Towards Trustworthy Delegation in Role-Based Access Control Model *

Manachai Toahchoodee, Xing Xie, and Indrakshi Ray

Department of Computer Science
Colorado State University
Fort Collins CO 80523-1873
{toahchoo,xing,iray}@cs.colostate.edu

Abstract. The need to delegate, which allows the temporary grant or transfer of access rights, arise in many applications. Although a lot of research appears in extending Role-Based Access Control (RBAC) to support delegation, not much appears on providing a formal basis for choosing delegatees. We provide an approach that allows one to assess the trustworthiness of potential delegatees in the context of the task that is to be delegated. It is also important to ensure that the choice of the delegatee does not cause any security policy violation. Towards this end, we show how to formally analyze the application using existing SAT solvers to get assurance that our choice of delegatee does not cause a security breach. Once the process of choosing delegatee can be formalized, it will be possible to automate delegation and use it for real-time applications.

1 Introduction

Role-Based Access Control (RBAC) is the de facto access control model for commercial organizations primarily because it is policy neutral and simplifies access control management. Since its conception, RBAC has evolved in various ways to meet the demands of various applications. One such extension is with regards to incorporating the notion of delegation. Delegation allows a user or a role to grant or transfer privileges to other users or roles. This makes it possible for organizations to continue functioning when some user is temporarily unavailable.

Although a lot of research appears in the area of delegation [3, 14, 15, 25, 23, 26], not much appears in formalizing the basis on which a delegator selects a delegatee. The choice of a delegatee should be determined by two factors. First, the trustworthiness of an entity must be taken into account while considering it as a delegatee. This requires that the privileges should not be delegated to another user who the delegator does not consider trustworthy. This factor becomes even more critical in the presence of delegation chains. Second, choosing a delegatee should not introduce any security policy violation. For example, choosing

* This work was supported in part by AFOSR under contract number FA9550-07-1-0042.

a specific delegatee may cause a violation in separation of duty constraints. We address these two factors.

The first factor requires evaluation of trustworthiness of candidates. Trust is a relationship between a truster and a trustee and it is dependent on a given *task*. The truster's trust for a trustee, with respect to a given task, depends on several factors, namely, *properties*, *experiences*, and *recommendations*. Properties are verifiable characteristics of the trustee. Experiences correspond to the past work experience of the trustee. Recommendations are the information that the truster obtains from reputable sources about the trustee. We show how to quantify these factors and assess the trustworthiness of an entity before designating him as the delegatee. We also show how trustworthiness of an entity can be used to decide and reason about delegation chains.

Sometimes a truster may not have enough information about a trustee with respect to a given task that will allow him to trust the trustee. However, information about related tasks may be available. We formalize the relationships among the various tasks using the concept of *task graphs*. Task graphs are directed acyclic graphs where the nodes correspond to the different tasks in an organization and the edges correspond to generalization/specialization and composition relationships. The labels on the edges give the degree of similarity between the different tasks. With the help of the task graphs, trust information of related tasks can be used to extrapolate the trust value for the given task.

Once a potential delegatee has been selected based on trustworthiness, we must ensure that this selection does not cause a security breach. We advocate the use of Alloy [9] for checking security policy violation. Alloy is a modeling language capable of expressing complex structural constraints and behavior. Alloy is supported by an automated constraint solver called Alloy Analyzer that searches instances of the model to check for satisfaction of system properties. The model is automatically translated into a Boolean expression, which is analyzed by SAT solvers embedded within the Alloy Analyzer. A user-specified scope on the model elements bounds the domain, making it possible to create finite Boolean formulas that can be evaluated by the SAT-solver. When a property does not hold, a counter example is produced that demonstrates how it has been violated. It has been successfully used in the modeling and analysis of real-world systems [8, 20].

The paper is organized as follows. Section 2 describes some of the important work related to delegation and trust modeling. Section 3 presents our trust model and how to assess trustworthiness of entities with respect to a given task in a quantitative manner. Section 4 shows how trustworthiness of entities can be used to decide on the levels of delegation. Section 5 discusses how to compute trustworthiness of entities with respect to a given task when we do not have any information about the entity with respect to the given task. Section 6 illustrates how trust computation is performed for potential delegatees. Section 7 provides an approach using Alloy that evaluates the potential delegatees who satisfy the security policies. Section 8 concludes the paper with some pointers to future directions.

2 Related Work

One of the early works on delegation is by Barka and Sandhu [3] who proposed Permission-Based Delegation Model (PBDM) supporting permission and role delegations and revocations. The PBDM was refined subsequently by Zhang et al. [26] into three versions, namely, PBDM0, PBDM1, and PBDM2. RDM2000 [25] is an extension of PBDM0 which provides rules for delegation in the role hierarchy and identifies the prerequisites that must be satisfied by delegatees. Joshi et al. [14, 15] focus on the relationship between delegation and role hierarchies in the context of Generalized Temporal Role-Based Access Control (GTRBAC) model. Crampton et al. [5] focussed on the workflow satisfiability problem (WSP) in the context of delegation. Workflow satisfiability in the context of delegation has also been addressed by Wang and Li [22]. The authors prove that WSP in the context of their R^2BAC model is an NP-complete problem. In a subsequent work [23], Wang and Li formalize the notion of secure delegation.

