Extended ReBAC Administrative Models with Cascading Revocation and Provenance Support

Yuan Cheng
Institute for Cyber Security
Univ. of Texas at San Antonio
yuan@ycheng.org

Khalid Bijon
MosaixSoft
khalid@mosaixsoft.com

Ravi Sandhu
Institute for Cyber Security
Univ. of Texas at San Antonio
ravi.sandhu@utsa.edu

ABSTRACT

Relationship-based access control (ReBAC) has been widely studied and applied in the domain of online social networks, and has since been extended to domains beyond social. Using ReBAC itself to manage ReBAC also becomes a natural research frontier, where we have two ReBAC administrative models proposed recently by Rizvi *et al.* [30] and Stoller [33]. In this paper, we extend these two ReBAC administrative models in order to apply ReBAC beyond online social networks, particularly where edges can have dependencies with each other and authorization for certain administrative operations requires provenance information. Basically, our policy specifications adopt the concepts of enabling precondition and applicability preconditions from Rizvi *et al.* [30]. Then, we address several issues that need to be considered in order to properly execute operation effects, such as cascading revocation and integrity constraints on the relationship graph. With these extended features, we show that our administrative models can provide the administration capability of the MT-RBAC model originally designed for multi-tenant collaborative cloud systems [34].

Categories and Subject Descriptors

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

Keywords

Access Control; Relationship; Administrative Model

1. INTRODUCTION

The rapid emergence of online social networks (OSNs) has led to emergence of several relationship-based access control (ReBAC) models in this domain, in both research and practice. In contrast with conventional access control, ReBAC determines access in terms of relationships among users and resources. Considerable research has been conducted on the extensions of ReBAC schemes in the context of OSNs [2, 6–9, 14, 16], offering users more fine-grained and expressive solutions than current commercial systems. In addition to social computing, Fong *et al.* proposed a series of ReBAC models that use modal logic as policy specifications and seek to apply ReBAC to general computing systems [2, 14, 16]. The RPPM model developed by Crampton *et al.* also intends to employ ReBAC to applications beyond social computing [11], with policy specifications similar to path expressions in Cheng *et al.*'s proposals for ReBAC [7, 8]. The name "RPPM" stands for "relationships, paths, and principal-matching".

Administration of ReBAC has to be carefully addressed, because a change of relationships may result in change of authorization. The dynamic and decentralized nature of OSNs, where ReBAC is mainly deployed so far, suggests a unified but decentralized solution to enforce administration in a scalable and efficient way. Following the prior success of using role-based access control (RBAC) to manage RBAC [10, 19, 21, 26, 31, 35], a natural direction for ReBAC adminstration would be using ReBAC itself to manage ReBAC.

Very recently we have seen proposals from researchers in this direction. Two groups of researchers extended the RPPM model by Crampton *et al.*, and independently developed their models for ReBAC administration. The main contribution of Rizvi *et al.* [30] lies in the implementation of ReBAC in a medical record system, where administrative actions regarding relationship edges are addressed in terms of security preconditions and execution effects. Stoller's RPPM^2 model [33], on the other hand, focuses on policy specifications and provides a complete coverage of ReBAC administration, including changes on entities, edges, and policies.

In this paper we seek to extend the ReBAC administration models cited above, inspired by an application domain for ReBAC beyond those considered in the literature thus far. The concept of multi-tenancy is essential to cloud computing, where multiple customers are served virtual resources within a single, shared physical computing environment. In addition to isolating tenants from each other, cloud service providers have incorporated facilities for authorized cross-tenant interaction. Based on trust relations among tenants, a multi-tenant RBAC model, namely MT-RBAC, has recently been developed for this purpose. MT-RBAC has been defined in traditional RBAC terms in [34]. However, it can alternatively be viewed as a ReBAC model. MT-

SACMAT'16, June 05-08, 2016, Shanghai, China
© 2016 ACM. ISBN 978-1-4503-3802-8/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2914642.2914655

RBAC features tenant trust relation, user-ownership, role-ownership, and object-ownership in addition to user-role assignments and permission-role assignments in the original RBAC model [13, 32]. These various relationships between users, roles, objects, and tenants can be cast as a relationship graph, analogous to the social graph in OSNs and thereby exploited for ReBAC authorization and administration.

A significant difference between the OSN domain wherein ReBAC emerged and more traditional IT (information technology) domains such as MT-RBAC is in the nature of integrity requirements for the relationship graph. Consider the familiar friend relationship in Facebook. Creation of a friend relation between, say Alice and Bob, often requires a prior friend-of-friend relation through some common third user, say Cathy [15]. However, once established the friend relationship between Alice and Bob will persist even if Cathy drops her friendship with either one or both of them. In MT-RBAC however the dependence of relationships on other relationships endures beyond the initial creation. To be concrete, a user u owned by tenant x can be assigned to a role r owned by tenant y only if tenant x trusts tenant y. This tenant-tenant trust is required not only when the u to r relationship is established but also subsequently. Therefore, if the tenant trust relationship is revoked at some later time there is an obligation to also revoke the u to r assignment. Depending on policy requirements this may entail a cascading revocation, which introduces subtleties in defining an appropriate ReBAC administration model. While cascading revocation has been extensively studied in the literature (e.g. [1,12,18]), to the best of our knowledge this paper is the first to incorporate this phenomenon in context of ReBAC.

In this piece of research, we develop a family of three administrative models for ReBAC. Our *first contribution* is the base model called AReBAC_1 that formally represents the administrative model proposed by Stoller [33]. It also augments the capability of Stoller's model by including consistency checking functionality to the administrative operations and adding support for pre-applicability conditions proposed in [30]. A pre-applicability condition seeks to preserve the integrity constraints of the relationship graph. Our *second contribution* is AReBAC_2 that extends AReBAC_1 to support cascading revocation. We propose a cascading revocation algorithm that is specifically designed for the context of ReBAC. We also conduct evaluation on the algorithm. Our *final contribution* is AReBAC_3 that offers additional ability to address authorization based on provenance information. To summarize, this work identifies and addresses some important issues in ReBAC administration, which the existing administrative models have not considered and, henceforth, promotes ReBAC administration beyond the conventional context of social computing. These models are inspired by considering administrative requirements for the MT-RBAC model. They are not intended as a replacement or generalization of MT-RBAC but rather proposed as a relation-based framework to explore ReBAC administrative issues. MT-RBAC is a rather novel ReBAC instance relative to current ReBAC literature, and brings to light significant administrative aspects which have not been recognized so far.

The paper is organized as follows. In the next section, we describe the motivating administrative issues inspired by MT-RBAC. Section 3 formally introduces AReBAC_1. Section 4 presents AReBAC_2 and AReBAC_3, and compares them with the prior work. This section also contains our proposed algorithm for cascading revocation. Section 5 analyzes performance of the algorithm. Section 6 reviews related work regarding ReBAC and administrative access control models. Section 7 gives our conclusions.

