TBA: A Hybrid of Logic and Extensional Access Control Systems

Timothy L. Hinrichs[1], William C. Garrison III[2], Adam J. Lee[2], Skip Saunders[3], and John C. Mitchell[4]

[1] University of Chicago
[2] University of Pittsburgh
[3] MITRE Corporation
[4] Stanford University

Abstract. Logical policy-based access control models are greatly expressive and thus provide the flexibility for administrators to represent a wide variety of authorization policies. Extensional access control models, on the other hand, utilize simple data structures to better enable a less trained and non-administrative workforce to participate in the day-to-day operations of the system. In this paper, we formally study a hybrid approach, *tag-based authorization* (TBA), which combines the ease of use of extensional systems while still maintaining a meaningful degree of the expressiveness of logical systems. TBA employs an extensional data structure to represent metadata tags associated with subjects and objects, as well as a logical language for defining the access control policy in terms of those tags. We formally define TBA and introduce variants that include tag ontologies and delegation. We evaluate the resulting system by comparing to well-known extensional and logical access control models.

1 Introduction

Logical access control systems, in which users write formal logic to express access control policies, are expressive and supremely flexible but are hard to use because they require fluency in formal logic. Extensional access control systems (*e.g.*, the access matrix, role-based access control, Bell-La Padula), in which users enter atomic values (*e.g.*, roles, rights, classifications) into simple data structures (*e.g.*, a matrix or a pair of binary relations), are in contrast easy to use but are far less flexible. Judging from the prominence of extensional approaches in real-world organizations, ease-of-use is more important than flexibility; nevertheless, the problems with extensional systems are well known and can be addressed to a large extent with the flexibility of logical access control systems. Thus, a hybrid approach to access control that achieves the flexibility of logic and the usability of extensional systems would serve the community well.

As a case in point, MITRE recently published a report outlining the problems the U.S. military has had with their extensional access control system in the context of dynamic coalitions [18]. The main problem is the frequency with which partner countries enter and leave coalitions, causing the U.S. to make massive, frequent changes to its authorization policy. Logical access control systems are better suited to making large, frequent changes than extensional systems and so are an attractive alternative to the current system; however, it seems clear that the entire military cannot be trained to write formal logic in the near future. The right solution seems to be a combination of extensional and logical systems that allows relatively untrained personnel to create and contribute data while trained security experts write formal logic to express the desired access control policy.

Building a hybrid access control system that combines logic and extensionality is hard because the simplicity—and therefore usability—of extensional systems appears fundamentally at odds with logic's flexiblity. Extensionality's simplicity comes from its rigid commitment to a single representation of an access control policy, *e.g.*, RBAC grants subject s access to permission p when $\exists r. UR(s, r) \wedge PA(r, p)$. Logic's flexibility comes from its ability to represent a single policy in a myriad of ways—allowing security experts to choose the form best suited for supporting new and unforeseen demands. A hybrid system must therefore concede some of its flexibility by committing to a single representation for some component of the access control policy and must also concede some of its simplicity by allowing multiple representations of the policy.

In this paper, we formally study tag-based authorization (TBA), a hybrid access control system that combines the flexibility of logical access control systems and the usability of extensional systems. Relatively untrained people choose descriptive tags for the system's subjects and objects (similar to the tagging employed by many popular and successful web applications, such as Flickr and YouTube), and trained security experts write logical policies that define access permissions using combinations of subject and object tags (Section 2). One step we take to make TBA flexible yet easy to use is including delegation but separating delegation decisions from access control decisions. We replace delegation primitives inside the policy language with a scheme for combining policies outside the language (Section 3). We evaluate TBA by demonstrating its ability to express a number of well-known access control paradigms (Section 4). Finally we discuss related work (Section 5) and conclude (Section 6).

2 Tag-based Authorization

Tag-based authorization combines the strengths of logical access control systems and extensional access control systems. Just as with logical access control, formal logic is used to describe the authorization policy. Just as with extensional access control, subjects and objects are ascribed a small set of simple properties when they are added to the system (*e.g.*, roles in Role-based Access Control or classifications and clearances in Bell-La Padula). The properties ascribed to subjects and objects are tags that capture all of the security-relevant properties of that subject or object. The authorization policy is defined in terms of tags: it dictates which subject tags are sufficient for which rights to which object tags. Because of the simplicity of tagging, relatively untrained users can tag subjects and objects, while a relatively small number of administrators write the logical authorization policy.