One of the most important work in trust modeling is by Jøsang [10, 11] where he claimed that trust is a relationship between two entities on a specific *statement* and is represented using degrees of *belief* b, *disbelief* d and *uncertainty* u. A statement describes a particular type of trust. Within a specific statement, Jøsang called the triple {b, d, u} as an opinion $\omega = \langle b, d, u \rangle$ representing the confidence to trust that declaration. Recommendation also plays a part in increasing the decision confidence. Jøsang utilized *subjective logic* to define *recommendation* and *consensus* formulae in order to take into account multiple subjective views on the same statement. Recommendation is when entity A asks another entity B for recommendation about how B trusts statement S, while consensus is the cumulative result caused by A and B in trusting S. Jøsang also formalized the notion of trust chains when recommenders indirectly provide input for a particular statement. In subsequent works[12], Jøsang added a new component *base rate* a, where $a \in [0, 1]$. The base rate gives the default trust value in the absence of information about a given entity. In another work [13], they show how to specify trust networks consisting of multiple paths between the trusted parties and provide a practical method for analyzing and deriving measures of trust in such environments.

Although a lot of research has been done in the context of access controls for open systems [2, 4, 6, 7, 16–18, 21], we describe only the ones that are closely related to this work. Chakraborty et al. [4] proposed a Trust Based Access Control (TrustBAC) model where the assignment of users to roles depended on their trust values which can range from -1 to 1. The authors propose three factors, namely, *knowledge* W_K, *experience* W_E, and *recommendation* W_R, that impact a user's trustworthiness and show how to evaluate them. The trust value also takes into account history information in trustworthiness computation. Ray et al. [18] propose a trust model where the notion of trust contexts were formalized. The relationships among different contexts were represented using *context graph*. *Specialization* and *composition* are two kinds of relationships in context graph, where the labels on the graph indicate the degree of similarity between the contexts.

A lot of work also appears in the use of Alloy for analyzing security policies. Zao et al. [24] show how to model RBAC and Bell-Lapadula using Alloy. Schaad et al. [19] model user-role assignment, role-permission assignment, role hierarchy, and separation of duty features of RBAC extension using Alloy, and also describe how to detect conflicts.

3 Trust Modeling and Computation

Delegator refers to the role or user whose privileges are being transferred or granted to another role or user and the recipient of the privileges is termed *delegatee*. We show how the delegator can compute the trustworthiness of various entities in the context of the task that he is about to delegate.

Trust is a relationship between a truster and trustee with respect to a given context. The context in the case of delegation is the task for which delegation is needed. Trust relationship for a given context depends on three factors: *properties*, *experiences* and *recommendations*. Properties are verifiable characteristics of the trustee. For instance, it may be the role and credentials possessed by the trustee. Experiences are the past interactions that the truster had with the trustee. Recommendations are provided by third-parties whom the truster trusts about the capabilities of the trustee. In the following, we describe how the trust relationship is quantified.

3.1 Quantifying Properties

Properties depend on the attributes of the entity and also the role associated with it.

Measuring Necessary Attributes $\mathcal{A}$

Every task in an organization requires some attributes of the user. For example, the task of performing surgery requires the user to be a certified surgeon. A task may require one or more attributes. The information about user attributes is contained in the credentials belonging to the user. Credentials are unforgeable and verifiable. Measuring necessary attributes requires evaluating what percentage of the necessary attributes are possessed by the user.

Let the set of attributes needed for task T_i be denoted by TA_i where $TA_i = \{a_{i1}, a_{i2}, \ldots, a_{in}\}$. Let wa_{i1}, wa_{i2}, ..., wa_{in} be the weights of attributes a_{i1}, a_{i2}, ..., a_{in} respectively. The weights of the attributes indicate their relative importance with respect to task T_i and $\Sigma_{r=1}^{n} wa_{ir} = 1$. Each user profile contains the credentials possessed by the user. Let the set of all attributes possessed by the user U_j be given by UA_j, where $UA_j = \{a_{j1}, a_{j2}, \ldots, a_{jm}\}$. Let $p = |TA_i \cap UA_j|$. The attribute value for user j with respect to task T_i, denoted by $\mathcal{A}_{ij}$, is calculated as follows: $\mathcal{A}_{ij} = \Sigma_{k=1}^{p} wa_k$ where wa_k $(1 \leq k \leq p)$ is the weight associated with attribute a_k and $a_k \in TA_i \cap UA_j$.

Measuring Role Attribute $\mathcal{R}$

The roles in the organization are arranged in the form of a hierarchy. The hierarchy can be represented as a labeled directed acyclic graph where the nodes

represent the roles and the edges denote the hierarchical relationship. Note that, edges are drawn only for direct senior and junior relationship; transitive edges are not explicitly added. The edges in the hierarchy are labeled with a number in the range (0,1] which indicates the closeness relationship between the roles. A number close to 0 indicates that the two roles are very distant, whereas a number close to 1 denotes that the roles are very close. We assume that the assignment of the numbers is done by the system administrator who has knowledge about the relationships between roles. If there is a path between role i and role j, the closeness relationship, denoted by $dist(r_i, r_j)$, is calculated by taking the product of all the edges constituting this path. Note that, if there are multiple paths connecting role i and role j, both the paths should give the same value. Otherwise, the role graph is said to be inconsistent. The formal definition of the role graph appears below.

