Interoperability of Relationship- and Role-Based Access Control

Syed Zain R. Rizvi Philip W. L. Fong
University of Calgary
Alberta, Canada
{szrrizvi, pwlfong}@ucalgary.ca

ABSTRACT

Relationship-Based Access Control (ReBAC) was recently proposed as a general-purpose, application-layer access control paradigm, such that authorization decisions are based on the relationship between the access requestor and the resource owner. A first, large-scale implementation of ReBAC in an open-source medical records system was recently attempted by Rizvi *et al.*

In this work, we extend the ReBAC model of Rizvi *et al.* to support fine-grained interoperability between the ReBAC model and legacy Role-Based Access Control (RBAC) models. This is achieved by the introduction of the notion of demarcations as well as an authorization-time constraint system. Also presented are the design of two authorization algorithms (one of which has an algorithmic structure akin to an SMT solver), their optimization via memoization, and the empirical evaluation of their performances.

CCS Concepts

•**Security and privacy** → **Access control;**

Keywords

Relationship-based access control, role-based access control, interoperability, demarcations, constraints, lazy evaluation, memoization.

1. INTRODUCTION

Access Control is a cornerstone of application security. For several decades, ***Role-Based Access Control (RBAC)*** [26] has been the dominant access control model for the application layer of computer systems. RBAC facilitates administration by having roles as an intermediary abstraction between users and privileges.

Recent authors have advocated the use of ***Relationship-Based Access Control (ReBAC)*** as a general-purpose access control paradigm [14, 16, 7, 10, 9, 15, 13, 24] for organizational applications. This is mainly motivated by the need for fine-grained authorization, such that access is not only determined by job functions (as in RBAC), but also by how the access requestor and the resource owner are related to one another within the organization (e.g., not all doctors can access, but only my family doctor may access). ReBAC policies also facilitate the expression of trust delegation (e.g., consultants referred by my family doctor may access) [14]. Electronic Health Records (EHR) systems are considered an archetypical application domain for deploying ReBAC. Rizvi *et al.* recently reported the first large-scale implementation of ReBAC in an open-source EHR system, OpenMRS [24].

Due to significant investment, RBAC is not going to disappear soon. Any extension of an existing software application to incorporate ReBAC must find a way for the new access control model to interoperate harmoniously with the legacy RBAC model. One path of least resistance is taken by Rizvi *et al.*'s implementation: an access request is granted when both the ReBAC and RBAC authorization mechanisms grant access. Otherwise, ReBAC and RBAC are essentially orthogonal to one another.

In this work, we argue that there are important reasons to allow ReBAC and RBAC to interact with one another in more fine-grained ways than in the work of Rizvi *et al.* In particular, there are important classes of access control policies that applications may need to support:

- **Privilege inheritance between roles and relationships.** For example, a family doctor (a relationship-based principal) is also a general practitioner, or GP (a role-based principal). Privileges granted to a GP shall also be granted to any family doctor. This is an example in which relationship-based principals inherit the privileges of role-based principals. The other direction (role-based principals inheriting from relationship-based principals) shall also be supported.
- **Mutual exclusion between roles and relationships.** When a requestor attempts to justify her access by a certain relationship, the security policy may not allow her to simultaneously justify her access using certain roles. For example, suppose a specialist is a role-based principal, and, as before, a family doctor is a relationship-based principal. If a clinician attempts to justify access by claiming she is both a specialist and a family doctor, then something suspicious maybe going on. One either acts as a specialist or a family doctor, but not both.
- **Qualification and refinement between roles and relationships.** In order to supervise a medical intern (a relationship-based principal), one needs to have the

CODASPY'16, March 9–11, 2016, New Orleans, LA, USA.

DOI: http://dx.doi.org/10.1145/2857705.2857706

qualification of a doctor (a role-based principal). In short, the principal supervisor is a refinement of the principal doctor.

In this work, we extend the ReBAC model of Rizvi *et al.* to support the above fine-grained interactions between roles and relationships. Our specific contributions are the following:

1. We introduce the notion of demarcations [20] into Rizvi *et al.*'s ReBAC model. This minor extension allows privilege inheritance to work across the boundary of ReBAC and RBAC (§2).
2. We propose an *authorization-time* constraint system that supports (a) using roles as qualification requirements of relationships (and vice versa), and (b) preventing the use of an incompatible pair of role and relationship to justify access (§3).
   Note that constraint enforcement is performed at the time of an authorization check. This stands in sharp contrast to static constraints in RBAC systems, in which constraints are enforced at the time when a session is initiated. Authorization time enforcement is necessitated by the fact that membership in relationship-based principals is established at the time of authorization (just like how it is done in UNIX [12]).
   At the time of authorization, the access control subsystem attempts to construct a rationale that justifies the requested access. This access justification consists of a set of role- and relationship-based principals. Our constraints essentially restrict how a legitimate access justification can be formed by the various principals.
3. As one would expect, constraint enforcement is computationally hard (NP-complete). To repeatedly verify constraint satisfaction for each authorization is, at least in theory, a computational challenge. We thus propose two authorization procedures, based respectively on eager and lazy evaluation, for enforcing constraints. The lazy authorization procedure has an algorithmic structure akin to an SMT solver: it uses a SAT solver to opportunistically discover computationally intensive subtasks, and uses learned clauses to constrain its search space (§4).
4. We propose three caching schemes to optimize the two authorization procedures above (§5).
5. We evaluate the performance of our authorization algorithms and caching schemes in a simulated environment (§6). A surprising result is that, even though in theory constraint enforcement is computationally hard, our authorization algorithms and caching strategies deliver competitive performance, thereby demonstrating that our scheme for ReBAC/RBAC interoperability is computationally feasible in practice.

2. ReBAC/DEMARCATIONS

We use the umbrella term ***ReBAC2015*** to refer to the family of access control models presented in this paper as well the one presented in Rizvi *et al.* [24]. We begin this section by reviewing the first member of this family: the ReBAC model of Rizvi *et al.* (§2.1). Then in §2.2 we present a minor extension of Rizvi *et al.*'s model to incorporate the notion of ***demarcations*** [20]. This is followed by a discussion of how the provision of demarcations enables RBAC and ReBAC to interoperate in interesting ways (§2.3). By the end of this section, we will have all the notational devices needed for defining the constraint system of §3.

2.1 A Review of Core ReBAC2015

Recent authors have advocated the application of ReBAC to domains other than social computing [14, 16, 7, 10, 9, 15, 13, 24]. A general-purpose model of ReBAC was proposed in [14], in which the protection state is a social network — an edge-labelled, directed graph in which vertices represent users, edges represent interpersonal relationships, and edge labels represent relationship types.[1] Access control policies have the form "*grant access to o if rp*," where o is the resource protected by the policy, and rp is a ***relationship predicate*** of the form $rp(G, u, v)$, returning a boolean authorization decision based on whether the requestor v is related to the owner u of resource o in a specific way within the social network G. For example, a relationship predicate that returns 1 whenever u and v are within a distance of 2 in G captures the friend-of-friend policy. An innovation of [14] is the use of modal logic as a policy language for specifying predicates of the form $rp(G, u, v)$. The modal policy language was later found to be limited in expressiveness [16]. ***Hybrid logic*** (an extension of modal logic to incorporate first-order features) was subsequently proposed as a policy language for ReBAC, to overcome the aforementioned limitations [7].

Crampton and Sellwood proposed another general-purpose ReBAC model [13]. A first innovation is that the protection state is no longer a social network, but an edge-labelled, directed graph with vertices representing both users and resources. This ***authorization graph*** therefore captures not only user-to-user relationships, but also resource-to-resource relationships as well as user-to-resource relationships (cf. [9]). Consequently, authorization decisions can be based on a much wider class of relationships among system entities. A second innovation is the notion of ***authorization principals***, which is a relationship-based analogue of roles originally inspired by UNIX [12]. In UNIX there are three fixed principals — owner, group and world — permissions are granted to principals, and principal membership is determined by how the requester is related to the requested resource (e.g., owner, same group, etc). In the ReBAC model of Crampton and Sellwood, the administrator may define an arbitrary set of principals. The authorization decision is a function of two sets of rules. Firstly, ***principal matching rules*** associate principals with relationship predicates. At the time of request, if the requestor and the resource satisfy the relationship predicate associated with a principal, then the requestor is considered a member of that principal. This contrasts with the static membership of roles in RBAC. Secondly, ***authorization rules*** assign permissions (either positive or negative) to principals. At request time, users beloning to a principal will be granted the permissions associated with that principal. Because of the presence of both positive and negative permissions, conflict resolution strategies are also considered. A third innovation is the employment of a regular expression-based policy language for specifying relationship predicates (cf. [10, 9]).