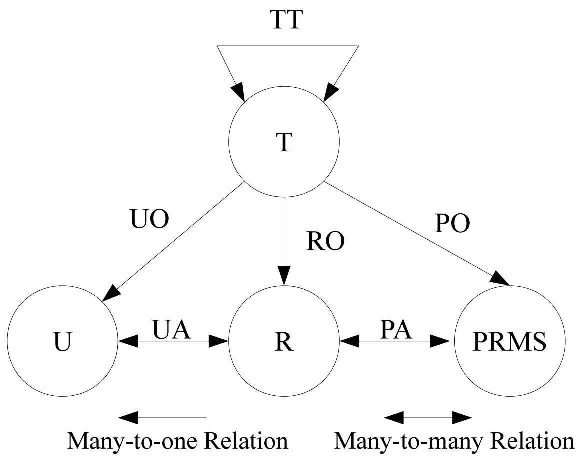

Figure 1: Multi-tenant RBAC model structure

2. MOTIVATION

Administrative operations are more risky than regular operations. A properly designed administrative model needs to ensure that these operations are performed safely. It becomes an even more important requirement for decentralized systems as these operations can be done by regular users who may not possess the expertise as a system administrator does. Moreover, administrative operations must comply with the business logic of the system and the semantics of the data model. One such system is multi-tenancy authorization in cloud computing environment where dynamic trust relations among the tenants (organizations) drive their collaborations. The main challenge here is dynamic administration of user privileges that includes dynamic cascading revocation of user privileges and conflict resolution of the requested administrative operations from multiple tenants.

Recently, the MT-RBAC model [34] extends traditional role-based access control model to provide multi-tenancy authorization. MT-RBAC is defined in the traditional style of RBAC models. In this paper we recast it in the style of ReBAC models, which leads us to consider some administrative issues with respect to MT-RBAC which are not so convenient to formalize in the RBAC style formulation. Motivated by these considerations we extend the ReBAC administrative models proposed in [30, 33]. We show how to configure variations of MT-RBAC in these extended administrative models in a relatively straightforward manner.

MT-RBAC consists of four components as illustrated in Figure 1: tenants (T), users (U), roles (R) and permissions (PRMS). A tenant is a virtual partition of a cloud service. A user is an individual owned by a single tenant via user-ownership (UO) relation. Each tenant may own multiple users. Hence the UO relation is many-to-one, relating multiple users to one tenant. A role is a job function associated with a single tenant while a tenant may own multiple roles. Thereby, role-ownership (RO) is many-to-one. A permission is a specification of a privilege to an object in a tenant. A permission is denoted as a 3-tuple (privilege, tenant, object). For example, (read, Dev.E, /root/) represents a permission of reading the "/root/" path on tenant Dev.E. Every permission is associated with a single tenant, who can own multiple permissions. Therefore, permission-ownership (PO) is also many-to-one. User-role assignment (UA) and permission-role assignment (PA) relations connect users and permissions through roles. These are many-to-many relations.

Tenant-trust (TT) denotes a many-to-many trust relation between tenants. We use notation $\trianglelefteq$ to represent trust between two tenants so $T_A \trianglelefteq T_B$ means T_A (trustor) trusts T_B (trustee). The reflexive (but not transitive, symmetric or anti-symmetric) TT relation enables cross-tenant collaboration. If $T_A \trianglelefteq T_B$ then users of T_A can be assigned to T_B's roles by T_B, hence gaining permissions that are associated with T_B's roles. Note that due to reflexivity $T_A \trianglelefteq T_A$ always holds so intra-tenant assignment of T_A's users to T_A's roles is always allowed. The main objective of MT-RBAC is to enable cross-tenant assignment based on TT.

In MT-RBAC, along with the removal of a tenant entity, its correlated trust relations and authorization assignments should also be removed accordingly. Similarly, the revocation of a trust relation between two tenants should induce revocation of its correlated user-role assignments as well. This property is known as cascading revocation, however, the current MT-RBAC literature does not provide any mechanism to address it. Inspired by MT-RBAC we also recognize that edges and nodes in ReBAC systems can have dependency and correlation with each other, hence, cascading removal of nodes/edges is intrinsic to ReBAC. To make the administrative model comprehensive, we need to address the dependency issues adequately. The applicability check prior to the operation and the post-operation effects can accommodate the cascading revocation. It is more convenient to consider these issues in a ReBAC setting rather than in the traditional RBAC style of models.

Furthermore, in some settings for MT-RBAC, such relations can be added, altered or removed by multiple administrative users from different tenants, causing potential conflicts or unexpected results. To clarify the situation, the administrative model should offer the ability to distinguish administrative operations initiated by different users, or support additional data structures to record the provenance (history) of these operations. Again, some relations in MT-RBAC such as the user-ownership relation between users and tenants have to be many-to-one so each user has a unique owner tenant. Similarly for the tenant-role relationship. In general there are many global integrity constraints like these two examples, which need to be maintained before and after each administrative operation is conducted. These constraints specify the configurations of the data model for the relationship graph that are considered semantically correct. However, the integrity constraint check is often overlooked in ReBAC literature, since authorization in ReBAC mainly focuses on specifying path conditions that regulate the requesting subject. Specifically, the RPPM^2 model copes with the situations regarding adding an edge that already exists or removing an edge that does not exist, but these special cases are not further generalized in the policy language.

Motivated by these problems, we extend the existing administrative ReBAC models and propose a family of three models that capture global integrity policy checks, cascading revocation, and multi-ownership conflict, respectively.

3. BACKGROUND

In this section, we first summarize the RPPM^2 [33] model. We then formally present our proposed core administrative model for ReBAC, namely AReBAC$_1$.

3.1 RPPM^2

In context of developing an administrative model for ReBAC, Stoller [33] proposed RPPM^2 (RPPM Modified), which extends the RPPM model proposed in [11]. Motivation and illustrative examples of these models are given in the respective papers. This paper introduces a family of administrative models, which basically extends the RPPM^2 model. We now describe the RPPM^2 model as follows.

System Model and System Instance. A system model comprises a set of types T, a set of relationship labels R, a set of symmetric relationship labels $S \subseteq R$ and a permissible relationship graph $G_{PR} = (V_{PR}, E_{PR})$, where $V_{PR} = T$ and $E_{PR} \subseteq T \times T \times R$. Given a system model (T, R, S, G_{PR}), a system instance is defined by a system graph $G = (V, E)$ and a type function τ: $V \to T$, where V is the set of entities and $E \subseteq V \times V \times R$. G is well-formed if for each entity v in V, $\tau(v) \in T$, and for every edge $(v, v', r) \in E$, $(\tau(v), \tau(v'), r) \in E_{PR}$.

Request. A request req is in the form of $(s, op(v_1, \dots, v_n))$, where s is the subject (i.e., an entity that requests for the operation), op is the operation, and the v_i are target entities on which the operation will perform.