[**Weighted Role Hierarchy Graph**] Weighted role hierarchy graph, denoted by $WRH = (V, A)$, is a weighted directed acyclic graph where V is a set of nodes corresponding to the roles, and A is a set of arcs corresponding to the hierarchical relationship; $(v_i, v_j) \in A$ indicates that role v_j is directly senior to the role v_i. The weight of the edge (v_i, v_j), denoted by $w(v_i, v_j)$, is a number in the range (0,1] that gives a measure of the closeness of the two roles.

Each task T_i is associated with a set of roles TR_i who are authorized to execute this task. The roles associated with a task include roles who have the direct permission to execute those tasks, as well as those authorized by virtue of role hierarchy. Each user U_j also has a set of roles UR_j assigned to him. We choose the role belonging to the user that is closest to some role associated with the task. The distance between these two roles gives the role attribute $\mathcal{R}_{ij}$ of user U_j with respect to task T_i.

Computing the Properties Value

Some organizations may give greater importance to the role factor, whereas others may consider attribute factor to be more useful. Let w_a and w_r be the weights assigned to attributes and roles respectively, where $w_a, w_r \in [0, 1]$ and $w_a + w_r = 1$. The exact values of w_a and w_r will be decided by the organization's policies. We use these weights to compute the property value $\mathcal{P}_{ij}$ of user U_j with respect to task T_i: $\mathcal{P}_{ij} = w_a * \mathcal{A}_{ij} + w_r * \mathcal{R}_{ij}$

3.2 Quantifying Experience

Experience constitutes an important factor in delegation. A delegator is more likely going to choose a candidate as a delegatee if the delegatee has prior experience of doing the task. Two factors contribute towards experience. One factor is when the task was performed, and the second factor is how well the task was performed. Note that, information about these factors is stored in the users' profile, UP. Events that have occurred in the recent past have more influence than that occurred in the distant past. To accommodate this, we give the most recent slot has the highest weight and the most distant slot has the lowest one. For each time slot t_k, we get the value for performance p_i. Recall that, performance on the task measures how well the task has been performed. The performance on

Algorithm 1 Measuring Experience

Input: No. of slots n, User Profile UP_j
Output: $\mathcal{P}_{ij}$
Procedure:

$performance = 0$
for all $k : 1 \leq k \leq n$ **do**
  $weight\_slot_k = k$
end for
$total\_weight = n(n+1)/2$
for all $k : 1 \leq k \leq n$ **do**
  $w_k = (2 * k)/(n(n+1))$
end for
for all $k : 1 \leq k \leq n$ **do**
  $experience = experience + w_k * p_k$
end for
RETURN *experience*

the task can be graded on a scale of [0,1]. A value closer to 0 indicates poor performance, while that closer to 1 indicates excellent performance. Not performing the task in a slot, gives a performance value equal to 0. Algorithm 1 shows how to assign weights to the various time slots and evaluate the experience. Sometimes the past experience may not exactly match the the task, but is related to it. We show how to extrapolate the trust value in such cases in Section 5.

3.3 Quantifying Recommendation

A truster may obtain recommendation from one or more recommenders about the trustee with respect to its ability to perform the given task. In order to quantify the recommendation obtained from each recommender, we need to evaluate two factors. First, we need to obtain the trust value that the truster has with respect to the recommender providing recommendation about the trustee with respect to the given task. If the recommender is sufficiently trusted, then we need to get from him the recommendation value for the trustee. Algorithm 2 shows how to compute the recommendation component.

3.4 Computing Trustworthiness

Trust, with respect to a given task T_i for user U_j, denoted by $\mathcal{T}_{ij}$, depends on three factors, namely, properties $\mathcal{P}_{ij}$, experiences $\mathcal{E}_{ij}$, and recommendations, $\mathcal{R}_{ij}$. The exact weight assigned to each factor will be decided by the organization. Let w_p, w_e, and w_r be the weights assigned to the three factors respectively where $w_p, w_e, w_r \in [0, 1]$ and $w_p + w_e + w_r = 1$. $\mathcal{T}_{ij}$ is given by, $\mathcal{T}_{ij} = w_p * \mathcal{P}_{ij} + w_e * \mathcal{E}_{ij} + w_r * \mathcal{R}_{ij}$. Note that $\mathcal{T}_{ij}$ will evaluate to some value in the range [0,1]. The delegator can choose a threshold value for trust $\mathcal{H}$. If $\mathcal{H} \leq \mathcal{T}_{ij}$, then user U_j can be a potential delegatee.

Algorithm 2 Measuring Recommendation

Input: Sequence of recommendations for user $U_j =< r_{1j}, r_{2j}, \ldots, r_{mj} >$, sequence of trust values for recommenders $= < t_1, t_2, \ldots, t_m >$
Output: $\mathcal{R}_{ij}$
Procedure:

$reco = 0; total = 0$
for all $k : 1 \leq k \leq m$ **do**
  $reco = reco + t_k * r_{kj}$
end for
for all $k : 1 \leq k \leq m$ **do**
  $total = total + t_k$
end for
$reco = reco/total$
RETURN $reco$

4 Using Trust Values in Delegation Chains

The privilege that a user receives can be further delegated resulting in what is known as a delegation chain. In some cases, we may want to limit the level of delegation. This level of delegation can be decided by the trustworthiness of the users involved in the delegation chain. Thus, delegation chain is dependent on the concept of trust chains. Trust chains are formalized using the concept of trust graphs defined below.