Rizvi *et al.* implemented ReBAC in an open source medical records system, OpenMRS [24]. The implemented ReBAC model incorporates the authorization graph and au-

[1]More precisely, the protection state is a number of social networks, one for each ***context***. We overlook this subtlety to simplify discussion.

thorization principals of Crampton and Sellwood, and the hybrid logic of Bruns *et al.* as the specification language for relationship predicates. The methods of OpenMRS are protected by privilege requirements called ***guards***. To accommodate these guards, Rizvi *et al.* invented two different semantics for principal matching: ***liberal-grant*** and ***strict-grant*** semantics. In the rest of this paper, we call this ReBAC model of Rizvi *et al.* ***Core ReBAC2015***.[2]

2.2 Enters Demarcations

The rest of this section presents ***ReBAC2015/Demarcations***, which is a minor extension of Core ReBAC2015 to incorporate the notion of demarcations [20]. The aims of this presentation are (a) to standardize notations for the rest of the paper, (b) to demonstrate how RBAC and ReBAC can interoperate in interesting ways due to the provision of demarcations, and (c) to pave the way to the introduction of a new model ***ReBAC2015/Constraints*** in §3. The description of his model is accompanied by a running example in order to demonstrate the purpose of each component.

A demarcation is an abstract grouping of privileges, just like a role or an authorization principal is an abstract grouping of users. Access control policies are expressed partly by associating authorization principals with demarcations. Such a scheme decouples the management of privilege grouping, user grouping, and user-group/privilege-group association.

DEFINITION 1. *The protection state of ReBAC2015/Demarcations has the following components.*

- O*: The set of objects (resources).*
- $S \subseteq O$*: The set of subjects (users).*
- I*: The set of relation identifiers (i.e., relationship types).*
- $G \subseteq I \times O \times O$*: The authorization graph, in which* $(i, u, v) \in G$ *refers to an edge from vertex* u *to vertex* v *that is labelled with the relation identifier* i*.*
- AP*: The set of authorization principals.*
- RP*: The set of relationship predicates supported by the system. A relationship predicate is a function with signature* $\mathcal{P}(I \times O \times O) \times O \times S \to \mathbb{B}$*. That is, a relationship predicate* $rp(G, o, s)$ *is a function that takes an authorization graph* G*, a resource* o *and a requestor* s *as inputs, and returns a boolean value as output.*
- $AR : AP \to RP$*: The assignment of a relationship predicate to each authorization principal (i.e., the principal matching rules).*
- DM*: The set of demarcations.*
- $\leq \subseteq DM \times DM$*: The demarcation hierarchy is a partial ordering over the set of demarcations (i.e., reflexive, transitive, anti-symmetric). Given two demarcations* $d_1, d_2 \in DM$ *for which* $d_1 \leq d_2$*, we say that* d_1 *is* **inferior***, and* d_2 *is* **superior***. Intuitively, superior demarcations "inherit" privileges from inferior demarcations. See Definition 8 for details.*
- $AD : AP \to DM$*: A principal-demarcation assignment (i.e., the authorization rules).*
- PR*: The set of privileges.*
- $PD \subseteq PR \times DM$*: The privilege-demarcation assignment relation.*
- MD*: A set of methods (actions) that can be applied to objects.*
- GD*: A set of guards (privilege requirements) for methods (to be further defined in Definition 4).*
- $MG : MD \to GD$*: An assignment of a guard to every method.*

It is assumed that O*,* I*,* AP*,* PR*,* DM *and* MD *are pairwise disjoint.*

DEFINITION 2. *An* **authorization request** *(or simply* **request***) is a triple* $(m, o, s) \in MD \times O \times S$*, whereby user* s *requests that method* m *be applied to resource* o*.*

EXAMPLE 3. *Alice, a doctor, wishes to read the health record of Bob, her patient. Suppose the health record is composed of identification information, demographic information, clinical visits history, and prescriptions history. Alice wishes to read these all together instead of reading each component individually. To do this, she invokes the method* read_hr(bob_hr) *where* bob_hr *is Bob's health record. This method invocation gets translated to the authorization request (*read_hr*,* bob_hr*, Alice).*

DEFINITION 4. *A* **guard** g *is a specification of privilege requirements that must be satisfied by the requestor in order for the authorization request to be granted. A guard* g *can have either of the two forms:*

1. `one-of(`ϕ`)`*, where* $\phi \subseteq PR$*, requires* one *of the privileges in* ϕ*.*
2. `all-of(`ϕ`)`*, where* $\phi \subseteq PR$*, demands* all *of the privileges in* ϕ*.*

Formally, the following rules specify when we say that a privilege set $\varphi \subseteq PR$ **satisfies** *a guard* g*, written* $\varphi \models g$*.*

$$\varphi \models \texttt{one-of}(\phi) \text{ iff } \phi \cap \varphi \neq \emptyset$$
$$\varphi \models \texttt{all-of}(\phi) \text{ iff } \phi \subseteq \varphi$$

EXAMPLE 5. *Once the authorization request of Example 3 is issued, the guard* $g_{\mathsf{read\_hr}} = MG(\mathsf{read\_hr})$ *is looked up. Since a health record is composed of four components (Example 3), a separate privilege is required to read each component, and thus* $g_{\mathsf{read\_hr}}$ *is an* `all-of` *guard:*

$$g_{\mathsf{read\_hr}} = \texttt{all-of}(\{\, r_{id\_info}, r_{dem\_info}, r_{cv\_history}, r_{pres\_history} \,\}).$$

As an abstract grouping of users, authorization principals are the ReBAC analogue of roles in RBAC. Membership in a principal, however, depends on the resource that is being accessed, and thus principal membership is determined at the time of request. This contrasts sharply with role membership in RBAC, which is statically defined.

DEFINITION 6 (PRINCIPAL MATCHING). *Given a request* (m, o, s)*, we say that an authorization principal* $ap \in AP$ *is* **enabled by** s **for** o *(or simply* **enabled***) whenever* $rp(G, o, s) = 1$*, where* $rp = AR(ap)$*. We write* $\mathsf{enabled}_o(s)$ *for the set of all authorization principals enabled by* s *for* o*:*

$$\mathsf{enabled}_o(s) = \{\, ap \in AP \mid AR(ap)(G, o, s) = 1 \,\} \quad (1)$$

EXAMPLE 7. *Continuing Example 5, we illustrate below how* $\mathsf{enabled}_{\mathsf{bob\_hr}}(Alice)$ *is computed.*

Suppose the system defines a principal FamDoc, for which $AR(FamDoc)$ *is a relationship predicate* rp *such that*

[2]Rizvi *et al.* also discuss a basic administrative model for ReBAC, but that is not relevant to this work.

$rp(G, o, s) = 1$ *whenever there is a directed edge with label* Family-Doctor *from the owner of* o *to the requestor* s*. Suppose further that Alice is indeed the family doctor of Bob, and thus such an edge exists. It follows that* $FamDoc \in \mathsf{enabled}_{bob\_hr}(Alice)$).

For subsequent examples, let us suppose the following:

$$\mathsf{enabled}_{bob\_hr}(Alice)) = \{\, FamDoc, GP, AuthUser \,\} \qquad (2)$$

Intuitively, Alice is a member of the principal GP because she works as a general practitioner in the hospital, and she is a member of the principal AuthUser because she is an authenticated user.

Definition 8 (Privilege Inheritance). *Suppose* $\alpha \subseteq AP$*. We define* $\mathsf{dem}(\alpha)$ *to be the set of demarcations associated with the authorization principals in* α*, and* $\mathsf{priv}(\alpha)$ *to be the set of privileges granted to the principals in* α*. Formally,*

$$\mathsf{dem}(\alpha) = \{\, d \in DM \mid \exists\, ap \in \alpha \,.\, d \leq AD(ap) \,\} \qquad (3)$$

$$\mathsf{priv}(\alpha) = \{\, p \in PR \mid \exists\, d \in \mathsf{dem}(\alpha) \,.\, (p, d) \in PD \,\} \qquad (4)$$

Note how superior demarcations inherit privileges from inferior ones.