Path Expression and Path Condition. Path expressions are defined recursively: $\diamond$ is a path expression; r is a path expression, for all $r \in R$; if π and π' are path expressions, then $\pi;\pi'$, $\pi+$, $\pi*$, and $\bar{\pi}$ are path expressions. A path condition in RPPM^2 has the form $e_1 \cdot \pi \cdot e_2$, where each e_i is an entity constant or a variable that belongs to V_{PR} of the permissible relationship graph $G_{PR} = (V_{PR}, E_{PR})$, and π is a path expression. A path condition $e_1 \cdot \pi \cdot e_2$ holds if there exists a substitution θ mapping the variables (if any) in the path condition to values such that the system graph contains a path from $e_1\theta$ to $e_2\theta$ that matches $\pi\theta$, where $t\theta$ denotes the result of applying substitution θ to term t.

Principal Matching. Principal matching replaces a path between two entities with a single principal name. The principal name is a shorthand for path expression. In RPPM^2, a name can be defined to represent multiple path expressions.

Authorization Rule and Authorization Policy. An authorization rule $\mathcal{R}$ is defined in the form (req, c, d), where req is a request, c is a conjunction of path conditions, and d is binary decision. A rule says that the decision d is true for request req if all conditions in c are satisfied. An authorization policy is a collection of authorization rules.

Defaults. A system-wide default decision can also be specified, which is used when no rules and no other defaults apply. We refer interested reader to [11, 33] for details.

Figure 2 shows the ReBAC configuration for MT-RBAC that we described in section 2. In this configuration, the set of types T contains tenant, user, role and permission. The set of relationship label R contains UO, RO, PO, UA and PA. Figure 2-A shows the permissible relationship graph. Suppose there is a rule ($read$, c, d) in $\mathcal{R}$, where c is the condition defined as $user$. UA . $role \wedge$ $role$. PA . $permission$.

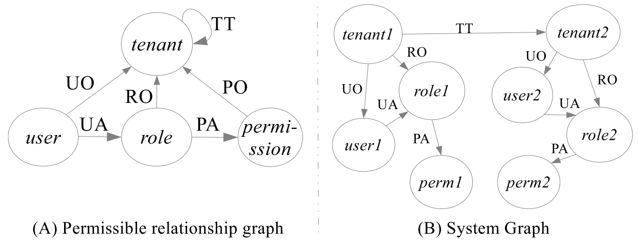

Figure 2: ReBAC for MT-RBAC example

Now, given the system graph in Figure 2-B, if $user_1$ tries to access the permission $perm_1$, the value of d in the rule (*read*, *c*, *d*) is true.

3.2 AReBAC$_1$ Model

The administrative model proposed by Stoller [33] includes operations to add and delete entities, edges, and authorization rules, plus three administrative actions to set defaults. We only focus on administrative operations to add and delete edges. We now formally present AReBAC$_1$, which is the core model of our proposed family of administrative models. Basically, it summarizes these two operations as they are proposed in [33] with the extension of consistency policies and global integrity constraints. Consistency policies ensure that the system graph $G = (V, E)$ is always well-formed after allowing each administrative operation. The global integrity constraints are semantically equivalent to "applicability precondition" defined by Rizvi *et al.* [30] that constrain operations based on certain conditions for both primary and auxiliary participants. Note that, we augment this model for supporting cascading revocation of edges in the following section.

AReBAC$_1$ provides two operations called $\mathcal{A}dd$ and $\mathcal{RM}$ respectively to add and remove edges to a system graph $G = (V, E)$, where each operation is a function that takes as input the administrative entity that performs the operation, a relationship label and two entities between which the edge with given relationship label will be added/removed. Each operation also evaluates the consistency policy in order to keep G well-formed. Formally these two operations are defined as follows (the notation for defining these operations is similar to the notation of schema used in NIST RBAC [13]).

$\mathcal{A}dd(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2, \mathrm{r}) \lhd$
$\mathrm{e}_{admin} \in V \wedge \mathrm{e}_1 \in V \wedge \mathrm{e}_2 \in V \wedge$
$\mathrm{r} \in R \wedge (\tau(\mathrm{e}_1), \tau(\mathrm{e}_2), \mathrm{r}) \in E_{PR}$
$E' = E \cup \{<\mathrm{e}_1, \mathrm{e}_2, \mathrm{r}>\} \rhd$
$\mathcal{RM}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2, \mathrm{r}) \lhd$
$\mathrm{e}_{admin} \in V \wedge (\mathrm{e}_1, \mathrm{e}_2, \mathrm{r}) \in E$
$E' = E - \{<\mathrm{e}_1, \mathrm{e}_2, \mathrm{r}>\} \rhd$

Here, a successful execution of an operation is allowed if the specified consistency policy is satisfied. Note that, e_{admin} is an entity, which we sometimes refer to as administrator. Besides she is authorized to perform an operation, an administrator is similar to other entities. For example, she may also have non-administrative permissions. In this system, the authorization of an administrator for an operation is regulated by a fixed set of positive policy rules $\mathcal{P}$. Each policy rule $p \in \mathcal{P}$ has the form $p = \mathsf{OP}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2, \mathrm{r}) \leftarrow \mathsf{enableC}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2) \wedge \mathsf{preC}(\mathrm{e}_1, \mathrm{e}_2)$, where OP is $\mathcal{A}dd$ or $\mathcal{RM}$. This represents that an administrator is authorized to request the operation if both predicates preC and enableC are satisfied. An enableC is an enabling precondition that specifies certain relationship between the administrator, e_{admin}, and two target entities, e_1 and e_2. An enableC can be specified as conjunction of multiple path conditions and verified with e_{admin}, e_1, e_2 and other necessary instances of the system graph. On the other hand, a preC specifies relations between two target entities disregarding the administrator who requests to perform the operations. Unlike enableC, preC is not specified as path condition, rather it should be specified as hybrid logic formula as mentioned in [30]. In this paper, we do not aim to develop policy specification language for preC. Instead, we express them using simple set theory notation. Note that, in a policy rule $p \in \mathcal{P}$ one may also specify one or both predicates to be always true.

Table 1 shows examples of AReBAC$_1$ based on Figure 1. Example 1 shows an $\mathcal{A}dd$ operation for a tenant-trust (TT) edge where tenant$_1$ is the e_{admin} who wants to add the edge between tenant$_1$ and tenant$_2$. Note that, in order to authorize this operation only consistency check is necessary, hence, predicates enableC and preC are always true. In example 2, tenant$_1$ wants to remove a user-role (UA) edge between user$_1$ and role$_1$. Here, besides consistency condition, enableC ensures that both user$_1$ and role$_1$ belong to tenant$_1$. However, it does not require an applicability precondition, thereby preC is always true. Finally, in example 3, for adding a user-ownership (UO) between tenant$_2$ and user$_2$ no enableC is required. However, according to MT-RBAC, UO is one-to-many relation. Hence, it is necessary to check if user$_2$ already belongs to another tenant or not. A preC checks this global integrity constraint by checking if an edge (—, user$_2$, TT) already exists in G. Here, '—' represents all the tenants in the system. According to Figure 1 there exists no such edge and the request should be authorized.

4. ENHANCEMENT OF THE MODEL

In this section, we extend the AReBAC$_1$ model to facilitate cascading removal of edges and dynamic conflict resolution by provenance support.

