ub(\mathcal{P}) \models m(A, r, Z)\} \quad (7)$$

$$\llbracket A.r \rrbracket_{LB(\mathcal{P})} = \{Z \mid lb(\mathcal{P}) \models m(A, r, Z)\} \quad (8)$$

And by definition we have that

Remark 4.4

- If $A.r \notin \mathcal{S}$ then $\llbracket A.r \rrbracket_{LB(\mathcal{P})} = \emptyset$.
- If $A.r \notin \mathcal{G}$ then $\llbracket A.r \rrbracket_{UB(\mathcal{P})} = \text{Principals}(\mathcal{P}) \cup \{\top\}$. \square

The next theorem gives the link between the two new semantics and the problem of checking that a constraint is satisfied in all reachable \mathcal{P}' .

Theorem 4.5 ([12]) Let \mathcal{R} be a role monitor, \mathcal{P} be a state, and $\lambda \sqsubseteq \varrho$ be a containment constraint.

- If $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$ then $\mathcal{P}' \vdash \lambda \sqsubseteq \varrho$ for each \mathcal{P}' reachable from \mathcal{P} ,
- if either λ or ϱ is *static* (i.e., it is a set of principals) then $\mathcal{P}' \vdash \lambda \sqsubseteq \varrho$ for each \mathcal{P}' reachable from \mathcal{P} implies that $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$ ¹. \square

We now proceed as in the previous section, by identifying the roles we have to monitor.

Positive Dependencies, with Untrusted Roles

In the light of Theorem 4.5, given a state \mathcal{P} , a role monitor \mathcal{R} , and a role $A.r$, we want to isolate a set $\Gamma_{\mathcal{P}}^{\mathcal{G}}(A.r)$ of roles we have to monitor, as they might affect the *growth* of $\llbracket A.r \rrbracket_{UB(\mathcal{P})}$. One might think that when some roles are untrusted, we need only restrict $\Gamma_{\mathcal{P}}(A.r)$ to the \mathcal{G} -roles (or to check that $\Gamma_{\mathcal{P}}(A.r) \subseteq \mathcal{G}$). The following example shows that this is not adequate. Consider the constraint $A.r \sqsubseteq B.r$, where $A.r$ is defined by

$$A.r \leftarrow C.r \cap D.r \quad (9)$$

$$D.r \leftarrow E.r \quad (10)$$

...

¹Actually, though we do not prove it here, we believe that a stronger version of this part holds, stating that if $\Gamma_{\mathcal{P}}(\lambda) \cap \Gamma_{\mathcal{P}}(\varrho) = \emptyset$ then $\mathcal{P}' \vdash \lambda \sqsubseteq \varrho$ for each \mathcal{P}' reachable from \mathcal{P} implies that $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$.

$A.r$ depends on $C.r$, $D.r$ and $E.r$ (which are in $\Gamma_{\mathcal{P}}(A.r)$), and, if we used the method of the previous section, we would have to monitor all three of them. We now make two observations about monitoring when it is not possible to monitor all three roles. First, if $E.r$ is not in \mathcal{G} , we cannot monitor it. This implies that there is no point in monitoring $D.r$ either, as it directly depends on $E.r$. Second if $D.r$ is not in \mathcal{G} , there is no point in monitoring it nor in monitoring $E.r$ (which can only influence $A.r$ via $D.r$).

To cope with this we now define the \mathcal{P} -core of \mathcal{G} , which intuitively contains those role of \mathcal{G} which additionally do not fully depend on an untrusted role.

Definition 4.6 (\mathcal{P} -Core) Let \mathcal{P} be a state and \mathcal{G} be a set of roles. The \mathcal{P} -core of \mathcal{G} , $core_{\mathcal{P}}(\mathcal{G})$, is the maximal subset of \mathcal{G} such that

- If $A.r \leftarrow B.r_1 \in \mathcal{P}$, and $B.r_1 \notin core_{\mathcal{P}}(\mathcal{G})$, then $A.r \notin core_{\mathcal{P}}(\mathcal{G})$
- If $A.r \leftarrow A.r_1.r_2 \in \mathcal{P}$, and $A.r_1 \notin core_{\mathcal{P}}(\mathcal{G})$, then $A.r \notin core_{\mathcal{P}}(\mathcal{G})$.
- If $A.r \leftarrow A.r_1.r_2 \in \mathcal{P}$, and $\exists B \in \llbracket A.r_1 \rrbracket_{UB(\mathcal{P})}$ such that $B.r_2 \notin core_{\mathcal{P}}(\mathcal{G})$, then $A.r \notin core_{\mathcal{P}}(\mathcal{G})$.
- If $A.r \leftarrow A_1.r_1 \cap \dots \cap A_n.r_n \in \mathcal{P}$, and for every i , $A_i.r_i \notin core_{\mathcal{P}}(\mathcal{G})$, then $A.r \notin core_{\mathcal{P}}(\mathcal{G})$. \square

The following proposition is proved in [7].