Example 9. *By* (2), (3) *and* (4)*, we know the following:*

$$\mathsf{priv}(\mathsf{enabled}_{bob\_hr}(Alice)) = \mathsf{priv}(\{FamDoc\}) \cup \mathsf{priv}(\{GP\}) \cup \mathsf{priv}(\{AuthUser\})$$

Let us suppose that $\leq$*, AD and PD are defined such that the three principals are assigned the following privileges:*

$$\mathsf{priv}(\{FamDoc\}) = \{\, r_{id\_info}, r_{dem\_info}, r_{cv\_history}, r_{pres\_history}, \dots \,\}$$
$$\mathsf{priv}(\{GP\}) = \{\, r_{id\_info}, r_{dem\_info}, \dots \,\}$$
$$\mathsf{priv}(\{AuthUser\}) = \{\, \dots \,\}$$

We therefore have the following:

$$\mathsf{priv}(\mathsf{enabled}_{bob\_hr}(Alice)) = \{\, r_{id\_info}, r_{dem\_info}, r_{cv\_history}, r_{pres\_history}, \dots \,\}$$

Like Core ReBAC2015 (§2.1), ReBAC2015/Demarcations can be configured with one of two authorization semantics — liberal grant or strict grant. Intuitively, a request is authorized in the liberal-grant semantics whenever the set of *all* enabled principals jointly supply the privileges required by the guard. In strict-grant semantics, a request is authorized when there exists *one* enabled principal that can "single-handedly" satisfy the guard.

Definition 10 (Liberal Grant). *A request* (m, o, s) *is authorized according to the* **liberal-grant semantics***, denoted by the predicate* $\mathsf{auth}_{lib}(m, o, s)$*, iff the following holds:*

$$\mathsf{priv}(\mathsf{enabled}_o(s)) \models MG(m) \qquad (5)$$

Definition 11 (Strict Grant). *A request* (m, o, s) *is authorized according to the* **strict-grant** *semantics, denoted by the predicate* $\mathsf{auth}_{str}(m, o, s)$*, iff the following holds:*

$$\exists\, ap \in \mathsf{enabled}_o(s) \,.\, \Big(\mathsf{priv}(\{ap\}) \models MG(m)\Big) \qquad (6)$$

It can be shown that the two semantics induce identical authorization decisions for `one-of` guards. They differ in behaviour only for `all-of` guards [24, footnote 7]. Moreover, if a request is authorized in the strict-grant semantics, then it is also authorized in the liberal-grant semantics [24, footnote 8].[3]

Example 12. *If the system enforces liberal-grant semantics then the authorization decision will be based on* $\mathsf{priv}(\mathsf{enabled}_{bob\_hr}(Alice))$*, and access will be granted since:*

$$\mathsf{priv}(\mathsf{enabled}_{bob\_hr}(Alice)) \models g_{read\_hr}$$

On the other hand, if the system enforces strict-grant semantics, then access must be justifiable by a single principal. In this case, access is granted because $FamDoc \in \mathsf{enabled}_{bob\_hr}(Alice)$ *can "single-handedly" satisfy the guard:*

$$\mathsf{priv}(\{FamDoc\}) \models g_{read\_hr}$$

2.3 Interfacing ReBAC and RBAC

In Rizvi *et al.*'s implementation of ReBAC for OpenMRS, the ReBAC authorization scheme is layered on top of the legacy RBAC authorization scheme in an orthogonal manner: access is granted whenever both RBAC and ReBAC allow access [24]. Otherwise, the two authorization schemes do not interact with one another. Our extension of ReBAC with demarcations, as we shall see below, provides a clean mechanism for ReBAC2015 to interact with RBAC.

Embedding RBAC in ReBAC2015/Demarcations. We first describe how an RBAC protection state can be entirely embedded in a ReBAC2015/Demarcations protection state. Suppose an RBAC protection state is given: (R, RH, PA, UA). Here R is a set of roles. $RH \subseteq R \times R$, the role hierarchy, is a partial ordering such that $(r_1, r_2) \in RH$ means that r_1, a more senior role, dominates r_2, a more junior role, and thus r_1 inherits all privileges granted to r_2. $PA \subseteq PR \times R$ is the privilege assignment relation. $UA \subseteq S \times R$ is the user assignment relation. This RBAC protection state can be embedded in a ReBAC2015/Demarcations protection state using the following two-step construction.

1. *Role Predicates.* We define a principal ap_r for each role $r \in R$. The user assignment relation UA can be simulated by appropriately defined relationship predicates. Specifically, we formulate $AR(ap_r) = rp_r$ such that $rp_r(G, o, s) = 1$ whenever $(s, r) \in UA$. When formulated to capture role membership, a relationship predicate is also called a ***role predicate***. Otherwise, the relationship predicate is said to be ***proper***.
2. *Isometric Demarcations.* We create a demarcation d_r for each role $r \in R$, such that $AD(ap_r) = d_r$. Such an arrangement, in which each principal is paired with a distinct demarcation, is called ***isometric demarcations***. In addition, we define PD in such a way that $(p, d_r) \in PD$ whenever $(p, r) \in PA$. we define $\leq$ in such a way that $d_{r_1} \leq d_{r_2}$ iff $(r_2, r_1) \in RH$.

When ReBAC Interacts with RBAC. The more interesting scenario is when the ReBAC2015 protection state contains both principals defined by role predicates (a.k.a.

[3]There are two main differences between the present treatment of ReBAC2015/Demarcations and that of Core ReBAC2015 in [24]: (1) The treatment here is a mathematical formalization of the implementation-oriented description in [24]; (2) Demarcations are added into the model of [24].

role-based principals) as well as principals defined by proper relationship predicates (a.k.a. ***relationship-based principals***). Suppose FamDoc is a relationship-based principal, such that s belongs to FamDoc iff she is the family doctor of the owner of a patient record o. Say GP is a role-based principal, indicating that the requestor is a general practitioner. If $AD(\mathsf{GP}) \leq AD(\mathsf{FamDoc})$, then family doctors inherit the privileges granted to GPs. This illustrates how the demarcation hierarchy and authorization principals jointly support privilege inheritance across the boundary of ReBAC and RBAC.

3. REBAC2015/CONSTRAINTS

Comparing Definitions 10 and 11, one would notice that in both semantics, authorization is granted when a certain principal set $\alpha \subseteq \mathsf{enabled}_o(s)$ satisfies the condition below:

$$\mathsf{priv}(\alpha) \models MG(m) \tag{7}$$

For liberal-grant, α is $\mathsf{enabled}_o(s)$ itself; for strict-grant, α is constrained to be a singleton subset of $\mathsf{enabled}_o(s)$. To borrow RBAC terminology, the set α of enabled principals are "activated" to satisfy the guard of m. More precisely, the set α constitutes the justification of access.

Liberal- and strict-grant represent the two extreme ends of the design spectrum. Liberal-grant imposes no restriction on the choice of α, and thus α is chosen to be the maximal such set: i.e., $\mathsf{enabled}_o(s)$ itself. In strict-grant, it is required that privileges be contributed only by one principal, and thus α must be a singleton set. The intuition is that perhaps the set of privileges in `all-of` guards are so sensitive that access must be justified by a single principal capable enough to contribute all of them.

In some applications, however, restrictions on the selection of authorization principals for justifying access must be imposed according to the specific security needs of the application domain. For example, it may be the case that one authorization principal shall not be activated along with another authorization principal (mutual exclusion), or it may be the case that an authorization principal can only be activated in conjunction with another principal (prerequisite). Capturing these security restrictions in the form of a one-size-fits-all authorization semantics (e.g., liberal- or strict-grant) makes the access control model more rigid than necessary. In the following, we propose an extension of ReBAC2015 to incorporate the notion of authorization-time constraints, which specify restrictions over the choice of α, the set of principals to be activated to justify access. Note the difference between our authorization-time constraints and the traditional session-based constraints in RBAC. In the latter case, constraints such as static mutually exclusive roles are enforced at the time when a session is initiated. In our case, constraints are enforced every time when an access request is authorized, to ensure that a proper justification can be constructed for each authorized access.

3.1 The Extended Model

ReBAC2015/Constraints builds on ReBAC2015/Demarcations. Specifically, constraints are imposed over the constitution of the set α of principals in (7).

Definition 13. *The protection state of ReBAC2015/Constraints is made up of the same components as ReBAC2015/Demarcations (Definition 1), plus the following two additional components.*

- $\# \subseteq AP \times AP$*: The* **Mutually Exclusive Authorization Principal (MEAP)** *constraints form an irreflexive binary relation $\#$ over AP. (A binary relation R is irreflexive if xRx is never true.) Intuitively, if $ap_1 \# ap_2$, then either ap_1 or ap_2, but not both, can be activated to satisfy a guard.*
- $\sqsubseteq \subseteq AP \times AP$*: The* **Prerequisite Authorization Principal Requirement (PAPR)** *constraints form a partial ordering $\sqsubseteq$ over AP (i.e., reflexive, transitive and anti-symmetric). Intuitively, if $ap_1 \sqsubseteq ap_2$, then ap_2 cannot be activated unless ap_1 is also activated.*

Intuitively, a MEAP constraint $ap_1 \# ap_2$ prevents the two principals ap_1 and ap_2 from being used simultaneously for forming access justification. Therefore, MEAP constraints are therefore used for modelling incompatibilities among access rationales.