4.1 AReBAC$_2$: Cascading Revocation

In ReBAC, creation of some edges might depend on the existence of another edge, whereby, dependent edges need to be removed at the time of removal of the dependency edge. We define this situation as cascading revocation of edges. Cascading revocation implies that the operation will trigger a series of recursive removal of edges on the graph in addition to the direct consequence of the operation. We augment the functionality of AReBAC$_1$ here in AReBAC$_2$ to support this cascading revocation.

AReBAC$_2$ augments the representation of each policy $p \in \mathcal{P}$ that regulates $\mathcal{RM}$ operations as follows.

$p = \mathcal{RM}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2, \mathrm{r}) \leftarrow \mathsf{enableC}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2) \wedge \mathsf{preC}(\mathrm{e}_1, \mathrm{e}_2) : \mathcal{C}_{revoke}(\mathrm{e}_1, \mathrm{e}_2, \mathrm{r})$.

$\mathcal{C}_{revoke}(\mathrm{e}_1, \mathrm{e}_2, \mathrm{r})$ is a function that takes as input e_1, e_2 and r, and returns a set of edges that needs to be removed (possibly empty) when the policy p is used to authorize operation $\mathcal{RM}(\mathrm{e}_{admin}, \mathrm{e}_1, \mathrm{e}_2, \mathrm{r})$. Note that, edges returned by the function are being removed without further authorization. In many systems, a cascading revocation is desired

Table 1: The Policies in AReBAC$_1$: An MT-RBAC Example

Description	Operation	Enabling Pre-Condition	Applicability Pre-Condition
1. Add tenant-trust edge	$\mathcal{A}dd$(tenant$_1$, tenant$_1$, tenant$_2$, TT)	True	True
2. Remove user-role assignment edge	$\mathcal{RM}$(tenant$_1$, user$_1$, role$_1$, UA)	$user \cdot UO \cdot tenant \wedge role \cdot RO \cdot tenant$	True
3. Add user-ownership edge	$\mathcal{A}dd$(tenant$_2$, tenant$_2$, user$_2$, UO)	True	(—, user$_2$, UO) $\notin$ E

instead of a non-cascading one. For instance, in MT-RBAC, along with the removal of a tenant, its correlated trust relations, users, roles and permission assignments should be also removed. Again, when a tenant trust relation is revoked, the user-role assignments initiated by the trustee tenant also require to be consequently removed. Similar examples can be found in online social networks, health care systems, and database systems as well, where dependency of relationships exists.

Figure 3 illustrates two examples of the cascading revocation in MT-RBAC scenarios. Figure 3-A shows that the removal of a user-ownership (UO) edge between tenant$_1$ and user$_1$ also causes removal of user-role assignment (UA) edge between user$_1$ and role$_1$. In this case, $\mathcal{C}_{revoke}$ takes tenant$_1$, user$_1$ and UO as parameters and returns a set that contains a tuple (user$_1$, role$_1$, UA). Similarly, in Figure 3-B, when a revocation of a TT relation is issued, the correlated cross-tenant user-role assignments specified by the trustee are automatically removed. Here, $\mathcal{C}_{revoke}$ takes tenant$_1$, tenant$_2$ and TT as parameters and returns the set {(user$_1$, role$_2$, UA)}.

For a particular policy, a $\mathcal{C}_{revoke}$ may return zero to multiple edges that are revoked as a consequence of revocation of an edge. Note that, identification of such edges in an efficient way is a non-trivial process since a system graph can have thousands of edges with arbitrary cascading relations. One trivial solution is to maintain relations between each dependency and dependent edge pair in the system graph. When an administrator adds a new edge, the system will find the dependency edges for it and create a new relation. Later, if the dependency edge is removed, dependent edges for it will be retrieved from the maintained relations and removed accordingly. However, this process is not scalable in a large system. Toward this end, we develop a scalable solution for identifying dependent edges.

4.1.1 Dependent-Edge Discovery Algorithm

We discuss our developed procedure that dynamically discovers dependent edges. In this procedure, we maintain a function called $\Phi_{dependency}$ that maps an edge (e$_1$, e$_2$, *label*) to a tuple $\langle$ Path, R$_d$ $\rangle$. Note that, (e$_1$, e$_2$, *label*) is the dependency edge that will cause cascading removal of other edges from system graph G. Here, Path is an ordered set that contains relationship labels in order and R$_d$ is another set containing relationship labels where edges with these labels will be removed. Note that, Path can contain multiple instances of same relationship label. Algorithm 1 finds the dependent edges. Basically, it is based on a depth-first search algorithm, where, for a given dependency edge (e$_1$, e$_2$, *label*), it starts with a source e$_1$ (or possibly e$_2$) in G and recursively tries to reach destination node e$_2$ (or e$_1$). In order to reach e$_2$, it only visits edges according to the given

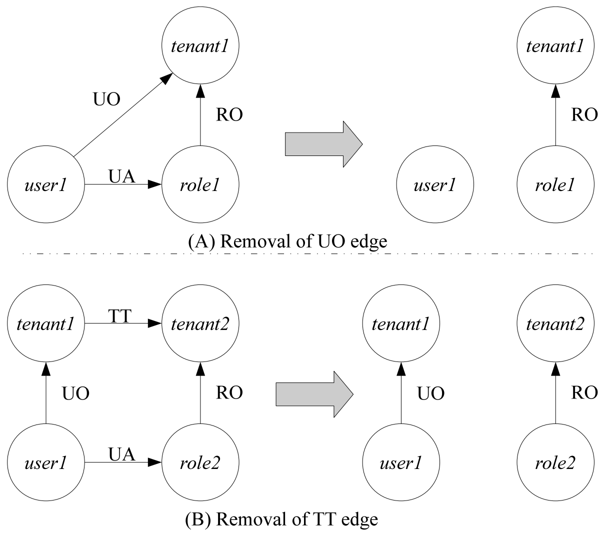

Figure 3: Cascading revocation during the removal of UO and TT edges

order of the labels in Path. Line 14 of the algorithm shows that a label called curE is picked from the top position of Path. Then, lines 16-17 ensures that the algorithm will only visit next node e$_i$ from node e$_s$, if the edge between e$_s$ and e$_i$ is labeled with curE. Line 18 further checks if the label is in R$_d$ and marks it as dependent edge by putting it in set T$_d$. Then, this scenario is recursively applied for e$_i$ with next top label in Path. Finally, if it reaches destination node e$_j$, lines 9-13 check if all the labels are used from Path, and then add the dependent edges from T$_d$ to E$_d$. Finally, the algorithm returns all the dependent edges in E$_d$ (line 7).

For the example given in Figure 3-B, $\Phi_{dependency}$ maps (tenant$_1$, tenant$_2$, TT) to the tuple $\langle$ Path$_1$, R$_{d_1}$ $\rangle$ where Path$_1$ = (UO, UA, RO) and R$_{d_1}$ = {UA}. Then, algorithm 1 takes as input tenant$_1$, tenant$_2$, Path$_1$, and R$_{d_1}$ and returns the set {(user$_1$, role$_2$, UA)}, which edges should also be removed. Note that a cascading revocation can be applied to all the dependent edges that the revokee (dependency) edge previously enabled and, recursively, all the dependents of the dependents of the revokee edge or entity. For simplicity, we only consider one-level of cascading revocation here.