Proposition 4.7 Let \mathcal{P} be a set of statements and \mathcal{G} be a set of roles.

- If $A.r \notin core_{\mathcal{P}}(\mathcal{G})$, then $\llbracket A.r \rrbracket_{UB(\mathcal{P})} = \text{Principals}(\mathcal{P}) \cup \{\top\}$. \square

We now construct the set of roles that must be monitored for new definitions to detect growth in a role's membership.

Definition 4.8 Let $A_0.r_0$ be a role in $core_{\mathcal{P}}(\mathcal{G})$, \mathcal{R} be a role monitor, and \mathcal{P} be a state; $\Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0) \subseteq \text{Roles}(\mathcal{P})$ is the least set satisfying the following:

- If $A_0.r_0 \in core_{\mathcal{P}}(\mathcal{G})$, $A_0.r_0 \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$.
- If $A.r \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$, and $A.r \leftarrow B.r_1 \in \mathcal{P}$, then $B.r_1 \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$.
- If $A.r \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$ and $A.r \leftarrow A.r_1.r_2 \in \mathcal{P}$, then $A.r_1 \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$ and $X.r_2 \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$ for all $X \in \llbracket A.r_1 \rrbracket_{UB(\mathcal{P})}$
- If $A.r \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$ and $A.r \leftarrow A_1.r_1 \cap \dots \cap A_n.r_n \in \mathcal{P}$, then, for each $i \in [1, n]$ if $A_i.r_i \in core_{\mathcal{P}}(\mathcal{G})$, $A_i.r_i \in \Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$. \square

It is easy to prove by a simple induction on the steps in the iterative construction of $\Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0)$ that $\Gamma_{\mathcal{P}}^{\mathcal{G}}(A_0.r_0) \subseteq core_{\mathcal{P}}(\mathcal{G})$

We now have the counterpart of Lemma 3.7.

Lemma 4.9 Assume $\top \notin \llbracket A.r \rrbracket_{UB(\mathcal{P})}$. Let \mathcal{R} be a role monitor, $\mathcal{P}' = \mathcal{P} \cup \{stmt\}$, where $head(stmt) \notin \Gamma_{\mathcal{P}}^{\mathcal{G}}(A.r)$, then

- (a) $\llbracket A.r \rrbracket_{UB(\mathcal{P}')} = \llbracket A.r \rrbracket_{UB(\mathcal{P})}$, and
- (b) $\Gamma_{\mathcal{P}'}^{\mathcal{G}}(A.r) = \Gamma_{\mathcal{P}}^{\mathcal{G}}(A.r)$.

Moreover, if \mathcal{P}' is obtained from \mathcal{P} by (a) adding zero or more statements whose head is not in $\Gamma_{\mathcal{P}}^{\mathcal{G}}(A.r)$, and (b) removing zero or more statements, then

$$(c) \llbracket A.r \rrbracket_{UB(\mathcal{P})} \supseteq \llbracket A.r \rrbracket_{UB(\mathcal{P}')} , \text{ and}$$

$$(d) \Gamma_{\mathcal{P}}^{\mathcal{G}}(A.r) \supseteq \Gamma_{\mathcal{P}'}^{\mathcal{G}}(A.r).$$

Proof (sketch). The result follows by using reasoning similar to that used for proving Lemma 3.7. \square

Negative Dependencies, with Untrusted Roles

To handle the right hand side of the constraints we simply have to generalize Lemma 3.11 in the obvious way by taking into account the presence of the role monitor. The proof of this lemma is also identical to that of Lemma 3.11

Lemma 4.10 Let $\mathcal{R} = (\mathcal{G}, \mathcal{S})$ be a role monitor, $A.r$ be a role, D be a principal, \mathcal{P} be a state and Σ be a \mathcal{P} -support of D for $A.r$ such that $\Sigma \subseteq \mathcal{S}$. Then

$$1. D \in \llbracket A.r \rrbracket_{LB(\mathcal{P})}$$

Moreover, if \mathcal{P}' is obtained from \mathcal{P} by (a) removing zero or more statements whose head is not in Σ , and (b) adding zero or more statements, then

$$2. \Sigma \text{ is a } \mathcal{P}'\text{-support for } A.r, \text{ and therefore}$$

$$3. D \in \llbracket A.r \rrbracket_{LB(\mathcal{P}')} . \quad \square$$

Recall that by Remark 4.4, if $A.r \notin \mathcal{S}$ then we have that $\llbracket A.r \rrbracket_{LB(\mathcal{P})} = \emptyset$. Consequently, it is easy to show that if $D \in \llbracket A.r \rrbracket_{LB(\mathcal{P})}$, then there exists a \mathcal{P} -support of D for $A.r$ consisting of roles that are in \mathcal{S} .