On the other hand, a PAPR constraint $ap_1 \sqsubseteq ap_2$ ensures that ap_2 is used for justifying an access only if ap_1 is also used for justifying the same access. In other words, to activate ap_2, the requestor s must satisfy not only the relationship predicate $AR(ap_2)$, but also $AR(ap_1)$. PAPR constraints model the refinement of principal membership.

Constraint semantics is formalized in the definition below.

Definition 14 (Constraint Satisfaction). *Given a set $\alpha \subseteq AP$, we write* $\mathsf{consistent}(\alpha)$*, and say that α is* **consistent***, iff the following conditions hold:*

$$\forall ap_1, ap_2 \in AP \,.\, \big(ap_1 \# ap_2 \Rightarrow (ap_1 \notin \alpha \vee ap_2 \notin \alpha)\big) \tag{8}$$

$$\forall ap_1, ap_2 \in AP \,.\, \big(ap_1 \sqsubseteq ap_2 \Rightarrow (ap_1 \in \alpha \vee ap_2 \notin \alpha)\big) \tag{9}$$

Definition 15. *A request (m, o, s) is authorized by ReBAC2015/Constraints, denoted by the predicate $\mathsf{auth}_{con}(m, o, s)$, iff the following holds:*

$$\exists \alpha \subseteq \mathsf{enabled}_o(s) \,.\, \mathsf{consistent}(\alpha) \wedge \big(\mathsf{priv}(\alpha) \models MG(m)\big) \tag{10}$$

The principals in the set α above are said to be **activated** *(to satisfy $MG(m)$).*

Theorem 17 below formally justifies our previous characterization of liberal- and strict-grant semantics as the two extreme ends of the design spectrum. It turns out that, ReBAC2015/Constraints can be configured to various points between the two ends of the spectrum.

Definition 16. *The protection state is said to be* **prerequisite-free** *iff $ap_1 \sqsubseteq ap_2$ implies $ap_1 = ap_2$. The protection state is* **conflict-free** *iff $\#$ is an empty relation (i.e., $ap_1 \# ap_2$ never holds). The protection state is* **conflict-complete** *iff $\#$ is maximal (i.e., $ap_1 \neq ap_2$ implies $ap_1 \# ap_2$).*

Theorem 17 (Expressiveness). *Suppose the protection state is prerequisite-free. Then:*

$$\mathsf{auth}_{str}(m, o, s) \Rightarrow \mathsf{auth}_{con}(m, o, s) \Rightarrow \mathsf{auth}_{lib}(m, o, s) \tag{11}$$

The first implication above is strengthened to a logical equivalence (i.e., $\mathsf{auth}_{str}(m, o, s) \Leftrightarrow \mathsf{auth}_{con}(m, o, s)$) when the protection state is conflict-complete. The second implication is strengthened to a logical equivalence (i.e., $\mathsf{auth}_{lib}(m, o, s) \Leftrightarrow \mathsf{auth}_{con}(m, o, s)$) when the protection state is conflict-free. In short, liberal- and strict-grant semantics are but two instantiations of ReBAC2015/Constraints.

The theorem follows directly from the definition.

3.2 Interfacing ReBAC and RBAC

The provision of MEAP and PAPR constraints allows one to express interesting interactions between ReBAC and RBAC. The following are some possibilities:

- Consider the role-based principal Specialist and the relationship-based principal FamDoc. Imposing the MEAP constraint Specialist # FamDoc ensures that the requestor acts as either a specialist or a family doctor, but not both. This constraint prevents inconsistent explanation of access rationale.
- Consider the role-based principal Doctor and the relationship-based principal Supervisor. Imposing the PAPR constraint Doctor $\sqsubseteq$ Supervisor ensures that only doctors can supervise medical interns, thus capturing the qualification requirement for a relationship.

4. ENFORCEMENT MECHANISMS

In ReBAC2015/Constraints, authorization involves the testing of condition (10). Two authorization algorithms, based respectively on the eager and lazy evaluation of relationship predicates, are presented in this section for testing (10). Comparison of their performance is deferred to §6.

4.1 Authorization via Eager Evaluation

Given an authorization request (m, o, s), a straightforward procedure for computing authorization decisions operationalizes (10) into the following two steps:

1. Compute the principal set $E = \mathsf{enabled}_o(s)$.
2. Decide if there exists $\alpha \subseteq E$ such that:

$$\mathsf{consistent}(\alpha) \wedge \big(\mathsf{priv}(\alpha) \models MG(m)\big) \qquad (12)$$

Step 1 involves going through each principal $ap \in AP$, and invoking a model checker to evaluate $AR(ap)(G, o, s)$ (assuming that relationship predicates are specified in hybrid logic [7]). Since multiple principals may share the same relationship predicate, the implementation may cache and reuse the result of evaluating $AR(ap)$ to prevent duplicated computations. Specifically, an empty cache is initialized every time Step 1 begins. Whenever a relationship predicate rp is evaluated to a Boolean value b, the pair $\langle rp, b\rangle$ is cached, so that next time the value of rp is needed, a cache lookup is performed. This optimization is called ***RP*-caching**. Three different caching schemes will be discussed in §5.

Step 2 is a computationally hard problem:

The ReBAC2015/Constraints Authorization Problem (Re-Auth)
Instance: A protection state (Def. 13), a set $E \subseteq AP$, and a method $m \in MD$.
Question: Does there exist $\alpha \subseteq E$ such that (12) holds?

Theorem 18. Re-Auth *is NP-complete.*

A proof of Thm 18 is given in Appendix A, which also shows that the MEAP constraints (#) are the source of complexity.

Step 2 can be implemented by a reduction of the Re-Auth instance to a SAT instance, and using a SAT solver to solve the latter. Such a reduction is described in §4.3. Despite the NP-completeness result, performance profiling tells us that modern SAT solvers (such as [2]) can cope with Step 2 gracefully, because (a) the number of principals is only moderate in magnitude, and (b) most of the naturally-occurring Re-Auth instances are easy.

Algorithm 1 Authorization Checking via Lazy Evaluation

```
Input: (m, o, s)
 1: F ← getSATInstance(AP, m)
 2: satInitSolver(F)
 3: initialize E[·] to an empty map
 4: σ ← satNextModel()
 5: if satNextModel() fails then
 6:    return "deny access"
 7: end if
 8: α ← extractPrincipals(σ)
 9: for all ap ∈ α do
10:    rp ← AR(ap)
11:    if E[rp] is undefined then
12:       E[rp] ← rp(G, o, s)
13:    end if
14:    if E[rp] = 0 then
15:       satAddClause((¬ap))
16:       goto 4
17:    end if
18: end for
19: return "allow access"
```

4.2 Authorization via Lazy Evaluation

From the experiences reported in [24, §10], a significant computational challenge lies within Step 1 above, in which the relationship predicate of each authorization principal is evaluated to determine if the principal is enabled. The hybrid logic model checker [22] must be invoked over an authorization graph of potentially formidable size. The process involves both backtracking as well as queries against very large databases. This motivates an alternative enforcement mechanism that employs a lazy strategy in discovering whether principals are enabled. This contrasts sharply with the previous authorization procedure, which eagerly evaluates the relationship predicates of all principals.

Our lazy strategy employs a SAT solver to enumerate all possible principal sets $\alpha \subseteq AP$ for which $\mathsf{priv}(\alpha) \models MG(m)$. For each α, the relationship predicates of principals in α are then model checked. Again, predicate value caching is applied (§4.1). Consequently, model checking is conducted only when it is needed.

Modern SAT solvers (e.g., [2]) offer two features that make this strategy possible. First, users may iterate through the models of a formula. Second, while iterating through the models, the user may add additional clauses to the formula in order to forbid certain models from being generated in subsequent iterations. This second feature is exploited by the lazy strategy, to exclude from future consideration those principals that have been found to be disabled. This effectively cuts down the number of models that need to be examined. Consequently, the algorithmic structure of our authorization procedure is akin to the structure of a modern SMT solver, in which a SAT solver drives the invocation of a theory solver, and learns new clauses to guide backtracking.

Algorithm 1 presents the lazy authorization procedure in the form of pseudocode. A line-by-line explanation of Algorithm 1 is provided in Appendix B. This particular incarnation of lazy authorization incorporates *RP*-caching (§4.1). Other caching schemes are discussed in §5.

4.3 Reducing RE-AUTH to SAT

We describe here a reduction of RE-AUTH to a CNF-SAT instance, so that a standard SAT solver can be employed to solve the problem. We start by fixing a data representation for the PAPR constraints and the demarcation hierarchy.