4.1.2 Time Complexity of $\mathcal{C}_{revoke}$

In our proposed solution, the time complexity of $\mathcal{C}_{revoke}$ depends on the complexities of algorithm 1 and $\Phi_{dependency}$. $\Phi_{dependency}$ is a mapping function and the time complexity of it, to map an input edge to a tuple $\langle$ Path, R$_d$ $\rangle$, depends on the implementation choice. We implemented $\Phi_{dependency}$ as

Algorithm 1 Discover Dependent-Edge

Require: A source node e_s, a destination node e_d, an ordered set of relationship labels Path, a set of relationship labels R_d where edges with these labels are the dependent edges.

Ensure: Returns a set of edges E_d that will be removed.

```
 1: $\mathsf{E}_d := \emptyset$
 2: $\mathsf{Visited} := \emptyset$
 3: if $\mathsf{R}_d = \emptyset$ or $\mathsf{Path} = \emptyset$ then
 4:     return $\mathsf{E}_d$
 5: end if
 6: Find_Edges($e_s$, $e_d$, Path, $\mathsf{R}_d$, $\mathsf{E}_d$, Visited, {})
 7: return $\mathsf{E}_d$
 8: procedure FIND_EDGES($e_s$,$e_d$,Path,$\mathsf{R}_d$,$\mathsf{E}_d$,Visited,$\mathsf{T}_d$)
 9:     if $e_s = e_d$ then
10:         if $\mathsf{Path} = \emptyset$ then
11:             $\mathsf{E}_d := \mathsf{E}_d \cup \mathsf{T}_d$
12:         end if
13:     end if
14:     curE := Path.top()
15:     $\mathsf{Visited} := \mathsf{Visited} \cup e_s$
16:     for all $e_i \in V$ do
17:         if $e_i \notin \mathsf{Visited}$ and $\langle e_s, e_i, \text{curE}\rangle \in E$ then
18:             if curE $\in \mathsf{R}_d$ then
19:                 $\mathrm{T}_d := \mathrm{T}_d \cup \langle e_s, e_i, \text{curE}\rangle$
20:             end if
21:             Find_Edges($e_i$,$e_d$,Path-{curE},$\mathsf{R}_d$,$\mathsf{E}_d$,Visited, $\mathsf{T}_d$)
22:             if $\langle e_s, e_i, \text{curE}\rangle \in \mathrm{T}_d$ then
23:                 $\mathrm{T}_d := \mathrm{T}_d - \langle e_s, e_i, \text{curE}\rangle$
24:             end if
25:         end if
26:     end for
27:     $\mathsf{Visited} = \mathsf{Visited} - e_s$
28: end procedure
```

HashMap, discussed in section 5, and its time complexity is $\mathcal{O}(1)$. Each execution of the algorithm 1 performs a depth-first search in the current system graph $G = (V, E)$. We assume that E is always represented as adjacency-list and, therefore, the time complexity of the algorithm is $\mathcal{O}(V+E)$. The overall time complexity of $\mathcal{C}_{revoke}$, to find the set of edges that needs to be removed due to dependency on a removed edge, is $\mathcal{O}(V+E)$.

4.2 AReBAC$_3$: Provenance Support

In ReBAC, relationships are utilized to make access decision. But it is likely the case that in a real world system, other forms of information and knowledge will also come into play together with relationships for achieving desirable access control objectives. There has recently been a surge of interest in harnessing provenance information to enable additional versatile control capabilities not available with conventional access control solutions [24, 28].

Provenance refers to the documentation of the history of a data item starting from its original sources to its current state. Provenance data can provide utilities such as pedigree information, query, usage tracking, versioning, data auditing, and error detection, etc. Among various kinds of provenance data and usage, we are particularly interested in causality dependencies that record the flow of transactions that occurred in the system, since they can provide us the foundation for building and delivering more expressive access control features.

The Open Provenance Model (OPM) [22] is a model of provenance that aims to capture the causality dependencies between entities. It provides a foundation for expressing such dependencies, the provenance graph. A provenance graph is defined as a directed graph, whose nodes are artifacts, processes and agents, and whose edges are causal relationships between the aforementioned nodes. It can be computed from the transaction records of the system. An artifact is used to represent a state of a data object (e.g., an added edge or a removed edge). A process denotes an action or a series of actions performed on or caused by artifacts, and resulting in new artifacts (e.g., create or remove an edge). An agent corresponds to a user who executes the action (e.g., a tenant). There are five causal relationships defined in OPM: a process used an artifact; an artifact was generated by a process; a process was triggered by a process; an artifact was derived from an artifact; and a process was controlled by an agent. We adopt these causal relationships to represent dependencies among artifacts, processes and agents.

We build our provenance feature on top of OPM as the model enables us to capture and express the casuality dependencies. We name the provenance-assisted model, AReBAC$_3$.

Next, we present an exemplary usage of the provenance support in the multi-tenancy scenario mentioned earlier. In the previous MT-RBAC example, we assume that the user-ownership is one-to-many, which means a user is restricted to be owned by one single tenant. If many-to-many ownership is allowed, then a user can be assigned to multiple tenants, thus making the authorization assignments more complicated than before.

As shown in Figure 4, user$_1$ is owned by tenant$_1$ and tenant$_2$; meanwhile both tenant$_1$ and tenant$_2$ trust another tenant tenant$_3$. Due to the tenant trust with the owners, the trustee tenant tenant$_3$ is allowed to assign user$_1$ to one of its roles, say role$_1$. Later on when one of the tenants, say tenant$_2$, decides to revoke the tenant trust relation it previously initiated, we will have to consider whether to remove the UA relation between user$_1$ and role$_1$ or not.

There are many strategies to resolve conflicts among different administrators: permissions-take-precedence, denials-take-precedence, recency precedence, distance precedence, etc. This problem has been extensively studied in many domains in the past, and a detailed consideration is out of scope of this paper. We will leave it to the system architect to decide the conflict resolution policy. But first of all, we need to distinguish the two assignments from different tenants and then decide whether to remove them or not.

The RPPM^2 model offers relationship label with typed parameters. In our scenario, we can use typed parameters to distinguish edges assigned by different tenants. The user-role assignments become two separate edges (user$_1$, role$_1$, UA(tenant$_1$)) and (user$_1$, role$_1$, UA(tenant$_2$)). Revocation initiated by tenant$_2$ will only remove the edge (user$_1$, role$_1$, UA(tenant$_2$)) and leave (user$_1$, role$_1$, UA(tenant$_1$)) as it is.