Formally, we use S to denote the set of subjects, O to denote the set of objects, and R to denote the set of rights. T denotes the set of possible tags, and tag denotes the function that maps subjects and objects to tag sets: $tag : S \cup O \rightarrow 2^T$. Tag denotes the set of all possible tag functions.

An authorization policy is written in some logical access control language $\langle \mathcal{P}, \mathcal{L}, \models \rangle$. $\mathcal{P}$ is the set of all authorization policies; $\mathcal{L}$ is the set of queries, which we assume always includes $allow(s, o, r)$ for all subjects s, objects o, and rights r; $\models$ dictates which queries are true given an authorization policy and a tag function.

Definition 1 (Tag-based authorization (TBA)). *For a logical language $\langle \mathcal{P}, \mathcal{L}, \models \rangle$, a policy $\Delta \in \mathcal{P}$, and a tag function tag where*

- *$\mathcal{P}$: the set of all authorization policies*
- *$\mathcal{L}$: the set of queries including $allow(s, o, r)$ for all subjects s, objects o, rights r*
- *$\models$: a subset of $\mathcal{P} \times Tag \times \mathcal{L}$*

$$auth_{TBA}(s, o, r) \text{ if and only if } \Delta, tag \models allow(s, o, r)$$

The following example illustrates TBA using DATALOG as the policy language.

Example 1 (Basic Tag-Based Authorization). Consider two subjects—s_1 and s_2—and two objects—o_1 and o_2—that are tagged as follows:

- $tag(s_1) = \{US, Army, enduring\_freedom, signals\}$
- $tag(s_2) = \{France, Navy\}$
- $tag(o_1) = \{submarine, radar\}$
- $tag(o_2) = \{Kandahar, sat\_732, high\_res\}$

Further, consider the following policy.

$$allow(S, O, read) :- US \in tag(S), Navy \in tag(S), submarine \in tag(O)$$
$$allow(S, O, read) :- France \in tag(S), Navy \in tag(S), submarine \in tag(O)$$
$$allow(S, O, read) :- signals \in tag(S), submarine \in tag(O)$$
$$allow(S, O, read) :- US \in tag(S), enduring\_freedom \in tag(S),$$
$$high\_res \in tag(O), sat\_732 \in tag(O)$$

This policy allows U.S. and French naval officers to access documents about submarines (via rules 1 and 2), all signals officers to access documents about radar systems (rule 3), and all members of the U.S. military serving on Operation Enduring Freedom to access high resolution satellite photographs taken by sat_732 (rule 4). As a result, subject s_1 can access objects o_1 and o_2, while subject s_2 can only access object o_1. ■

Tag-based authorization differs from standard logical access control systems in that tag has a fixed semantics and is defined outside of the policy. The fixed semantics of tag forces policy-writers to define an authorization policy at a higher level of abstraction than the usual $S \times O \times R$. Policies in TBA are really concerned with access control decisions over the space of tags where subjects and objects are replaced by tag sets: $2^T \times 2^T \times R$. This abstraction results in a less flexible system since tag-space may not be the right one for a particular situation; however, the loss of flexibility is the price of a more understandable system for the majority of users. Relatively untrained users can contribute to the system by changing tag, yet trained administrators can utilize the flexibility of logic for expressing an access control policy. Thus, TBA enables a more thorough utilization of the spectrum of skills present in a typical workforce.

2.1 Tag Ontologies

One of TBA's limitations is that the number of relevant tags for a given subject or object can be large and must be managed properly to ensure that (i) everyone uses the same tags to mean the same thing and (ii) people are not routinely forced to tag subjects/objects with hundreds or thousands of tags, *e.g.*, the tag $boat$ may imply the tag $aquatic$, $aquatic$ might imply $vehicle$, and so on.