DEFINITION 19. *Suppose $ap_1 \sqsubseteq ap_2$ but $ap_1 \neq ap_2$. Then we write $ap_1 \sqsubset ap_2$. Suppose further, for every $ap \in AP$ for which $ap_1 \sqsubseteq ap \sqsubseteq ap_2$, either $ap = ap_1$ or $ap = ap_2$. Then we write $ap_1 \sqsubset' ap_2$.*

Similarly, we can define $<$ and $<'$ for the demarcation hierarchy ($\leq$).

We assume that $\sqsubset'$ and $<'$ are available to the reduction algorithm. The algorithm would run correctly even if $\sqsubset$ and $<$ are used in place of $\sqsubset'$ and $<'$, but the CNF generated by the reduction would not be as compact.

Our reduction produces a CNF formula F with $|E|+|DM|$ variables: one for each principal in E, and one for each demarcation in DM. As $E \subseteq AP$, the number of variables is bounded by $|AP| + |DM|$. To simplify our notation, we write ap_i to denote the variable corresponding to principal ap_i, and d_j to denote the variable representing demarcation d_j. By construction, if the formula F is satisfied by a truth assignment σ, then:

- Let α be the set of principals $ap \in E$ for which $\sigma(ap) = 1$, and let δ be the set of demarcations $d \in DM$ for which $\sigma(d) = 1$.
- We know that $\delta = \mathsf{dem}(\alpha)$ and α satisfies (12).

Thus, both the activated principals and their corresponding demarcations can be recovered from a model of F.

Given a request (m, o, s), the formula F produced by the reduction has the following form:

$$F = F_{guard} \wedge F_{MEAP} \wedge F_{PAPR} \wedge F_{DH} \tag{13}$$

Each of F_{guard}, F_{MEAP}, F_{PAPR} and F_{DH} is a CNF formula (i.e., a set of clauses). F_{guard} encodes the guard of the method involved in the request. The other three encode the protection state. We describe their construction in turn.

Construction of F_{guard}. This formula contains clauses that encode the privilege requirements of method m. Let g be the guard $MG(m)$. Then g can be in either one of two forms: `one-of(`ϕ`)` or `all-of(`ϕ`)`.

Case g = `one-of(`ϕ`)`: Let $\delta(\phi) = \{\, d \in DM \mid \exists p \in \phi \,.\, (p, d) \in PD \,\}$, the set of demarcations that offer at least one privilege in ϕ. The clause $\left(\bigvee_{d \in \delta(\phi)} d\right)$ is added into F_{guard}. This ensures that at least one demarcation that provides at least one of the privileges in ϕ is granted.

Case g = `all-of(`ϕ`)`: Let $\delta(p) = \{\, d \in DM \mid (p, d) \in PD \,\}$, the set of demarcations that offers p. For each $p \in \phi$, the clause $\left(\bigvee_{d \in \delta(p)} d\right)$ is added into F_{guard}. This ensures every privilege in ϕ is provided by a granted demarcation.

Construction of F_{MEAP}. The clauses in F_{MEAP} enforce the MEAP constraints (#). For every (unordered) pair of principals $ap_1, ap_2 \in E$ such that $ap_1 \# ap_2$, the clause $(\neg ap_1 \vee \neg ap_2)$ is added into F_{MEAP}. This ensures that ap_1 and ap_2 are not both activated.

Construction of F_{PAPR}. The clauses in F_{PAPR} enforce PAPR constraints ($\sqsubseteq$). These clauses are of two kinds.

First, for each pair of principals $ap_1, ap_2 \in E$ for which $ap_1 \sqsubset' ap_2$, the clause $(ap_1 \vee \neg ap_2)$ is added into F_{PAPR}. This ensures that ap_2 is not activated unless ap_1 is activated.

Second, for each principal $ap_2 \in E$ for which there is a principal $ap_1 \in AP \setminus E$ such that $ap_1 \sqsubset' ap_2$, the unit clause $(\neg ap_2)$ is added into F_{PAPR}. This ensures that principals with prerequisites outside of E are not activated.

Construction of F_{DH}. This formula models the demarcation hierarchy ($\leq$) and the authorization rules (AD). For each $d \in DM$, a clause C_d is added into F_{DH}. The clause C_d has the form

$$\left(\neg d \vee \left(\bigvee_{ap \in \alpha} ap\right) \vee \left(\bigvee_{d \in \delta} d\right)\right) \tag{14}$$

where $\alpha = \{\, ap \in E \mid AD(ap) = d \,\}$ and $\delta = \{\, d' \in DM \mid d <' d' \,\}$. This clause ensures that the demarcation d is granted only if either one of the principals associated with it is activated, or one of its immediate superior demarcations is granted. Note that, if α and δ are both empty, C_d becomes the unit clause $(\neg d)$, and d will not be activated at all.

Additional implementation strategies for generating and storing these SAT instances are described in Appendix C.

5. CACHING STRATEGIES

Preliminary tests revealed that evaluating relationship predicates by a model checker is computationally more demanding than solving RE-AUTH instances using a SAT solver, and thus reducing the amount of work done by the model checker can potentially optimize the enforcement mechanism. This can be achieved by caching the results of computations previously performed by the model checker and reusing those results when possible. This section presents three caching strategies that can potentially improve the performance of the authorization procedure. Comparison of their performance is deferred to §6.2.

Note that these caching strategies do not cache authorization decions across authorization requests. The purpose of these caching strategies is to facilitate the computation of a single authorization decision.

*AP-**Caching**.* The idea behind this caching strategy is to prevent repeating principal membership tests. During the authorization of a request (m, o, s), we ensure that $AR(ap)(G, o, s)$ is computed at most once for each ap, and the result is cached. If membership in ap needs to be checked again during the authorization of the same request, then the cached result is reused. This caching strategy is not useful for the eager evaluation strategy, as membership in each principal is checked only once. AP-caching, however, is effective for the lazy evaluation strategy. Recall that the lazy evaluation strategy repeatedly generates sets $\alpha \subseteq AP$ for which $\mathsf{consistent}(\alpha) \wedge \big(\mathsf{priv}(\alpha) \models MG(m)\big)$. For each such α, it tests if $\alpha \subseteq \mathsf{enabled}_o(s)$ by making an invocation of the model checker for each $ap \in \alpha$. Now, suppose α_1 and α_2 are two candidate principal sets generated by the lazy authorization procedure. Then membership for those principals in $\alpha_1 \cap \alpha_2$ will be tested twice. AP-caching is designed to optimize away such repeated membership tests.

*RP-**Caching**.* The idea behind this caching strategy is to prevent computing $rp(G, o, s)$ more than once, for any given $rp \in RP$. During the authorization of a request (m, o, s), suppose membership in principal ap is to be tested. Let $rp = AR(ap)$. The model checker is invoked to compute $b = rp(G, o, s)$. The pair $\langle rp, b \rangle$ is then cached. When membership of another principal ap' is to be tested later during the authorization of the same request, if a pair $\langle AR(ap'), b' \rangle$

Parameter	Values
$\|PR\|$	$\{3 \times \|AP\|\}$
$\|PD\|$	$\{7 \times \|AP\|\}$
$\|\phi\|$	$\{3\}$
$\|AP\|$	{50, 100, 150, 200}
$\|\#\|$	{50, 100, 150, ..., 500}
$\mid \sqsubset' \mid$	{50, 100, 150, ..., 500}
$\mid <' \mid$	{50, 100, 150, ..., 500}

Table 1: Parameters used for test cases.

is found in the cache, the value b' is reused rather than computed from scratch. This caching strategy utilizes the fact that multiple authorization principals may share a common relationship predicate. This reduces the number of times the model checker is invoked during an authorization check, for both the eager and lazy evaluation strategy. More specifically, the model checker is invoked at most $|RP'|$ times for an authorization check, where RP' is the range of the function AR on input AP. It should be simple to observe that if each authorization principal has a distinct relationship predicate (i.e. $|AP| = |RP'|$), then RP-caching behaves the same as AP-caching. In general, AP-caching is no more effective than RP-caching.

MC-***Cross-Caching***. This third caching scheme exploits the memoization mechanism of the hybrid logic model checker in a creative way. The model checker in the ReBAC Java Library implements the inductively specified semantics of Bruns *et al.* [7]. As such, the model checker has a recursive structure. To prevent pathological cases and to speed up model checking, the model checking procedure caches the return values of recursive calls — a technique commonly known as ***memoization*** in functional programming literature [28, §19.6]. The cache is local to the model checker instance. In the implementation of AP- and RP-caching, every time model checking is conducted, a new model checker instance is created. Consequently, the model checker cache is discarded after each test of principal membership.