An alternative way of distinguishing multiple ownerships is to incorporate provenance information to edges. We can capture the user-role assignment using the OPM provenance graph illustrated in Figure 5. The UO edge (tenant$_1$, user$_1$, UO) was generated by the process create$_1$ controlled by tenant$_1$. Similarly, we can express the causal relationships for edges (tenant$_3$, role$_1$, RO) and (tenant$_1$, tenant$_3$, TT). There are three "used" edges from the process create$_4$ to three artifacts generated in the prior processes. These artifacts are input objects used in the process create$_4$, indicating

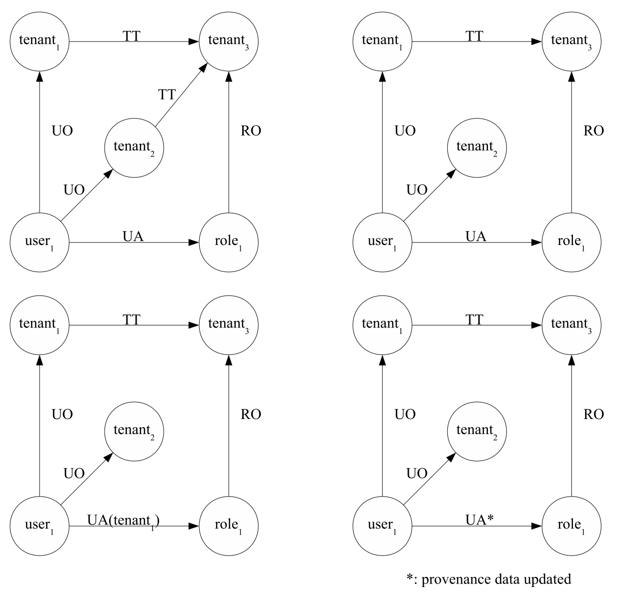

Figure 4: Two ways of distinguishing edges assigned by multiple administrators

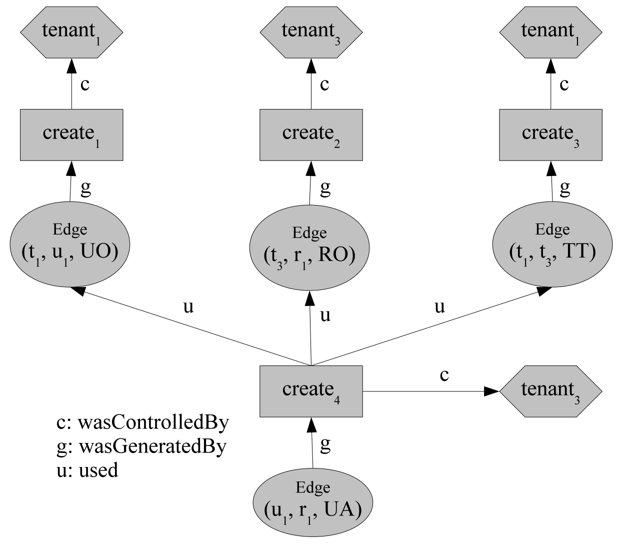

Figure 5: OPM provenance graph for adding UO edge

that a complete execution of the process create$_4$ requires the existence of these three edges. The "wasGeneratedBy" edge says that the process create$_4$ was required to generate the edge (user$_1$, role$_1$, UA).

The scenario depicted in Figure 4 can be captured in OPM provenance graph as well, as shown in Figure 6. Both edges (tenant$_1$, tenant$_3$, TT) and (tenant$_2$, tenant$_3$, TT) were used to generate the user-role assignment (user$_1$, role$_1$, UA) initiated by tenant$_3$. Depending on the conflict resolution strategy a system picks, the removal of the edge (tenant$_2$, tenant$_3$, TT) would trigger either the removal of the edge (user$_1$, role$_1$, UA) or an update on the UA edge (and its metadata). In either case, a new artifact would be generated by the triggered process to reflect the change.

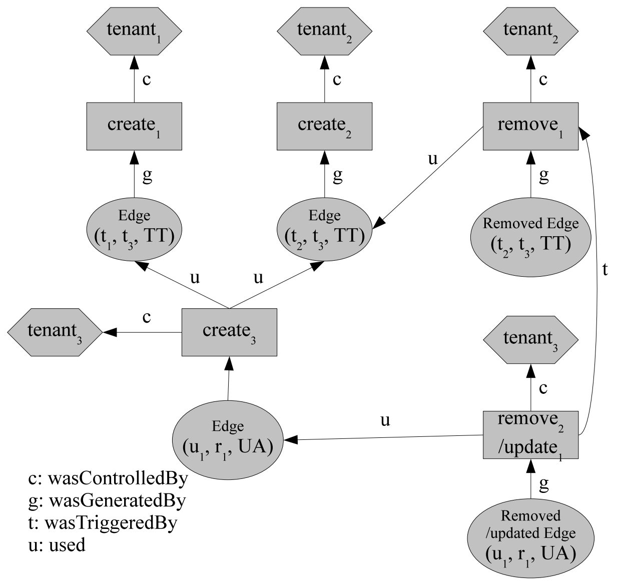

Figure 6: OPM provenance graph for removing UO edge

With the provenance data, we can keep track of the history of the relationship edges without creating multiple edge instances. The system is also able to get necessary information to remove or update dependant edges as a response to those changes that have been made in the system.

4.2.1 Discussion

The initial intention of introducing provenance information to ReBAC is to capture and express causality dependencies for assisting authorization decisions, such as resolving conflicts and ambiguity due to multiple ownerships. Therefore, we only need to model the provenance data and enable path queries on the provenance graph, which are independent from the formalization of the ReBAC language. We choose the Resource Description Framework (RDF) [20] data presentation to express the provenance data, as it naturally supports a directed graph structure. Standard RDF query languages, such as SPARQL [29], can be employed to query over provenance data stored in RDF triples. The provenance-assisted ReBAC model can be also extended to enable provenance-based access control [28], which means authorization decisions are made directly based on provenance information, in addition to relationships. In this case, we have to formalize provenance information in the existing ReBAC policy language. One possible way of such formalization can be achieved by extending the XACML language [23], as described in [3, 24, 25].

Querying over a provenance graph introduces additional computational overhead. The performance of such query depends on the shape of the provenance graph. As the system evolves, the provenance graph eventually grows in both width and depth. A performance study of a similar problem was conducted in [24]. However, the evaluation and optimization of the provenance graph query is beyond the scope of this paper. The nature of the provenance graph query indicates that, for the sole purpose of distinguishing multiple ownerships, the typed parameter approach in the RPPM^2 model is simpler and less costly. However, the provenance-based approach offers greater expressive power and richer information, which can be further utilized for many other purposes. Provenance-based access control is definitely among one of them. Multiple-level cascading revocation is another usage example that cannot be facilitated through typed parameter but can be realized via provenance information.

4.3 Comparison with Existing Administrative ReBAC Models

In [15], the proposed ReBAC model is designed to mimic the authorization process in Facebook-style OSNs. It models the communication protocol of friendship initiation and termination, and provides a discretionary policy administration based on resource ownership, which has limited expressive power compared with later proposals. The access control framework introduced by Carminati *et al.* features authorization, administrative and filtering policies in ontology-based representations [4]. However, this framework does not address administrative activities related to entities and relationships. Cheng *et al.*'s URRAC model proposes to arbitrate administrative activities using ReBAC, but it does not elaborate the details about administration [7]. These proposals are mostly targeted for OSN systems and are not applicable to general-purpose computing systems.