Both to help people reach consensus on tag meanings and to reduce the burden of document tagging, we propose employing an ontology to encode the relationships among tags. An ontology is helpful in the context of TBA in three ways. First, an ontology states which tags imply other tags, thereby reducing the number of tags that must be explicitly assigned to a subject or object; all tags implied are implicitly included, *e.g.*, tagging an object with $boat$ implicitly includes the tags $aquatic$ and $vehicle$. Second, an ontology simplifies policy-writing because it states that some tag combinations are illegal, *e.g.*, $short$ and $tall$, and the policy need not cover illegal tag combinations. Third, an ontology helps people communicate the meanings of tags because it explicitly states the relationships to other tags, *e.g.*, if bat implies $animal$, it is clear that bat refers to an animal instead of sports equipment.

Formally, a tag ontology Γ is a set of statements in propositional logic where the propositions are tags. A set of tags G is a legal combination whenever $G \cup \Gamma$ is logically consistent. The set of tags implied by some tag set G is the set of all t such that $G \cup \Gamma$ entails t, denoted $Cn^{\Gamma}(G)$. We use $Cn^{\Gamma}(tag)$ to denote the application of Cn to all tag sets in the tag function tag.

Employing ontologies leads to a new version of tag-based authorization.

Definition 2 (Ontology-aided TBA). *Suppose Δ is a TBA premise set, tag is a tag function, and Γ is an ontology. For every $x \in S \cup O$, $tag(x) \cup \Gamma$ must be consistent.*

$$auth_{\mathsf{TBA}}(s, o, r) \text{ iff } \Delta, Cn^{\Gamma}(tag) \models auth(s, o, r)$$

Example 2 (Ontology-Aided TBA). Consider a system containing some subject s and some object o that can be described as follows:

- $tag(s) = \{France, Navy\}$
- $tag(o) = \{submarine, radar\}$

Further, assume that the $policy$ is the following DATALOG.

$$auth(S, O, read) :- France \in tag(S), Navy \in tag(S), watercraft \in tag(O)$$

Intuitively, this policy allows French naval officers access to documents about watercrafts. In the basic tag-based authorization model, subject s would be denied access to object o because o is not explicitly tagged as a document about watercrafts. However, given a tag ontology containing the assertion $submarine \Rightarrow watercraft$, subject s would be permitted access because $tag(o)$ would implicitly include $watercraft$. ■

Ontology-aided TBA further enables all classes of users to contribute to the running of the system. Untrained personnel, who contribute mainly through generating and tagging data, can do so with even less effort thanks to the ability to tag with a smaller number of more specific tags. Administrative personnel also benefit because they can ignore incompatible tag combinations, inevitably leading to shorter policies.

3 Delegation

TBA employs a single logical policy to represent all access control decisions, but often that single policy is derived from many conceptually separate policies written by different security experts. The standard approach to providing the illusion of a single policy from multiple disparate policies is to include delegation primitives in the logical policy language that dictate how the disparate policies are to be combined, *e.g.*, [1, 14]. This approach is supremely flexible. For example, policy A might import policy B's decisions as long as policy C imports policy D's decisions on a specific topic. The downside to adding delegation to the language is that it can be difficult to understand how a given set of policies contribute to the overall policy—it might require reasoning about the logical consequences of all of the policies at once; moreover, small changes to any one policy may radically alter how the policies are pieced together.

Instead of adding delegation *inside* the logical language, TBA adds delegation *outside* of the logical language, thereby separating delegation decisions from access control decisions. In particular, we utilize constructs that arrange a set of policies into a partial order, where if $A \prec B$ then B delegates to A. Not only is this form of delegation especially simple to understand, it allows different security experts to choose different logical languages for writing their policies. The only restriction is that all of the logical languages used in the partial order must make access control decisions that are axiomatizable in a common logical language; otherwise, there would be no way to combine the access control decisions made by distinct policies. We call one of these partial orders of policies a *structured policy*.