[**Trust Graph**] Let $TG =< N, E >$ be the directed acyclic graph that represents trust relationship for a given context. The set of nodes N correspond to the entities in the system, and the set of edges E represent the trust relationship between the nodes. The edge (n_i, n_j) represents the trust relationship that node n_i has for node n_j with respect to the given task. The weight of the edge, denoted by $w(n_i, n_j)$, where $0 < w(n_i, n_j) \leq 1$, represents the trust value that node n_i has with respect to node n_j. Note that, the absence of a trust relationship between nodes n_r and n_s is indicated by the missing edge (n_r, n_s).

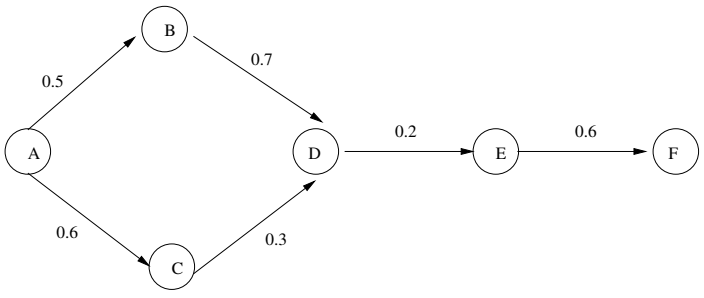

Fig. 1. Example of a Trust Graph

Given a trust graph, we define two types of operators to compute transitive trust. One is the sequential operator, and the other is the parallel operator. Sequential and parallel operators and their desirable properties have been proposed by Agudo et al. [1].

[**Sequential Operator**] Sequential operator, denoted by $\bigotimes$, is a binary operator that takes as input two trust values and returns a trust value that is the product of the two input values. Formally, $\bigotimes : [0,1] \times [0,1] \rightarrow [0,1]$. The sequential operator is used for computing the transitive trust value in a single path in the trust graph. Algorithm 3 gives the description of how transitive trust is computed. For instance, to compute the transitive trust that D has about F with respect to the given context is the product of 0.2 and 0.6 which equals 0.12. The

Algorithm 3 Computing Transitive Trust in a Single Path

Input: Trust Path $(n_1, n_2, \ldots, n_k)$
Output: Transitive trust between nodes n_1 and n_k
Procedure:

$trans\_trust = 0$
for all $i : 1 \leq i \leq (k-2)$ **do**
$trans\_trust = trans\_trust * w(n_i, n_{i+1}) \bigotimes w(n_{i+1}, n_{i+2})$
end for
RETURN $trans\_trust$

sequential operator is not adequate for calculating transitive trust when multiple paths are involved. For example, in Figure 1, computing transitive trust that A has about E using the path (A, B, D, E) gives a different value than that obtained using the path (A, C, D, E). The value is 0.07 for the path (A, B, D, E) and it is 0.036 for the path (A, C, D, E). Such differences are reconciled using the parallel operator. The parallel operator becomes useful when there are multiple paths from one node to another.

[**Parallel Operator**] Parallel operator, denoted by $\bigoplus$, is a binary operator that takes as input two trust values and returns a trust value that is the minimum of the two input values. Formally, $\bigoplus : [0,1] \times [0,1] \rightarrow [0,1]$. Algorithm 4 shows how to compute transitive trust when the source and destination are connected by parallel paths. The transitive trust that A has for D, computed using this algorithm, equals 0.18.

The delegator can specify an acceptable level of trust to support delegation chains. Delegation is disallowed if the transitive trust value computed from the chain of delegation is below this minimum threshold.

5 Extrapolating Trust Values

Sometimes the delegator may not have enough information to assess the trustworthiness of a user with respect to some given task. Although the user is not associated with a given task, it is possible that he has done some related tasks.

Algorithm 4 Computing Transitive Trust in the Presence of Multiple Paths

Input: Trust Paths $(n_1, n_{2_1}, \ldots, n_{(k-1)_1}, n_k)$, $(n_1, n_{2_2}, \ldots, n_{(k-1)_2}, n_k)$, $\ldots$, $(n_1, n_{2_j}, \ldots, n_{(k-1)_j}, n_k)$
Output: Transitive trust between nodes n_1 and n_k
Procedure:

```
min = 1;
for all l : 1 ≤ l ≤ j do
  trans_trust_l = 0
end for
for all l : 1 ≤ l ≤ j do
  for all i : 1 ≤ i ≤ (k − 2) do
    trans_trust_l = trans_trust_l + w(n_i, n_{i+1}) ⊗ w(n_{i+1}, n_{i+2})
  end for
end for
for all l : 1 ≤ l ≤ j do
  if trans_trust_l < min then
    min = trans_trust_l
  end if
end for
RETURN min
```

To handle such scenarios, we define the different relationships that can exist among the tasks in an organization.

Specialization Relation

Different tasks may be related by the generalization/specialization relationship which is anti-symmetric and transitive. We use the notation $\mathcal{T}_i \subset \mathcal{T}_j$ to indicate that task $\mathcal{T}_i$ ($\mathcal{T}_j$) is a generalization (specialization) of task $\mathcal{T}_j$ ($\mathcal{T}_i$). For instance, *Surgery Treatment* $\subset$ *Heart Bypass Surgery Treatment* and *Heart Treatment* $\subset$ *Heart Bypass Surgery Treatment*. However, the degree of specialization is different in the two cases. The *degree of specialization* captures this difference. The degree of specialization is denoted as a fraction whose value is determined using domain knowledge.