Putting Things Together

We can now prove the result we were aiming at. Differently from the case in which all roles were trusted, we now want to check that $\lambda \sqsubseteq \varrho$ holds in any \mathcal{R} -reachable state \mathcal{P}' . The additional problem here is we cannot rely on the cooperation of the roles that are not in \mathcal{G} (resp. \mathcal{S}) in monitoring the constraint and telling the constraint owner when a statement defining a role in $\Gamma_{\mathcal{P}}(\lambda)$ is added (resp. a statement defining a role in Σ is removed). Because of this we refer to two ‘‘pessimistic’’ semantics, $\llbracket \lambda \rrbracket_{UB(\mathcal{P})}$ and $\llbracket \varrho \rrbracket_{LB(\mathcal{P})}$, and we check if $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$. If this does not hold, then, by Theorem 4.5 the chance is high that in some reachable \mathcal{P}' the constraint is violated. If $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$ does hold, then we can apply the following:

Theorem 4.11 (Main with Untrusted Roles) Let $\mathcal{R} = (\mathcal{G}, \mathcal{S})$ be a role monitor. Assume that $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$. Let Σ be a \mathcal{P} -support of $\llbracket \lambda \rrbracket_{UB(\mathcal{P})}$ for ϱ such that $\Sigma \subseteq \mathcal{S}$, and let $\mathcal{P} \mapsto \mathcal{P}'$ be a (possibly multistep) change from \mathcal{P} to \mathcal{P}' . If

$$(i) \forall stmt \in \mathcal{P}' \setminus \mathcal{P}, \text{ head}(stmt) \notin \Gamma_{\mathcal{P}}^{\mathcal{G}}(\lambda), \text{ and}$$

$$(ii) \forall stmt \in \mathcal{P} \setminus \mathcal{P}', \text{ head}(stmt) \notin \Sigma$$

Then $\llbracket \lambda \rrbracket_{UB(\mathcal{P}')} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P}')} .$

Proof. Take any $D \in \llbracket \lambda \rrbracket_{UB(\mathcal{P}')} ,$ by Lemma 4.9, $D \in \llbracket \lambda \rrbracket_{UB(\mathcal{P})} .$ By assumption, $D \in \llbracket \varrho \rrbracket_{LB(\mathcal{P})} ,$ and by Lemma 4.10, $D \in \llbracket \varrho \rrbracket_{LB(\mathcal{P}')} .$ Hence the thesis. \square

Because of Theorem 4.11, in the presence of untrusted roles we can deploy a monitoring procedure very similar to that described after Theorem 3.12. First we check that $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$ holds². While doing this, we compute a \mathcal{P} -support Σ of $\llbracket \lambda \rrbracket_{UB(\mathcal{P})}$ for ϱ —this time a Σ such that $\Sigma \subseteq \mathcal{S}$. Second, we have to build $\Gamma_{\mathcal{P}}^{\mathcal{G}}(\lambda)$. Third, we monitor the roles in Σ and in $\Gamma_{\mathcal{P}}^{\mathcal{G}}(\lambda)$ so that each time a statement defining a role in $\Gamma_{\mathcal{P}}^{\mathcal{G}}(\lambda)$ (resp. Σ) is added to (resp. deleted from) \mathcal{P} , the constraint owner is warned. When the constraint owner receives a warning, he has to (a) check whether $\llbracket \lambda \rrbracket_{UB(\mathcal{P})} \subseteq \llbracket \varrho \rrbracket_{LB(\mathcal{P})}$ still holds, and (b) recompute $\Gamma_{\mathcal{P}}^{\mathcal{G}}(\lambda)$ and Σ .