More precisely, a structured policy is comprised of (i) a set of basic policies, (ii) a partial order of those policies, (iii) a set of guards on the partial order, (iv) a meta-language in which access control decisions are axiomatizable, and (v) a conflict resolution operator. A partial order over the policies enables delegation and *implicitly* imposes limits on the decisions delegated; the guards on the partial order *explicitly* limit the decisions that are delegated. If policy A is greater in the partial order than B then A delegates to B the decisions A does not make, and if that delegation is guarded by G then B's actual decisions are limited to those described by G. Because the ordering on policies is partial instead of total, some access control decisions are ambiguous, and the conflict resolution mechanism is used to disambiguate such decisions.

For example, in the U.S. military, basic policies might be written by the President, his chiefs of staff, and others. The partial order includes a single maximal policy: the President's. If the President allows or denies a request, the decision has been made; otherwise, the chiefs of staff have the opportunity to make a decision. The Army chief of staff is restricted from making decisions the Air Force chief of staff ought to make because of guards that restrict the Army to the Army-pertinent decisions and the Air Force to the Air Force-pertinent decisions. If the Army and Air Force make opposing decisions about a request that is pertinent to them both, the conflict resolution mechanism dictates whose decision will be enforced.

Definition 3 (Structured Policy). *A structured policy is a five-tuple* $\langle P, \prec, G, N, res \rangle$.

- P*: a finite set of basic policies. If policy* q *is written in logical language* $\langle \mathcal{P}_q, \mathcal{L}_q, \models_q \rangle$ *then both* $allow(s, o, r)$ *and* $deny(s, o, r)$ *belong to* $\mathcal{L}_q$ *for all* s, o, r.
- $\prec$*: a binary relation over* P *whose transitive closure is irreflexive (*i.e.*, no cycles)*
- G*: a set of functions* $guard_{B \prec C} : S \times O \times R \rightarrow \{true, false\}$ *for every* $B \prec C$
- N*: the meta language,* i.e.*, a logical language* $\langle \mathcal{P}^*, \mathcal{L}^*, \models^* \rangle$ *such that*
  - *all subsets of* $\bigcup_{q \in P} \mathcal{L}_q$ *are included in* $\mathcal{P}^*$

  – $\mathcal{L}^*$ *includes* $allow(s, o, r)$ *and* $deny(s, o, r)$ *for all s,o,r.*

- $res : 2^{\mathcal{L}^*} \times S \times O \times R \rightarrow \{allow, deny\}$ *is a conflict resolution operator: if* $allow(s, o, r)$ *is part of its input but* $deny(s, o, r)$ *is not then it returns allow, and vice versa.*

In this definition, the guard for an ordering $B \prec C$ is formalized as a function that dictates which subset of access control requests B is permitted to make. In practice that function is expressed in a logical policy language. For example, the guard might itself be a TBA (structured) policy, thereby deciding which requests are pertinent for B based on the tags for the subjects and objects.

Example 3 (Guards). Suppose the President wanted to scope the policy of his Army Chief of Staff so that it could only make authorization decisions about the objects the Army is primarily responsible for. If all such objects are tagged with $army$, the guard on the ordering $Army \prec Pres$ might be expressed as

$$allow(S, O, R) :- army \in tag(O). \quad \blacksquare$$

Another noteworthy part of our structured policy definition is the meta-language N. N represents a logical language in which the access control decisions of all the basic policies can be combined. Formally, the process of combining access control decisions is achieved with N's entailment relation: given the decisions made by (possibly) different policies, compute all the implications of those decisions. Technically, this requires the premise sets of N to include all possible combinations of access control decisions from the individual policy languages—a constraint included in the definition.

The final component of a structured policy that warrants discussion is the conflict resolution operator res. res is given the implications of all the appropriate policy decisions and must choose whether to allow or deny. For unambiguous cases (where either $allow$ or $deny$ is present but not both), its behavior is fixed, but for ambiguous cases where its input includes both allow and deny, it is free to make either decision. Because the language of access control decisions is unconstrained, those decisions can record a plethora of information important for conflict resolution, *e.g.*, the source of the decision or its proof. Thus, the conflict resolution operator may be given not only a series of $allow$ and $deny$ statements but also statements that justify each $allow$ and $deny$. For example, for conflict resolution that utilizes proofs, the individual policy decisions might always include a sentence of the form $explanation(allow/deny(s, o, r), proof)$. Thus, TBA makes no commitment to a particular conflict resolution operator or even the information upon which conflicts are resolved, as these issues have been studied heavily in the literature, *e.g.*, [2,3,9,11,16,20].