In MC-cross-caching, only one instance of the model checker is created for a given authorization check. The same model checker instance is reused for all principal membership tests during this authorization check. This means that, unlike AP- and RP-caching, in which the values of relationship predicates are cached, MC-cross-caching allows the values of the *subformulas* of the relationship predicates to be reused.

In general, RP-caching is no more effective than MC-cross-caching. They behave identically only when relationship predicates do not share common subformulas.

6. PERFORMANCE EVALUATION

An empirical study has been conducted to evaluate and compare the performance of the eager and lazy authorization procedures (§4) as well as the caching schemes (§5). The study was conducted in a simulated environment with synthetic data. Below we describe our experimental set-up followed by the results of the experiments comparing the authorization procedures (§6.1) and the experiments comparing the caching schemes (§6.2).

Core ReBAC2015 Components. We reused part of the ReBAC protection state in the work of Rizvi *et al.* [24]. First, we reused their authorization graph, which consists of 1.6 million nodes and 32 million edges. It was constructed from the social network dataset, soc-Pokec, from the Stanford Large Network Dataset Collection [3]. 10,000 of the nodes were taken as users (S), and the rest patients ($O \setminus S$). Consult [24, §10 and Appendix B] for details such as edge labelling. Second, we reused their values for $|PR|$, $|PD|$ and $|\phi|$ (Table 1, first 3 lines). Explanations of how these parameter settings are deduced from previously published experimental works on RBAC can be found in [24, §10 and Appendix A]. Third, we reused the hybrid logic formulas in [24, Appendix C] as relationship predicates. These formulas were adapted from the ones used in the Electronic Health Records System case study in [14, §5]. There is a total of 10 distinct formulas, and they are randomly assigned to authorization principals in the experiments (via a randomly generated AR). Consequently, some principals may share the same formulas.

Test Generation Scheme. Table 1 summarizes the parameter combinations used in our experiments. The last 4 lines of the table specify ranges of values for $|AP|$, $|\#|$, $\mid \sqsubset' \mid$ and $\mid <' \mid$, and account for 4,000 distinct parameter combinations. For each of the four experimental profiles, 20 test cases are randomly generated for each parameter combination. This amounts to 80,000 test cases for each profile.

To generate a test case, the parameter values are first fixed, and then the principals (AP), the principal matching rules (AR), isometric demarcations (DM for which $|DM| = |AP|$), privileges (PR), privilege-demarcation assignment (PD), the demarcation hierarchy ($<'$), and the constraints ($\#$ and $\sqsubset'$) are randomly generated in turn. Lastly, a triple consisting of an object ($o \in O$), a subject ($s \in S$) and a guard ($g \in GD$) is randomly generated to represent the authorization request of the test case.

Experimental Profiles. Four experimental profiles were generated, each is described below.

Profile 1. As discussed above, 80,000 test cases were generated for this profile. The 80,000 corresponding Re-Auth instances (with $E = AP$) were reduced to SAT instances and solved by a SAT solver. The running time of the SAT solver was recorded for each test case. The 500 test cases with the highest SAT solver running time were then selected. The eager and lazy authorization procedures were then fed these 500 test cases, and their running times were recorded.

This profile is designed to highlight the effect of hard Re-Auth instances on the two authorization procedures.

Since there were only 10 distinct hybrid logic formulas, and they were randomly assigned to 50–200 principals. This level of duplication is not representative. We therefore adopt RP-caching for this profile, but force the caching mechanism to treat the relationship predicate of each principal to be unique. This effectively turns off the effect of caching.

Profile 2. Again, 80,000 test cases were generated and fed to the lazy authorization procedure. The 500 test cases with the highest running time were selected. The two authorization procedures were then given these 500 test cases, and their running times were recorded. This profile compares the performance of the two authorization procedures on the test cases that are the hardest for lazy evaluation.

As in Profile 1, the hybrid logic formula of each principal was considered unique by the RP-caching mechanism.

Profile 3. This profile is similar to Profile 2, except that the 500 toughest test cases for the eager authorization procedure were selected.

Profile 4. This profile reuses the 500 test cases of Profile 1, but the relationship predicate of each principal is not

forced to be unique: i.e., RP-caching is in full effect. Recall that the 10 hybrid logic formulas inherited from [24] were randomly distributed among the authorization principals. With RP-caching, we expect the overhead of model checking to diminish significantly.

Hardware Configuration. The experiments were conducted on desktop machine with AMD FX-8350 8-core processor (16 MB cache), 16 GB RAM (1866 MHz, DDR3), and 840 EVO solid state drive running Windows 8.1.

Software Configuration. The eager and lazy authorization procedures were implemented in Java 7, with the hybrid logic model checker supplied by the open-source ReBAC Java Library [22]. We did not incorporate the optimization described in Appendix C, and thus all the timings recorded include the overhead of computing the reduction to SAT. We used Sat4j [2] as the SAT solver in the two authorization procedures. This SAT solver offers the two features mentioned in §4.2: model enumeration and learnt clauses. Following [24], the authorization graph was stored in a graph database, Neo4j [1], to obtain a more competitive performance than traditional relational databases.

6.1 Comparing Authorization Procedures

Fig. 1 depicts the average running times of the two authorization procedures in the four experimental profiles.

Finding 1: Re-Auth ***is not the bottleneck.*** The performance of both authorization procedures are competitive in Profile 1, indicating that the hardest Re-Auth instances do not slow down performance. Despite the need to solve an NP-complete problem at authorization time, ReBAC2015/Constraints exhibits competitive performance.

Finding 2: Lazy Evaluation is more efficient when model checking is expensive. The performance of lazy evaluation is significantly faster than that of eager evaluation in Profiles 2 and 3. This performance advantage remains whether the test cases are hard for lazy or eager evaluation.

To better understand this phenomenon, we examine the internal behaviour of the two authorization procedures in Table 2 (see the statistics for Profile 3). Eager evaluation makes many more invocations to the model checker (MC) than lazy evaluation. In contrast, eager evaluation makes only (at most) 1 invocation to the SAT solver, while lazy evaluation makes multiple invocations. Since the average time of a model checker invocation is much higher than that of the SAT solver, model checking time dominates the running time. (Note that reduction time is negligible in both authorization procedures.)

Rizvi *et al.* pointed out a major bottleneck of model checking large authorization graphs is database access [24]. We anticipate the performance advantage of lazy evaluation will become even more prominent when, for example, storing the authorization graph in an SSD is not possible due to hardware constraints, or the authorization graph must be stored in a legacy relational database rather than a graph database.

Finding 3: Eager evaluation has a performance advantage when model checking cost is low. In Profile 4, the performances of both lazy and eager evaluation are highly competitive, with eager evaluation exhibiting a performance advantage. In Profile 4, only 10 distinct relationship predicates are involved, and thus the benefit of predicate value caching becomes exceptionally prominent (Table 2, Profile 4). Such a performance advantage can be expected when the diversity of relationship predicates are low among the principals, or when the nesting depth of modal operators in hybrid logic formulas is small.

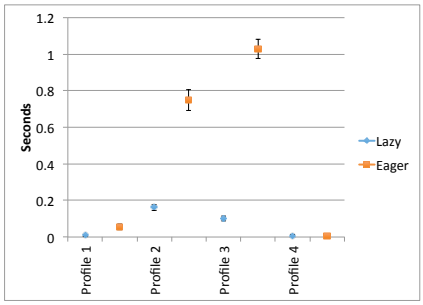

Figure 1: Average time of authorization checks (with 95% confidence interval).

Profile 3		
Avg. Stat.	**Lazy**	**Eager**
# MC calls	13.806	161.206
Time / MC call	0.007 secs	0.007 secs
# SAT calls	13.758	0.994
Time / SAT call	6.6×10^{-7} secs	1.1×10^{-4} secs
Reduction time	4.8×10^{-4} secs	4.9×10^{-4} secs
Profile 4		
Avg. Stat.	**Lazy**	**Eager**
# MC calls	3.838	9.976

Table 2: Statistics of Profiles 3 and 4

6.2 Comparing Caching Schemes

We generated scenarios that potentially challenge the caching strategies (§5). One such scenario would be to generate relationship predicates that do not share common subformulas. A limitation of this scenario is that it does not effect AP- and RP-caching since these strategies are not concerned with the subformulas of the relationship predicates. The unique subformulas scenario potentially challenges MC-cross-caching by preventing cached results from being used across different invocations of the model checker. This scenario is discussed in [23, Chapter 9].