Composition Relation

Sometimes tasks can be linked together using the composition relation. A task can either be *elementary* or *composite*. An elementary task is one which cannot be subdivided into other tasks, whereas a composite task is one that is composed from other tasks. The individual tasks that form a composite one are referred to as the *component* tasks. A component task can either be composite or elementary. We use the notation $\mathcal{T}_i \ll \mathcal{T}_j$ to indicate that the task $\mathcal{T}_i$ is a component of task $\mathcal{T}_j$. For instance, we may have the component tasks *operation* and *medication* that are part of the composite task *Catheter-assisted Procedures*. This is denoted as *operation* $\ll$ *Catheter-assisted Procedures*.

Sometimes a composite task $\mathcal{T}_i$ may be composed from the individual tasks $\mathcal{T}_j$, $\mathcal{T}_k$ and $\mathcal{T}_m$. All these tasks may not contribute equally to form $\mathcal{T}_i$. The *degree of composition* captures this idea. A degree of composition is associated with each composition relation. Since two tasks related by composition will not be

exactly identical, the degree of composition is denoted as a fraction. The sum of all these fractions equals one if $\mathcal{T}_i$ is composed of $\mathcal{T}_j$, $\mathcal{T}_k$, and $\mathcal{T}_m$ only. If $\mathcal{T}_i$ is composed of $\mathcal{T}_j$, $\mathcal{T}_k$, and $\mathcal{T}_m$ and also other component contexts, then the sum of fractions associated with $\mathcal{T}_j$, $\mathcal{T}_k$, and $\mathcal{T}_m$ must be equal to or less than one. The exact value of the fraction representing the degree of composition will be determined by domain knowledge.

The generalization/specialization and composition relations are formally specified using the notion of *task graphs* defined below.

[**Task Graph**] A task graph $\mathcal{TG} = \langle \mathcal{N}, \mathcal{E}_c \cup \mathcal{E}_s \rangle$ is a weighted directed acyclic graph satisfying the following conditions.

- $\mathcal{N}$ is a set of nodes where each node n_i is associated with a task $\mathcal{T}_i$.
- The set of edges in the graph can be partitioned into two sets $\mathcal{E}_c$ and $\mathcal{E}_s$. For each edge $(n_i,n_j) \in \mathcal{E}_c$, the task $\mathcal{T}_i$ corresponding to node n_i is a component of the task $\mathcal{T}_j$ corresponding to node n_j. The weight of the edge (n_i, n_j), denoted by $w(n_i, n_j)$, indicates the percentage of component task that makes up the composite one. For each edge $(n_i,n_j) \in \mathcal{E}_s$, the task $\mathcal{T}_i$ corresponding to node n_i is a specialization of task $\mathcal{T}_j$ corresponding to node n_j. The weight of the edge (n_i, n_j), denoted by $w(n_i, n_j)$, indicates the degree of specialization.

5.1 Computing the Degree of Specialization and Composition

Consider two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ where $\mathcal{T}_i \subset \mathcal{T}_j$, that is, $\mathcal{T}_j$ is a specialization of $\mathcal{T}_i$. The degree of specialization is computed as follows. Let n_i, n_j be the nodes corresponding to tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ in the weighted graph. Let the path from n_i to n_j consisting of specialization edges be denoted as $(n_i, n_{i+1}, n_{i+2}, \ldots, n_{j-1}, n_j)$. The degree of specialization $= \Pi_{p=i}^{j-1} w(n_p, n_{p+1})$. This corresponds to our notion that the similarity decreases as the length of the path from the generalized node to the specialized node increases. Note that, in real world there may be multiple paths from $\mathcal{T}_i$ to $\mathcal{T}_j$. In such cases, it is important that the degree of specialization yield the same values when any of these paths are used for computation.

Consider two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ such that $\mathcal{T}_j$ is a component of $\mathcal{T}_i$. Degree of composition captures what portion of $\mathcal{T}_i$ is made up of $\mathcal{T}_j$. The degree of composition is computed as follows. Let n_i, n_j be the nodes corresponding to contexts $\mathcal{T}_i$ and $\mathcal{T}_j$ in the task graph. Let there be m paths consisting of composition edges from n_i to n_j. Let the qth path $(1 \leq q \leq m)$ from n_i to n_j be denoted as $(n_i, n_{i_q+1}, n_{i_q+2}, \ldots, n_{j_q-1}, n_j)$. The degree of composition $= \Sigma_{q=1}^{m} (w(n_i, n_{i_q+1}) \times w(n_{j_q-1}, n_j) \times \Pi_{p=i_q+1}^{j_q-2} w(n_p, n_{p+1}))$.

6 Trust Computation for Example Application

Consider a small healthcare organization that has six roles, namely, *senior doctor*, *junior doctor*, *cardiologist*, *surgeon*, *physician's assistant* and *patient*. *senior doctor* is senior to *junior doctor*, and *junior doctor* is senior to *cardiologist* and

physician's assistant. Allen and Miller are assigned to *senior doctor*, Bell and Nelson are assigned to *junior doctor*, Cox is assigned to *cardiologist*, and Davis is assigned to *physician's assistant*. Allen is also assigned to *surgeon* and Evans is assigned to *patient*. Allen is the assigned surgeon for performing Coronary Artery Disease Angioplasty (CAD type A) surgery on patient Evans. Since Allen has to leave town for family emergency, he must delegate the surgeon role to another doctor. He cannot delegate the surgeon role to his two trusted colleagues, Miller and Nelson, because they will be on vacation. The hospital policy requires that a person assigned to a doctor role or senior can be delegated the role of surgeon. This rules out Davis. Thus, he computes trust values for the only two viable candidates, Bell and Cox.