The formal semantics of a structured policy is defined in terms of the decision a given basic policy p makes about a given access control request $\langle s, o, r\rangle$. If p either allows or denies the request, p's decision stands; otherwise, p's decision is the combination of its partial decisions together with the union of the decisions made by the policies to which p delegated (*i.e.*, the policies immediately less than p in the policy ordering). A structured policy allows a request $\langle s, o, r\rangle$ if the conflict resolution operator when applied to the union of the decisions made by the maximal policies in the ordering returns $allow$; otherwise, the structured policy denies the request.

Furthermore, because the definition for a structured policy allows basic policies to be written in different logical languages (including *e.g.*, linear logic [7], first-order logic [12], and ASP [3]), the formal semantics correctly addresses heterogenous collections of basic policies, using $\models_p$ to denote the entailment relation for policy p.

Definition 4 (Structured Policy Semantics). *Consider a structured policy* $\langle P, \prec, G, \langle \mathcal{P}^*, \mathcal{L}^*, \models^* \rangle, res \rangle$*, tag function tag, ontology* Γ*, and an access control request* $\langle s, o, r \rangle$*. First, for all* $x \in S \cup O$*,* $tag(x) \cup \Gamma$ *is consistent. Second we define the point semantics of policy* $p \in P$ *on* $\langle s, o, r \rangle$*, written* $Point[p, s, o, r]$*, which is an element of* $\mathcal{P}^*$*. Let* $S = \{\phi \mid p, Cn^{\Gamma}(tag) \models_p \phi\}$*.*

1. *If* S *includes* $allow(s, o, r)$ *and/or* $deny(s, o, r)$ *then* $Point[p, s, o, r] = S$*.*
2. *Otherwise,* $Point[p, s, o, r] = S \cup \bigcup_{\substack{q \prec p \text{ and} \\ guard_{q \prec p}(s,o,r)}} Point[q, s, o, r]$*.*

Finally we define the structured policy semantics.

$$auth_{\mathsf{TBA}}(s, o, r) \text{ iff } res\left(Cn^*\left(\bigcup_{p \mid \nexists q. p \prec q} Point[p, s, o, r]\right)\right) = allow$$

Admittedly the formal definitions for a structured policy are not so simple; however, once the logical policy languages are chosen, explaining to policy writers how to use a structured policy is especially simple: write basic policies to make access control decisions and adjust the partial order and its guards to delegate those decisions.

Example 4 (Disjunctive decisions). Suppose an upper-level manager wants to ensure that every employee is either given access to object o_1 or object o_2 but not both. Moreover, she wants to delegate the choice to the low-level managers in the company. She can author a (first-order logic) policy, A, that says $\forall s.(allow(s, o_1, read) \vee allow(s, o_2, read))$ and $\forall s.(deny(s, o_1, read) \vee deny(s, o_2, read))$. Then if the low-level manager policies are $B_1,\ldots,B_n$, the upper-level manager ensures that $B_i \prec A$ with appropriate guards for $i \in \{1, \ldots, n\}$. Each policy B_i can then choose which of the objects to grant for each employee. Furthermore, if one of the low-level managers writes a policy that grants access to both objects or to neither, there will be a conflict, and the conflict resolution operator can choose to enforce A's policy by arbitrarily choosing between o_1 and o_2. ■

In the technical report version of this paper [13], we show the algorithms used to evaluate an ontology-aided, structured TBA policy to either allow or deny a given request; the algorithms are omitted here for brevity.

4 Evaluation

In this section, we evaluate the utility of TBA by exploring its expressive power. We first demonstrate that various incarnations of TBA can be used to express a range of common policy idioms. We then use the formal reduction framework developed by Tripunitara and Li [17] to demonstrate that TBA is more expressive than several representative access control schemes from the literature.