The RPPM^2 model [33] and the ReBAC implementation for OpenMRS [30] are two administrative ReBAC models appeared lately in literature. RPPM^2 is a comprehensive model that addresses administrative operations on entities, relationships as well as authorization policies. The main contribution of Rizvi *et al.*'s administrative model is on incorporating ReBAC in a production-scale system that originally uses RBAC. In particular, it mainly focuses on one type of administrative operations: adding or deleting relationship edges. In addition to regulating who can perform the operation, their hybrid logic-based policy language also captures the applicability of the operation, aiming to completely preserve the security constraints.

Our work follows the policy language defined in RPPM^2 and extends it to capture the issues we found in our use cases. Our models seek to preserve the global integrity constraints, address the cascading revocation as well as the multi-ownership issue. We also aim to apply ReBAC to applications beyond OSNs, thus using MT-RBAC, an access control model for collaborative cloud systems, for demonstration.

5. EVALUATION

The goal of the evaluation is to decide whether algorithm 1 can efficiently determine the set of dependent edges of a specific edge being removed. We implemented algorithm 1 using Java with version 1.8.0_60. The experiments are performed on an Intel Ci7 machine with 8 cores, 2.53 GHz and 16GB RAM, running Ubuntu 14.04.1 LTS (Trusty).

5.1 Implementation and Input Instance Generation

In our code, we represented $\Phi_{dependency}$, described in section 4, as java *HashMap*, where the key is the id of an edge and the value is the *CascadingElement* object. A *CascadingElement* has two variables: a java *Queue* called *path* and a java *Set* called *rSet*, which are the representation of Path and R_d of algorithm 1, respectively. Edges are

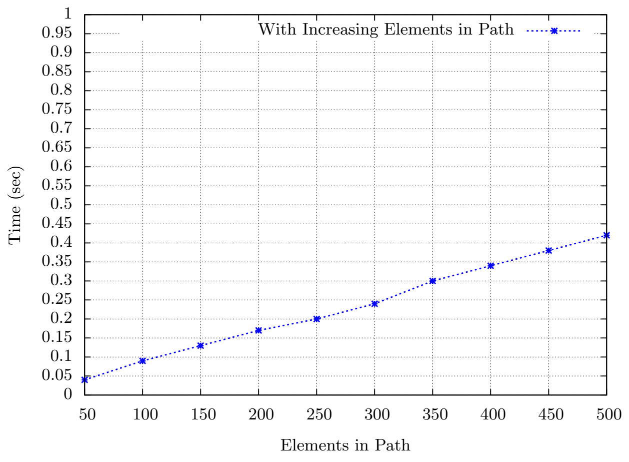

Figure 7: Dependent-Edge discover time for experiment 1

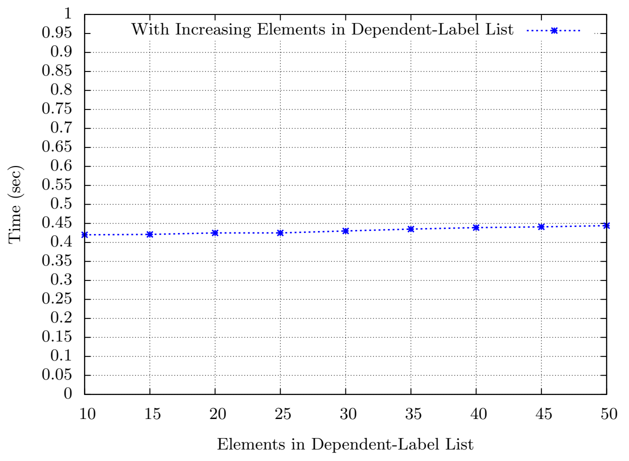

Figure 8: Dependent-Edge discover time for experiment 2

stored in a *Set* called E, where each edge is an object that has an id, id of node$_1$, id of node$_2$, and a label.

We synthetically generate problem instances for our evaluation. We believe the values of the different parameters are sufficient for a medium sized organization. We considered 100 types and 50 labels. Then we created the permissible relationship graph $G_{PR} = (V_{PR}, E_{PR})$, where $|E_{PR}| = 1000$. Based on G_{PR}, we created the well-formed system graph $G = (V, E)$, where the size of V equals to 10000, each type has 100 instance, and the size of E is 50000.

5.2 Evaluation Result

We populate $\Phi_{dependency}$ for 100 different edges. For each edge in $\Phi_{dependency}$, in the *first experiment*, we vary the size of *path* from 50 to 500, where we ensure that there is at least one valid path in the graph in the order of edges in *path*. Note that, *path* can contain same relationship label multiple times. However, each of them should be visited once and in the order it is specified. Also, for each element in $\Phi_{dependency}$, we fix the size of *rSet* to 10. Then, for each element in $\Phi_{dependency}$, we execute the algorithm and record the average time. We also repeat the whole process of first experiment for 15 times and record the average. From the results shown in Figure 7, we can see that for 50 elements in the *path*, the average time to discover the set is very minimal (< 0.05 sec); for 500 elements it is 0.415 sec. In the *second experiment*, we set the size of *path* to 500 elements and vary the size of $rSet$ from 10 to 50. We record the average execution time of the algorithm. Similar to the first experiment, this experiment is also repeatedly conducted for 15 times and the average time is recorded. The results shown in Figure 8 indicate that time does not vary that much with an increase in the size of $rSet$. Since each new element in *path* increases one more round of recursion call, its impact on the running time of dependent-edge discovery is larger than that of the size of $rSet$.

6. RELATED WORKS

With the emergence and growth of OSNs over the last decade, new access control schemes have been called upon to address the issues that conventional access control approaches cannot properly address. In [5, 6], Carminati *et al.* developed a pioneering work for access control in OSNs, where authorization policies are specified in terms of type, depth and trust level of the relationships. This type of access control, namely relationship-based access control, exploits relationships between users and resources as the basis for authorization decision. Since the term was invented in [17], ReBAC has undergone considerable development.

In [15], Fong *et al.* formalized the privacy preservation mechanism in Facebook-style OSNs into a two-stage procedure. In addition to modeling the Facebook-style policy predicates, such as "only-me", "only-friends", "friends-of-friends", and "everyone", the model is capable of expressing some topology-based policies, including "degree of separation", "clique", and so on, which are not available in Facebook and other well-known OSN systems.

Modal logic and its extension, hybrid logic, have been used for expressing ReBAC policies in [2, 14, 16]. Fong *et al.* formulated a ReBAC model and introduced a modal logic language to compose complex relationship-based policies [14]. In a subsequent work, the modal logic proposed earlier was extended and improved with more expressive power [16]. In [2], the authors adopted a hybrid logic to achieve better efficiency and greater flexibility in policy specification. Pang *et al.* also adopted a hybrid logic approach to formulate access control policies in their model for OSNs, where public information is also exploited for regulating access [27].