Quantifying Properties: To perform the CAD type A surgery, the hospital requires the following attributes from the candidates. First, the candidate should be a doctor ($a_1 = doctor$) and he should be able to perform a CAD type A surgery ($a_2 = Surgery_A$). So, $TA = \{doctor, Surgery_A\}$. The hospital policy ranks the ability to perform a CAD type A surgery higher than the doctor position, so the policy administrator assigned $w_{Surgery_A} = 0.7$ and $w_{doctor} = 0.3$. The hospital administrator assigned the value of closeness equal to 0.6 between roles *Senior Doctor* and *Junior Doctor* ($dist$(*Senior Doctor, Junior Doctor*)=0.6), and that between roles *Junior Doctor* and *Cardiologist* equals 0.3 ($dist$(*Junior Doctor, Cardiologist*)=0.3). Hence, by using the computation method explained in Section 5, we get the value of closeness between role *Senior Doctor* and *Cardiologist* equals to $0.6 * 0.3 = 0.18$ ($dist$(*Senior Doctor, Cardiologist*)=0.18). The hospital policy ranks the importance of necessary attributes and role attributes equally, hence $w_a = w_r = 0.5$.
Now, we quantify the properties of both candidates. Bell is a doctor who can perform the CAD type A surgery ($UA_{Bell} = \{doctor, Surgery_A\}$), and Cox is a cardiologist who can perform a bypass surgery ($UA_{Cox} = \{cardiologist, Surgery_B\}$). So, $\mathcal{A}_{Bell} = w_{Surgery_A} + w_{doctor} = 0.7 + 0.3 = 1$ and $\mathcal{A}_{Cox} = w_{doctor} = 0.3$. Since Bell is a junior doctor, $\mathcal{R}_{Bell} = dist$ (*Senior Doctor, Junior Doctor*)=0.6. Since Cox is a cardiologist, $\mathcal{R}_{Cox} = dist$ (*Senior Doctor, Cardiologist*)=0.18.
Using this information, we calculate the properties value of the candidates:
$\mathcal{P}_{Bell} = w_a * \mathcal{A}_{Bell} + w_r * \mathcal{R}_{Bell} = 0.5 * 1 + 0.5 * 0.6 = 0.8$, and
$\mathcal{P}_{Cox} = w_a * \mathcal{A}_{Cox} + w_r * \mathcal{R}_{Cox} = 0.5 * 0.3 + 0.5 * 0.18 = 0.24$.

Quantifying Experience: Here the experience is quantify based on the number of heart operations the candidates have done in the past five years and the unit of the slot of the time period is equal to one year. The weight for each time slot where $slot_1$ represents the time period closest to the present time is defined by policy as follow: $w_{slot_1} = 1, w_{slot_2} = 0.8, w_{slot_3} = 0.6, w_{slot_4} = 0.4$, and $w_{slot_5} = 0.2$. Bell has performed surgery once 300 days ago ($slot_1$) with performance 0.7 ($p_{Bell_{slot_1}} = 0.7$) and Cox has performed surgery once 700 days ago ($slot_2$) with performance 0.8 ($p_{Cox_{slot_2}} = 0.8$). Thus, the experience value of both candidates can be calculated as follow:
$\mathcal{E}_{Bell} = \sum_{i=1}^{5} w_{slot_i} * p_{Bell_{slot_i}} = 1 * 0.7 + 0 + 0 + 0 + 0 = 0.7$, and
$\mathcal{E}_{Cox} = \sum_{i=1}^{5} w_{slot_i} * p_{Cox_{slot_i}} = 0 + 0.8 * 0.8 + 0 + 0 + 0 = 0.64$.

Quantifying Recommendation: Here, we have two recommenders–Miller and Nelson. According to hospital policy, the recommendation coming from senior doctor is more trustworthy than the one coming from junior doctor. So, the administrator set the trust value that hospital has about Miller (t_{Miller}) to 0.8 and the trust value that hospital has about Nelson (t_{Nelson}) to 0.2. Miller recommendation for Bell ($r_{MillerBell}$) and Cox ($r_{MillerCox}$) are 0.4 and 0.6, respectively. Nelson recommendation for Bell ($r_{NelsonBell}$) and Cox ($r_{NelsonCox}$) are 0.9 and 0.2, respectively. The computation results yield the recommendation for Bell and Cox as follow:

$$\mathcal{R}_{Bell} = \frac{t_{Miller} * r_{MillerBell} + t_{Nelson} * r_{NelsonBell}}{t_{Miller} + t_{Nelson}} = \frac{0.8 * 0.4 + 0.2 * 0.9}{0.8 + 0.2} = 0.5,$$

and

$$\mathcal{R}_{Cox} = \frac{t_{Miller} * r_{MillerCox} + t_{Nelson} * r_{NelsonCox}}{t_{Miller} + t_{Nelson}} = \frac{0.8 * 0.6 + 0.2 * 0.2}{0.8 + 0.2} = 0.52.$$