This section presents an extreme scenario where the cost of model checking is extremely high. This scenario is achieved using the test cases of Profile 1 and prepending each relationship predicate with the prefix:

$$\langle \mathsf{agent} \rangle \langle \mathsf{agent} \rangle \langle \mathsf{agent} \rangle$$

This results in an increased workload for the model checker. The test cases of Profile 1 were used to keep the results from being biased towards either of the authorization procedures.

A secondary purpose of this test is to stress-test the MC-cross-caching strategy by forcing it to cache a high amount of data such that it results in the cache being full and thus forcing a data dump. The prefix causes the cache to be filled faster and potentially forcing invocations of the cache data dump, thereby countering the benefit of MC-cross-caching.

The average running time of the two evaluation schemes using the various caching strategies are summarized in Fig. 2.

Finding 4: As the depth of $\langle \cdot \rangle$ increases, the performance advantage of RP-caching and MC-cross-caching becomes highly pronounced, especially in the case of eager evaluation. Recall that the authorization graph consists of 30,622,564 edges between 1,632,803 nodes. For the

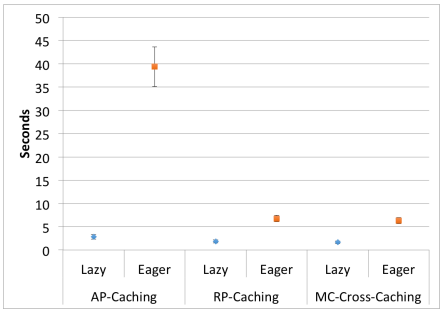

Figure 2: Average time of authorization checks using the caching strategies (w/ 95% CI).

construction of the authorization graph, Rizvi *et al.* separated the top 10,000 nodes with the highest in-degrees and labeled them as users, and the rest were labeled as patients. As a result of their construction, there are 28,538,682 patient-to-patient edges with the agent label, which means that there are approximately 18 (actually 17.586) outgoing agent edges per patient on average. The prefix used for the relationship predicates for this case results in an increase in the workload of the model checker. In the average worst case the workload is increased by a magnitude of 18^3 (= 5,832).

The results show significant performance advantage for using RP-caching or MC-cross-caching compared to AP-caching. The most prominent results are from the eager evaluation scheme. More specifically, the result of RP-caching with lazy evaluation or MC-cross-caching with lazy evaluation offer the best performance in the worst case scenario.

Finding 5: The MC-Cross-Caching does not invoke a data dump for this case. A surprising result here is that even at this depth, the cache used by the model checker does not invoke a single data dump for either the lazy or eager evaluation schemes. The cache used for the model checker for these experiments is the one provided by the ReBAC Java Library [22], which has a maximum size of 8,000,000 nodes. Since there is no data dump for the cache, all of its nodes are retained and already computed data is not lost. The relationship predicates can be further altered to force even more data to be stored within the model checker cache. For instance, one could use the $\downarrow x$ hybrid logic operator to bind each node to a variable and create the following prefix:

$$\langle \mathsf{agent} \rangle \downarrow x \ \langle \mathsf{agent} \rangle \downarrow y \ \langle \mathsf{agent} \rangle \downarrow z$$

Using the modified prefix would certainly result in a performance slowdown even more so than the original prefix. However, the performance using the original prefix already results in unacceptable running times and thus slowing it down further would not provide any benefits.

7. RELATED WORKS

The evolution of Fong's ReBAC model [14] to the Core ReBAC2015 model [24] has already been discussed in details in §2.1. Compared to [24], this work introduces the demarcation hierarchy (ReBAC2015/Demarcations) and an authorization-time constraint system (ReBAC2015/Constraints). These extensions increase the expressiveness of the model: the liberal- and strict-grant semantics of [24] are but two instantiations of our new model (Thm. 17). Furthermore, the new features support fine-grained interactions between RBAC and ReBAC (§2.3 and §3.2). This compares favourably with [24], in which ReBAC and the legacy RBAC models operate independently. Eager and lazy versions of principal matching algorithms are also presented in [24], but they cannot accommodate an authorization-time constraint system. This gap motivates our design of authorization checking algorithms in §4, which solve an NP-complete problem using a SAT solver, and our lazy authorization algorithm has an algorithmic structure akin to an SMT solver.

Independent of [13], Cheng *et al.* also considered the incorporation of user-to-resource and resource-to-resource relationships into the ReBAC protection state [9]. While relationship predicates are assumed to be specified in hybrid logic formulas [7] in this work, Cheng *et al.* [10, 9] and Crampton and Sellwood [13] advocate the use of regular expressions to specify relationship predicates. The two specification paradigms are incomparable in expressive power. Since backtracking is still involved in the checking of relationship predicates that are specified in regular expressions, our proposal of lazy evaluation would still apply. Lastly, the presence of negative permissions in [9, 13] necessitates conflict resolution schemes. We do not consider negative permissions in this work.

The constraint system presented in this paper stands on the shoulders of prior literature on RBAC constraints [21, 8, 26, 11, 5, 6, 4, 17, 18, 19, 27, 29]. For example, our MEAP constraints are direct descendants of mutual exclusion constraints, which are used to enforce separation of duty policies [25, 21]. Our work is unique in two ways. First, while mutual exclusion constraints (e.g., the SMER constraints in [21]) are imposed statically in a typical RBAC system (i.e., when a session is initiated), our MEAP and PAPR constraints are imposed dynamically (i.e., at the time of a ReBAC authorization check). This dynamism is necessitated by the fact that membership in authorization principals is defined dynamically. But then constraint solving is NP-complete, and there is no *a priori* reason to believe that authorization-time constraint enforcement is feasible. Thus the second uniqueness of this work is the demonstration of the surprising result that modern SAT solving technologies can be employed in a creative way (via our lazy evaluation strategy) to render authorization-time constraint enforcement a feasible task.

8. CONCLUSION AND FUTURE WORK

By introducing demarcations and constraints to the recently proposed ReBAC model of [24], we allow ReBAC and RBAC to interoperate in interesting ways. Constraints are checked at authorization time, via a lazy evaluation strategy akin to SMT solving. The performance of the authorization procedure have been shown empirically to be competitive.

A number of research opportunities arise from this work. First, one could consider negative permissions and study their impact on the constraint system. Second, one could extend ReBAC2015 to incorporate a richer guard language for specifying complex privilege requirements.

Acknowledgments

This work is supported in part by an NSERC Discovery Grant (RGPIN-2014-06611) and a Canada Research Chair (950-229712).

9. REFERENCES

[1] Neo4J. http://neo4j.com/.

[2] Sat4j. http://www.sat4j.org/.

[3] Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[4] Ahn, G.-J. *The RCL 2000 Language for Specifying Role-based Authorization Constraints*. PhD thesis, Fairfax, VA, USA, 2000. AAI9951117.

[5] Ahn, G.-J., and Sandhu, R. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the Fourth ACM Workshop on Role-based Access Control (RBAC'99)* (New York, NY, USA, 1999), ACM, pp. 43–54.

[6] Ahn, G.-J., and Sandhu, R. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur. 3*, 4 (Nov. 2000), 207–226.

[7] Bruns, G., Fong, P. W. L., Siahaan, I., and Huth, M. Relationship-based access control: Its expression and enforcement through hybrid logic. In *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY'12)* (San Antonio, TX, USA, Feb. 2012).

[8] Chen, H., and Li, N. Constraint generation for separation of duty. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies (SACMAT'06)* (New York, NY, USA, 2006), ACM, pp. 130–138.

[9] Cheng, Y., Park, J., and Sandhu, R. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *Proceedings of the 4th IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT'12)* (Amsterdam, Netherlands, Sept. 2012).

[10] Cheng, Y., Park, J., and Sandhu, R. A user-to-user relationship-based access control model for online social networks. In *Proceedings of the 26th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec'12)* (Paris, France, July 2012), vol. 7371 of *LNCS*.

[11] Crampton, J. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT'03)* (New York, NY, USA, 2003), ACM, pp. 43–50.

[12] Crampton, J. Why we should take a second look at access control in Unix. In *Proceedings of the 13th Nordic Workshop on Secure IT Systems (NordSec'08)* (Copenhagen, Denmark, Oct. 2008).

[13] Crampton, J., and Sellwood, J. Path conditions and principal matching: A new approach to access control. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies (SACMAT'14)* (New York, NY, USA, 2014), ACM, pp. 187–198.

[14] Fong, P. W. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY'11)* (New York, NY, USA, 2011), ACM, pp. 191–202.

[15] Fong, P. W. L., Mehregan, P., and Krishnan, R. Relational abstraction in community-based secure collaboration. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13)* (Berlin, Germany, Nov. 2013), pp. 585–598.

[16] Fong, P. W. L., and Siahaan, I. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT'11)* (Innsbruck, Austria, June 2011), pp. 51–60.