Cheng *et al.* proposed a series of three ReBAC models for OSNs that utilize regular expression-based notations to specify policies. In [8], a sequence of relationship edge labels forms a path expression, which can be interpreted as regular expression. If there is a path between the access requester and the resource owner satisfying the path expression in the policy, the requested access is granted. A path-checking algorithm is used to determine the existence of such qualified path. The authors subsequently extended the model to incorporate resources and actions to social graph, and track relationships among users and resources in addition to relationships between users and users [7]. Since multiple policies may be applicable to single resource, conflict resolution policies were introduced to arbitrate authorization policies. In another subsequent work, the model proposed in [8] was extended to exploit attribute information of users and relationships for the purpose of access control, enabling a richer policy language [9].

Several attempts have been made to adopt ReBAC in domains beyond OSNs. The proposed work in [14] was aimed for general-purpose computing systems, with an example scenario of electronic health records. The RPPM model recently proposed by Crampton *et al.* is a variant of ReBAC model for general-purpose computing systems [11], where policies are expressed in terms of path conditions similar to path expressions in [8]. The RPPM model introduced authorization principals, which are analogous to roles in RBAC. In [33], the author developed the RPPM^2 model, a direct extension of RPPM, that addresses administrative operations in ReBAC. Rivzi *et al.*'s OpenMRS access control mechanism [30] also features an administrative model for ReBAC, addressing how to enable users to manage access control relationships safely. Our work in this paper is based on these two administrative models, building on the policy language in [33] and the design objectives identified in [30].

7. CONCLUSION

In this paper, we present a family of three administration ReBAC models based on the policy language offered in the RPPM and RPPM^2 models. Our models cover the administrative operations on edges. In addition to regulating who can perform administrative operations, we identify three problems that were rarely discussed in the literature of ReBAC: integrity constraints, cascading revocation, and multi-ownership of edges. We adopt and modify the concept of enabling precondition and applicability precondition from [30] to express path conditions and integrity constraints. The cascading revocation is achieved by using our proposed depth-first search-based algorithm, which discovers the dependent edges that need to be removed. An evaluation is conducted to show the effectiveness and efficiency of the algorithm. The multi-ownership of edges can be properly distinguished by typed parameters or provenance data incorporated in the models. We demonstrate that our ReBAC models are capable of expressing policies in MT-RBAC.

Still at its early stage, ReBAC administration will remain a new research frontier for some time to come. Towards the adoption of ReBAC in various other application domains, many new opportunities will be identified along this direction. We will investigate these problems and enrich the ReBAC models with greater flexibility and more expressive power. One of the possible directions for us is to extend our work to address policy administration, which is very critical with essential decentralized components for ReBAC systems.

Acknowledgments

This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481.

8. REFERENCES

[1] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *IEEE TKDE*, 9(1):85–101, 1997.

[2] G. Bruns, P. W. Fong, I. Siahaan, and M. Huth. Relationship-based access control: its expression and enforcement through hybrid logic. In *Proceedings of the second ACM CODASPY*, pages 117–124. ACM, 2012.

[3] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. A language for provenance access control. In *Proceedings of the first ACM CODASPY*, pages 133–144. ACM, 2011.

[4] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. A semantic web based framework for social network access control. In *Proceedings of the 14th ACM SACMAT*, pages 177–186. ACM, 2009.

[5] B. Carminati, E. Ferrari, and A. Perego. Rule-based access control for social networks. In *OTM 2006 Workshops*, pages 1734–1744. Springer, 2006.

[6] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM TISSEC*, 13(1):6, 2009.

[7] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *PASSAT 2012*, pages 646–655. IEEE, 2012.

[8] Y. Cheng, J. Park, and R. Sandhu. A user-to-user relationship-based access control model for online social networks. In *DBSec XXVI*, pages 8–24. Springer, 2012.

[9] Y. Cheng, J. Park, and R. Sandhu. Attribute-aware relationship-based access control for online social networks. In *DBSec XXVIII*, pages 292–306. Springer, 2014.

[10] J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *ACM TISSEC*, 6(2):201–231, 2003.

[11] J. Crampton and J. Sellwood. Path conditions and principal matching: a new approach to access control. In *Proceedings of the 19th ACM SACMAT*, pages 187–198. ACM, 2014.

[12] R. Fagin. On an authorization mechanism. *ACM TODS*, 3(3):310–319, 1978.

[13] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.

[14] P. W. Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM CODASPY*, pages 191–202. ACM, 2011.

[15] P. W. Fong, M. Anwar, and Z. Zhao. A privacy preservation model for Facebook-style social network systems. In *Computer Security–ESORICS 2009*, pages 303–320. Springer, 2009.

[16] P. W. Fong and I. Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM SACMAT*, pages 51–60. ACM, 2011.

[17] C. Gates. Access control requirements for Web 2.0 security and privacy. In *Workshop on Web 2.0 Security & Privacy (W2SP)*, 2007.

[18] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM TODS*, 1(3):242–255, 1976.

[19] W. Kuijper and V. Ermolaev. Sorting out role based access control. In *Proceedings of the 19th ACM SACMAT*, pages 63–74. ACM, 2014.

[20] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. 1999.

[21] N. Li and Z. Mao. Administration in role-based access control. In *Proceedings of the 2nd ACM AsiaCCS*, pages 127–138. ACM, 2007.

[22] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, et al. The open provenance model core specification (v1. 1). *Future generation computer systems*, 27(6):743–756, 2011.

[23] T. Moses et al. Extensible access control markup language (XACML) version 2.0. *Oasis Standard*, 200502, 2005.

[24] D. Nguyen, J. Park, and R. Sandhu. A provenance-based access control model for dynamic separation of duties. In *PST 2013*, pages 247–256. IEEE, 2013.

[25] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han. An access control language for a general provenance model. In *Secure Data Management*, pages 68–88. Springer, 2009.

[26] S. Oh and R. Sandhu. A model for role administration using organization structure. In *Proceedings of the seventh ACM SACMAT*, pages 155–162. ACM, 2002.

[27] J. Pang and Y. Zhang. A new access control scheme for Facebook-style social networks. In *ARES 2014*, pages 1–10. IEEE Computer Society, 2014.

[28] J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *PST 2012*, pages 137–144. IEEE, 2012.

[29] E. Prud'Hommeaux, A. Seaborne, et al. SPARQL query language for RDF. *W3C recommendation*, 15, 2008.

[30] S. Z. R. Rizvi, P. W. Fong, J. Crampton, and J. Sellwood. Relationship-based access control for an open-source medical records system. In *Proceedings of the 20th ACM SACMAT*, pages 113–124. ACM, 2015.

[31] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM TISSEC*, 2(1):105–135, 1999.

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

[33] S. D. Stoller. An administrative model for relationship-based access control. In *DBSec XXIX*, pages 53–68. Springer, 2015.

[34] B. Tang, Q. Li, and R. Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *PST 2013*, pages 229–238. IEEE, 2013.

[35] H. Wang and S. L. Osborn. An administrative model for role graphs. In *DBSec XVII*, pages 302–315.
Springer, 2004.