Computing Trustworthiness: Allen prefers the delegatee with more experience. So, he set the weights for properties (w_p), experience (w_e), and recommendation (w_r) to 0.2, 0.6, and 0.2, respectively. The trustworthiness of Bell and Cox can be computed as follow:

$\mathcal{T}_{Bell} = w_p * \mathcal{P}_{Bell} + w_e * \mathcal{E}_{Bell} + w_r * \mathcal{R}_{Bell} = 0.2 * 0.8 + 0.6 * 0.7 + 0.2 * 0.5 = 0.68$

, and

$\mathcal{T}_{Cox} = w_p * \mathcal{P}_{Cox} + w_e * \mathcal{E}_{Cox} + w_r * \mathcal{R}_{Cox} = 0.2 * 0.24 + 0.6 * 0.64 + 0.2 * 0.52 = 0.54$

Bell is selected to be the delegatee after comparing the trustworthiness values between both candidates.

7 Model Analysis

Once we have determined the most trustworthy candidate, we need to formally ensure that the choice of this delegatee does not cause a security breach. We do the formal analysis using the Alloy Analyzer. An Alloy model consists of *signature* declarations, *fields*, *facts* and *predicates*. Each signature consists of a set of *atoms* which are the basic entities in Alloy. Atoms are *indivisible* (they cannot be divided into smaller parts), *immutable* (their properties do not change) and *uninterpreted* (they do not have any inherent properties). Each field belongs to a signature and represents a relation between two or more signatures. A relation denotes a set of tuples of atoms. Facts are statements that define constraints on the elements of the model. Predicates are parameterized constraints that can be invoked from within facts or other predicates.

The basic types in the access control model, such as, *User*, and *Role* are represented as signatures. For instance, the declarations shown below define a set named *User*, and a set named *Role* that represents the set of all users, and roles in the system. Note that we use the *abstract* signature to represent these sets, and the different of users, and roles are modeled as the subsignatures of each signature. The analyzer will then recognize that users, and roles consist of only these different types, and nothing else.

```
abstract sig User{}
```

```
one sig Allen, Bell, Cox, Davis, Evans,
            Miller, Nelson extends User{}

abstract sig Role{}
one sig SeniorDoctor, JuniorDoctor, Assistant,
            Cardiologist, Surgeon, Patient extends Role{}
```

The different relationships between the RBAC components are also expressed as signatures. Signature *UserRoleAssign* which represents the roles assigned to user has a field called *URAsmember* that maps to a cartesian product of *User* an d *Role*. Signature *UserRoleAcquire* which represents the roles user can acquire through the assignment and role hierarchy has a field called *URAcqmember* that maps to a cartesian product of *User* and *Role*. We use the signature *RoleHierarchy* to represent role hierarchy relationship.

```
one sig UserRoleAssign{URAsmember: User -> Role}
one sig UserRoleAcquire{URAcqmember: User -> Role}
one sig RoleHierarchy{RHmember : Role -> Role}
```

The various invariants in the RBAC model are represented as facts in Alloy. For instance, the fact *URAcq* states that user can acquire all roles assign ed to him together with all of his junior roles. This is specified in Alloy as shown below. Other invariants are modeled in a similar manner.

```
fact URAcq{
UserRoleAcquire.URAcqmember =
UserRoleAssign.URAsmember +
(UserRoleAssign.URAsmember).^(RoleHierarchy.RHmember)}
```

The policy constraints are modeled as predicates. First, consider the cardinality constraint. The following constraint says that role r can be assigned to only one user.

```
pred Cardinality(r: Role, uracq: User->Role){
         (#((uracq).r) >= 1) &&
         (#((uracq).r) <= 1)}
```

Next, consider the prerequisite constraint that says that if a user u can acquire role $r1$, then he can also acquire role $r2$. The other forms are modeled in a separate manner.

```
pred Prerequisite(u:User, r1, r2: Role,
uracq: User->Role){
    (u->r2 in uracq) => (u->r1 in uracq)}
```

The separation of duty constraint says that if a user u can acquire role $r1$, then he cannot acquire the conflicting role $r2$.

```
pred SoD(u:User, r1, r2: Role, uracq: User->Role){
    (u->r1 in uracq) => not (u->r2 in uracq)}
```

The different types of delegation are also modeled as predicates. The grant and transfer operation can be modeled as follows:

```
pred Grant[u: User, r: Role,
   uracq, uracq': User->Role]{
     uracq' = uracq + (u->r)}

pred Transfer[u1, u2: User, r: Role,
   uracq, uracq': User->Role]{
     uracq' = uracq + (u2->r) - (u1->r)}
```

Finally, we need to verify whether the selected delegatee could cause any security policy violation. We create an *assertion* that specifies the properties we want to check. After we create the assertion, we will let ALLOY analyzer validate the assertion by using *check* command. If our assertion is wrong in the specified scope, ALLOY analyzer will show the counterexample. For example, suppose we want to check whether separation of duty constraint is violated when Allen delegates his role to Bell. The assertion below will check whether the separation of duty constraint is violated after the transfer operation. The separation of duty constraint says that user cannot be assigned both *Assistant* and *Surgeon* roles. The counterexample illustrates that even though user *Bell* is not assigned to *Assistant* role, he can still acquire it from the effect of role hierarchy.

```
assert TestConflict3{
     all u1, u2: User, r: Role, uracq, uracq': User->Role|
          ((u1 = Allen) && (u2 = Bell) && (r=Surgeon) &&
          (uracq = UserRoleAcquire.URAcqmember) &&
          (u1->r in UserRoleAcquire.URAcqmember) &&
          (u2->Assistant not in UserRoleAssign.URAsmember) &&
          Transfer[u1, u2, r, uracq, uracq']) =>
             SoD[u2, r, Assistant, uracq']}
check TestConflict3
```