4.1 Representing Common Policy Idioms

Below we enumerate a list of well-known authorization policy idioms and show that each can be represented using some tag-based authorization system. When using tag-based authorization to represent each of these idioms, we use DATALOG as the underlying policy language. In doing so, we assume that every request not explicitly allowed is denied.

Access matrix. The access matrix uses a function $matrix : S \times O \to 2^R$ to store the rights that each subject has over every object. An access is permitted under the following condition: $auth_{mat}(s, o, r)$ iff $r \in matrix(s, o)$. To implement this scheme using tag-based authorization, there must be a unique tag for each document (*e.g.*, its inode number) and each user (*e.g.*, her uid). The policy consists of a series of simple statements such as the one below.

$$allow(S, O, read) :- user123 \in tag(S), doc789 \in tag(O)$$

Attribute-based access control. In attribute-based authorization systems, access decisions are made based on the attributes ascribed to a user by their organization. Basically, ABAC is TBA without object tags (or, more formally, where every object is tagged with the empty set). The following example allows any user to read $doc789$, provided that she is a member of the security group and has not been blacklisted.

$$allow(S, doc789, read) :- security \in tag(S), blacklist \notin tag(S)$$

Role-based access control. In RBAC systems, users are assigned to roles representing their job functions, and permissions are given to roles. Here we focus on RBAC1 as defined in [10], where the roles are arranged in a hierarchy, and a user is granted access when one of her roles is higher in the hierarchy than some role that is permitted access.

$$auth_{RBAC1}(s, o, r) \text{ iff } \exists g, g'.UR(s, g) \wedge g \geq g' \wedge PA(g', o, r))$$

To implement RBAC1 with tag-based authorization, the tag set is defined as $T = G \cup G \times R$, *i.e.*, the set of roles and the set of (role, right) tuples. Users are tagged with their roles, and documents are tagged with (role, right) tuples. The role hierarchy is axiomatized as an ontology Γ so that for every pair of roles such that $g \geq g'$, we have $\Gamma \models g \Rightarrow g'$. The following DATALOG policy implements RBAC1.

$$allow(S, O, R) :- G \in tag(S), \langle G, R \rangle \in tag(O)$$

Discretionary access control (Linux). In the Linux authorization model, each object has different rights for its owner, group, and the world. The Linux authorization policy gives a user access if (i) the user owns the document and the owner has access, (ii) the user belongs to the group that owns the document and the group has access, or (iii) the world has access. To implement the Linux scheme with TBA, the document tags T consist of the subjects S, groups G, and the rights tags $\{userread, groupread, worldread\}$. Permissions other than *read* can be handled in a similar manner. Each document is tagged with the user owner, the group owner, and a subset of the rights tags, and each user is tagged with the groups she belongs to. The Linux read policy is then given by the DATALOG fragment below.

$$\begin{array}{l} allow(U, D, read) :- U \in tag(D), userread \in tag(D) \\ allow(U, D, read) :- G \in tag(D), groupread \in tag(D), G \in tag(U) \\ allow(U, D, read) :- worldread \in tag(D) \end{array}$$

Mandatory (Lattice-Based) Access Control An LBAC system utilizes a set of classification/clearance levels (*e.g.*, Secret, TopSecret) and a set of compartments (*e.g.*, Nuclear, Submarine). Each subject and object is assigned a security classification: a level and a set of compartments. There is a total ordering $\leq$ on levels, which induces a partial ordering on level/compartment-set pairs. $(l_1, c_1) \sqsubseteq (l_2, c_2)$ if and only if $l_1 \leq l_2$ and $c_1 \subseteq c_2$. Subjects can read objects whose security classifications are dominated by their classification (no read up) and write documents whose classifications dominate their classification (no write down).