Example 4.12 Reconsider again Example 3.3. Suppose that *Emergency.dept* is (the only role) not in \mathcal{G} , then we have that *Emergency.responsePersonnel* $\notin \text{core}_{\mathcal{P}}(\mathcal{G})$. Therefore

$$\Gamma_{\mathcal{P}}^{\mathcal{G}}(\text{Emergency.hazmatPersonnel}) = \left\{ \begin{array}{l} \text{Emergency.hazmatPersonnel}, \\ \text{ATF.hazmatTraining} \end{array} \right\}$$

Nonetheless, if *ATF.hazmatDB* $\in \mathcal{S}$ we have that

$$\begin{aligned} & \llbracket \text{Emergency.hazmatPersonnel} \rrbracket_{UB(\mathcal{P})} \\ & \subseteq \llbracket \text{ATF.hazmatDB} \rrbracket_{LB(\mathcal{P})} \end{aligned}$$

so by Theorem 4.5 we know that the constraint

$$\text{Emergency.hazmatPersonnel} \sqsubseteq \text{ATF.hazmatDB}$$

is satisfied in all reachable \mathcal{P}' . By Theorem 4.11, if the two roles *Emergency.hazmatPersonnel*, and *ATF.hazmatTraining*, prompt a warning when a statement defining one of them is added and the role *ATF.hazmatDB* gives a warning when one of its statement is removed, then the constraint needs to be re-checked only when a warning is given. In that case, we also have to recompute Σ and $\Gamma_{\mathcal{P}}^{\mathcal{G}}(\text{Emergency.hazmatPersonnel})$. Theorem 4.11 guarantees that no matter which changes are made to \mathcal{P} , until a warning is given, we still have that every reachable³ \mathcal{P}' satisfies the constraint. \square

5. RELATED WORK

In database theory, an integrity constraint is a query that must remain *true* after the database has been updated. Originally, integrity constraints were introduced to prevent incorrect updates and to check the database for integrity. Nevertheless, integrity constraints have later been used for a number of purposes, ranging from query optimization to view updating. We refer to [8, 5] for illustrative examples of the uses of integrity constraints in deductive databases.

In Section 2, we listed several papers presenting various trust management systems. None of these incorporates a notion of integrity constraints. The work in trust management that is most closely related is [12]. As we discussed at the beginning of Section 4, that work is complimentary to

²Even if this does not hold, when neither λ nor ϱ is static, it is possible that $\llbracket \lambda \rrbracket_{SP(\mathcal{P}')} \subseteq \llbracket \varrho \rrbracket_{SP(\mathcal{P}')} for all \mathcal{P}' reachable from \mathcal{P} . However, in general, for the class of constraints we consider, determining this is **PSPACE**-hard [12], i.e., intractable. Thus, our technique makes an efficient conservative approximation for the more general constraints we consider.$

³Notice that changing \mathcal{P} also changes the reachability relation, i.e., the set of reachable \mathcal{P}' s.

ours. It studies the problem of determining, given a state \mathcal{P} , a role monitor \mathcal{R} , and a constraint Q , whether there is a reachable state in which Q is violated. By contrast, we analyze the problem of which roles must have their definitions monitored to detect when such a \mathcal{P} is entered.

6. CONCLUSION

We introduce the use, monitoring, and enforcement of integrity constraints in trust management-style authorization systems. We consider the portions of the policy state that must be monitored to detect violations of integrity constraints. We also address the extra difficulty that not all participants in a trust management system can be trusted to assist in such monitoring, and show how many integrity constraints can be monitored in a conservative manner so that trusted participants detect and report if the system enters a policy state from which evolution in unmonitored portions of the policy could lead to a constraint violation.

7. ACKNOWLEDGMENTS

We thank Pieter Hartel and Ha Manh Tran for their precious help and the anonymous referees for their comments.

8. REFERENCES

- [1] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*, pages 185–210. Springer-Verlag, 1999.
- [3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In IEEE Computer Society Press, editor, *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
- [4] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.
- [5] S. K. Das. *Deductive Databases and Logic Programming*. Addison-Wesley, 1992.
- [6] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.
- [7] S. Etalle and W. H. Winsborough. Integrity constraints in trust management. Technical Report TR-CTIT-05-12, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, March 2005.
- [8] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity constraints: Semantics and applications. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 265–306. Kluwer Academic, 1998.
- [9] C. Gunter and T. Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, September 2000.
- [10] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society Press, May 2001.
- [11] N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 114–130. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press, 2002.
- [12] N. Li, J. Mitchell, and W. Winsborough. Beyond proof of compliance: Security analysis in trust management. *Journal of ACM*, 2004. To appear.
- [13] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In V. Dahl and P. Wadler, editors, *5th International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, volume 2562 of *LNCS*, pages 58–73. Springer-Verlag, 2003.
- [14] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.
- [15] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security (TISSEC)*, 6(1):128–171, February 2003.
- [16] R. Rivest and B. Lampson. SDSI — a simple distributed security infrastructure, October 1996. Available at <http://theory.lcs.mit.edu/~rivest/sdsi11.html>.
- [17] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.
- [18] S. Weeks. Understanding trust management systems. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, pages 94–105. IEEE Computer Society Press, 2001.