[17] Gligor, V., Gavrila, S., and Ferraiolo, D. On the formal definition of separation-of-duty policies and their composition. In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on* (May 1998), pp. 172–183.

[18] Jaeger, T. On the increasing importance of constraints. In *Proceedings of the Fourth ACM Workshop on Role-based Access Control (RBAC'99)* (New York, NY, USA, 1999), ACM, pp. 33–42.

[19] Jaeger, T., and Tidswell, J. E. Practical safety in flexible access control models. *ACM Trans. Inf. Syst. Secur. 4*, 2 (May 2001), 158–190.

[20] Kuijper, W., and Ermolaev, V. Sorting out role based access control. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies (SACMAT'14)* (New York, NY, USA, 2014), ACM, pp. 63–74.

[21] Li, N., Tripunitara, M. V., and Bizri, Z. On mutually exclusive roles and separation-of-duty. *ACM Trans. Inf. Syst. Secur. 10*, 2 (May 2007).

[22] Rizvi, S. Z., and Hosseinkhani, M. ReBAC Java Library. http://sourceforge.net/p/rebac/.

[23] Rizvi, S. Z. R. ReBAC2015: Interoperability of Relationship- and Role-Based Access Control. Master's thesis, University of Calgary, Calgary, Canada, 2015.

[24] Rizvi, S. Z. R., Fong, P. W. L., Crampton, J., and Sellwood, J. Relationship-based access control for an open-source medical records system. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies (SACMAT'15)* (Vienna, Austria, June 2015), pp. 113–124.

[25] Saltzer, J., and Schroeder, M. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (Sept 1975), 1278–1308.

[26] Sandhu, R., Coyne, E., Feinstein, H., and Youman, C. Role-based access control models. *IEEE Computer 29*, 2 (Feb 1996), 38–47.

[27] Simon, R., and Zurko, M. E. Separation of duty in role-based environments. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations (CSFW'97)* (Washington, DC, USA, 1997), IEEE Computer Society, pp. 183–.

[28] Thompson, S. *Haskell: The Craft of Functional Programming*, 3rd edition ed. Addison-Wesley, 2012.

[29] Tidswell, J. E., and Jaeger, T. An access control model for simplifying constraint expression. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)* (New York, NY, USA, 2000), ACM, pp. 154–163.

APPENDIX

A. NP-COMPLETENESS OF RE-AUTH

To prove that RE-AUTH is NP-complete, we show that it is in NP, and that it is NP-hard.

RE-AUTH ***is in NP.*** A nondeterministic algorithm can decide RE-AUTH by first guessing subset $\alpha \subseteq E$, and then verifying that (a) $\mathsf{consistent}(\alpha)$, and (b) $\mathsf{priv}(\alpha) \models MG(m)$, both can be accomplished in time polynomial to the size of the protection state.

RE-AUTH ***is NP-hard.*** We reduce the Independent Set problem to RE-AUTH. The Independent Set problem, which is known to be NP-complete, asks, given an undirected graph $G = \langle V_G, E_G \rangle$ and a positive integer k, does there exist a set of vertices, $U \subseteq V_G$, with $|U| = k$, such that for every pair of vertices, $u, v \in U$, $uv \notin E_G$?

Suppose an instance of Independent Set is given: $G = \langle V_G, E_G \rangle$ and k, where $V_G = \{v_1, ..., v_n\}$. We describe below the construction of a corresponding instance of RE-AUTH consisting of a ReBAC2015/Constraints protection state, a method $m \in MD$, and a set $E \subseteq AP$ (where MD and AP are part of the protection state).

We construct a protection state that is order-free (i.e., no PAPR constraint) and has a flat demarcation hierarchy (i.e., no two distinct demarcations are related by $\leq$). Construct $PR = \{p_1, \ldots, p_k\}$ (i.e., $|PR| = k$). Let $MD = \{m\}$, where $MG(m) = \texttt{all-of(}PR\texttt{)}$. Construct an AP with $k \times n$ principals: ap_{ij}, where $1 \leq i \leq k$ and $1 \leq j \leq n$. The protection state has isometric demarcations: i.e., there is a corresponding demarcation d_{ij} for each ap_{ij} such that $AD(ap_{ij}) = d_{ij}$. We also define $PD = \{(p_i, d_{ij}) \mid p_i \in PR, d_{ij} \in DM\}$. Consequently, $\mathsf{dem}(\{ap_{ij}\}) = \{p_i\}$.

The last component of the protection state is $\#$, the MEAP constraints:

1. *Edge constraints:* For every $1 \leq x < y \leq n$ for which $v_x v_y \in E_G$, impose $ap_{ix} \# ap_{jy}$ for each $1 \leq i \leq k$ and $1 \leq j \leq k$.
2. *Vertex constraints:* For every $1 \leq x < y \leq n$, impose $ap_{ix} \# ap_{iy}$ for each $1 \leq i \leq k$.
3. *Privilege constraints:* For every $1 \leq i < j \leq k$, impose $ap_{ix} \# ap_{jx}$ for each $1 \leq x \leq n$.

Now the rest of the RE-AUTH instance consists of the only method m and the principal set $E = AP$.

It is not hard to see that a solution $\alpha \subseteq E$ exists iff the Independent Set instance has a solution. The constructed protection state consists of k privileges, $k \times n$ principals, $O(k^2 \times n^2)$ edge constraints, $O(n^2 \times k)$ vertex constraints, and $O(k^2 \times n)$ privilege constraints. The reduction can therefore be computed in $O(k^2 n^2)$ time.

Discussion. Note that the constructed RE-AUTH instance is order-free, and its demarcation hierarchy is flat. This means the MEAP constraints ($\#$) form the core of complexity for RE-AUTH. The demarcation hierarchy ($\leq$) and the PAPR constraints ($\sqsubseteq$) are not hard at all. This is easy to see because the reflexive transitive closures for $<'$ and $\sqsubset'$ can be computed in polynomial time.

B. LAZY EVALUATION IN DETAILS

There are three main steps in Algorithm 1.

Step 1: Prepare (lines 1–3). Line 1 reduces the present RE-AUTH instance to a SAT instance F using the reduction in §4.3. The function $getSATInstance(E, m)$ takes two arguments: a principal set E and a requested method m. Note that on line 1, the entire AP is passed as E. Distilling from AP what principals are enabled belongs to Step 3 below. Then on line 2 the SAT solver is initialized to work with F.[4] A map $E[\cdot]$ is used for caching the result of model checking: if $E[rp]$ is defined, then it is the value of $rp(G, o, s)$. $E[\cdot]$ is initialized to an empty map on line 3.

Step 2: Construct Principal Set (lines 4–8). First, the SAT solver is invoked to find the next model of F (line 4). If no such model is found (line 5), then we have exhausted all models of F, and thus access is denied (line 6). Otherwise, a model σ of F is found, and it encodes a principal set α that satisfies (12). The set α is extracted from σ on line 8. More specifically, $extractPrincipals(\sigma)$ returns $\{\, ap \in AP \mid \sigma(ap) = 1 \,\}$. See §4.3 for a justification.

Step 3: Verify That Principals Are Enabled (lines 9–19). The goal of this block of code is to check if all the principals in α are enabled. The for loop on lines 9–18 examines each principal ap in α in turn. Let rp be the relationship predicate that defines membership in ap (line 10). The value of $rp(G, o, s)$ is evaluated by an invocation to the model checker (line 12) if that value has not already been cached in the map $E[\cdot]$ (line 11). If ap is found not to be enabled (line 14), then the clause $(\neg ap)$ is added to F to prevent the SAT solver from attempting to enable ap in subsequently found models (line 15). As soon as α is found to contain a disabled principal, Step 2 is repeated (line 16). However, if all ap in α are confirmed to be enabled, then access will be granted (line 19).

C. EFFICIENT GENERATION AND STORAGE OF SAT INSTANCES

Note that formula F is independent of the resource (o) and requestor (s) from the request. Not only that, in the lazy authorization procedure (§4.2), E is fixed to AP, and thus F_{MEAP}, F_{PAPR} and F_{DH} do not vary across requests. Therefore, to increase efficiency, one can simply generate F once, offline, for each m, and reuse it for all requests involving m. More specifically, F_{MEAP}, F_{PAPR} and F_{DH} are the same for all methods. This means they need only be generated once offline, and stored globally. One can then generate the corresponding F_{guard} for each method m, and store it along with m. When a request is to be authorized, one can retrieve F_{guard} from m, and combine it with the globally stored F_{MEAP}, F_{PAPR} and F_{DH}, to obtain the appropriate F.

[4] Functions that interact with the SAT solvers have the prefix "*sat-*".