The result shown that, although Bell is the most trustworthy candidate, we cannot choose him as Allen's delegatee. Next, we verify the situation where Cox, another candidate with the lower trustworthiness, is chosen as the delegatee. The assertion below will check whether the separation of duty constraint is violated after the transfer operation.

```
assert TestConflict4{
     all u1, u2: User, r: Role, uracq, uracq': User->Role|
          ((u1 = Allen) && (u2 = Cox) && (r=Surgeon) &&
          (uracq = UserRoleAcquire.URAcqmember) &&
          (u1->r in UserRoleAcquire.URAcqmember) &&
          (u2->Assistant not in UserRoleAssign.URAsmember) &&
          Transfer[u1, u2, r, uracq, uracq']) =>
             SoD[u2, r, Assistant, uracq']}
check TestConflict4
```

Here, the analyzer cannot find the counterexample, which means the separation of duty constraint defined in the model is not violated. This indicates that Cox is a more suitable delegatee for Allen.

8 Conclusion and Future Work

Delegation gives temporary privilege to one or more users, that allows critical tasks to be completed. We provide a formal approach for choosing delegatees. The approach evaluates the trustworthiness of candidates, and then ensures that the chosen candidate does not cause a security breach. We also illustrate how trustworthiness can be used to decide on the length of the delegation chain. A lot of work remains to be done. The first work is with regards to implementing the model such that trust computation can be done in an efficient manner. The second is with respect to validating the model in the context of real-world applications. The results of this validation can be further used to refine the model.

References

1. I. Agudo, M. C. F. Gago, and J. Lopez. A model for trust metrics analysis. In *Proceedings of the 5th International Conference on Trust, Privacy and Security in Digital Business*, pages 28–37, 2008.
2. I. Agudo, J. Lopez, and J. A. Montenegro. Enabling Attribute Delegation in Ubiquitous Environments. *Mobile Networks and Applications*, 13(3-4):398–410, 2008.
3. E. Barka and R. S. Sandhu. A Role-Based Delegation Model and Some Extensions. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.
4. S. Chakraborty and I. Ray. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 49–58, 2006.
5. J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 31–40, 2008.
6. I. F. Cruz, R. Gjomemo, B. Lin, and M. Orsini. A location aware role and attribute based access control system. In *Proceedings of the 16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, page 84, 2008.
7. E. Damiani, S. D. C. di Vimercati, and P. Samarati. New paradigms for access control in open environments. In *Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology*, pages 540–545, 2005.
8. G. Georg, J. Bieman, and R. B. France. Using Alloy and UML/OCL to Specify Run-Time Configurati on Management: A Case Study. In *Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists.*, volume P-7 of *LNI*, pages 128–141. German Informatics Society, 2001.
9. D. Jackson. *Alloy 3.0 reference manual.* At http://alloy.mit.edu/reference-manual.pdf, 2004.

10. A. Jøsang. Artificial reasoning with subjective logic. In *Proceedings of the 2nd Australian Workshop on Commonsense Reasoning*, 1997.
11. A. Jøsang. An algebra for assessing trust in certification chains. In *Proceedings of the Network and Distributed Systems Security Symposium*, 1999.
12. A. Jøsang and T. Bhuiyan. Optimal trust network analysis with subjective logic. In *Proceedings of the Second International Conference on Emerging Security Information, Systems and Technologies*, pages 179–184, 2008.
13. A. Jøsang, E. Gray, and M. Kinateder. Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems*, 4(2):139–161, 2006.
14. J. Joshi and E. Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 81–90, 2006.
15. J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.
16. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.
17. T. Priebe, W. Dobmeier, and N. Kamprath. Supporting attribute-based access control with ontologies. In *Proceedings of the 1st International Conference on Availability, Reliability and Security*, pages 465–472, 2006.
18. I. Ray, I. Ray, and S. Chakraborty. An interoperable context sensitive model of trust. *Journal of Intelligent Information Systems*, 32(1):75–104, 2009.
19. A. Schaad and J. D. Moffett. A Lightweight Approach to Specification and Analysis of Role-Based Access Control Extensions. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 13–22, 2002.
20. M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Formal Techniques for Networked and Distributed Systems*, pages 240–256, 2003.
21. L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pages 45–55, 2004.
22. Q. Wang and N. Li. Satisfiability and resiliency in workflow systems. In *Proceedings of the 12th European Symposium on Research in Computer Security*, pages 90–105, 2007.
23. Q. Wang, N. Li, and H. Chen. On the security of delegation in access control systems. In *Proceedings of the 13th European Symposium on Research in Computer Security*, pages 317–332, 2008.
24. J. Zao, H. Wee, J. Chu, and D. Jackson. *RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis.* At http://alloy.mit.edu/publications.php, 2002.
25. L. Zhang, G.-J. Ahn, and B. Chu. A rule-based framework for role based delegation. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pages 153–162, 2001.
26. X. Zhang, S. Oh, and R. S. Sandhu. PBDM: A Flexible Delegation Model in RBAC. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pages 149–157, 2003.