To implement this policy idiom with tag-based authorization, the set of tags is the set of all compartments and security levels. Each subject and object is tagged with its level and all its compartments. Then the DATALOG (with negation) policy for the LBAC no-read-up idiom is given below, where compartment tags are identified by $comp$, level tags by $level$, and the total ordering on levels is represented by leq.

$$\begin{array}{l} allow(S, O, read) :- allowlevel(S, O), allowcomp(S, O) \\ allowlevel(S, O) :- C \in tag(S), level(C), \\ \qquad\qquad\qquad\quad E \in tag(O), level(E), leq(E, C) \\ allowcomp(S, O) :- \neg somecompmissing(S, O) \\ somecompmissing(S, O) :- C \in tag(O), comp(C), C \notin tag(S) \end{array}$$

In the first rule, the first condition, $allowlevel$, ensures that the object's security level is less than the subject's level. The second condition, $allowcomp$, ensures that the object's compartments are a subset of the subject's compartments, which is implemented by ensuring it is not the case that one of the object's compartments fails to be one of the subject's compartments.

The RT Trust Management Language. RT [14] employs a form of role-based delegation that consists of the four types of rules shown in Table 1. Structured policies of TBA can express a certain fragment of RT-style delegation. Given a set of rules of types 1–3, where the delegation graph of those rules is acyclic, we can emulate those rules by constructing a structured TBA policy. (The delegation graph consists of one node per principal and an edge from A to B if A delegates to B.) The partial order $\prec$ includes $B \prec A$ if A delegates to B. Then by using a new right $activate$, we proceed as follows for each of the rule types.

1. • Add $allow(userB, roleA.R, activate)$ to policy A
2. • Add $allow(S, roleA.R, activate) :- allow(S, roleB.R_1, activate)$ to A
   • Add $\langle S, roleB.R_1, activate\rangle$ to $guard_{B \prec A}$ for all S
3. • Add $allow(S, roleA.R, activate) :- allow(S, roleB.R_1, activate)$, $allow(S, roleC.R_2, activate)$ to policy A

Type	Rule	Description
1	$A.R \leftarrow B$	User B is a member of the role R defined by user A
2	$A.R \leftarrow B.R_1$	A's role R contains all members of B's role R_1
3	$A.R \leftarrow B.R_1 \cap B.R_2$	A's role R contains all users who are members of both B's role R_1 and C's role R_2
4	$A.R \leftarrow A.R_1.R_2$	A's role R contains all users who are members of X's role R_2 for some X in A's role R_1

Table 1. The four types of RT rules.

- Add $\langle S, roleB.R_1, activate\rangle$ to $guard_{B \prec A}$ for all S
- Add $\langle S, roleC.R_2, activate\rangle$ to $guard_{C \prec A}$ for all S

Rules of type (4) are not expressible since they cause the delegation graph to be dependent on the contents of basic policies. Even if it were reasonable to require that dynamic delegation graph to be acyclic, emulating type (4) rules would require changing $\prec$ each time a basic policy changed.

4.2 Formal Expressive Power Analysis

The preceding section demonstrates that TBA is capable of encoding many common policy idioms, but says nothing about whether these encodings have the same safety analysis properties of common implementations of these idioms. In [17], Tripunitara and Li introduce a framework for comparing the expressiveness of access control systems that views an access control system as a state transition system and performs comparisons using a type of bisimulation and a generalized definition of safety. The crux of their framework relies on demonstrating the existence or non-existence of a *state-matching reduction* between two systems. Intuitively, a state-matching reduction from A to B is a mapping from the states of A to the states of B so that an external observer affecting access control changes and making queries can not distinguish whether she is using A or B, and further implies that B maintains all safety analysis properties of A. As a result, state-matching reductions are a way of analyzing the relative expressive power of two systems.

First, we formally represent TBA within the definition of access control scheme proposed by Tripunitara and Li [17], the representation that allows us to construct state-matching reductions. Following this definition, an access control scheme is a state-transition system $\langle \Gamma, \Psi, Q, \vdash\rangle$, where:

- Γ is a set of states. Each state contains all the information needed to make an access control decision at any given moment.
- Ψ is a state-transition rule that describes how the system changes state.
- Q is a set of queries. Each query is answered by $true$ or $false$.
- $\vdash$ is the entailment relation that determines whether a given query is true or false in a given state.

In TBA, we assume the existence of a set T of possible tags, a set I of access rights, and a logical language L used to define the policy. These components are not defined as part of the system state, as they do not change. TBA is then defined as the state-transition system $\left\langle \Gamma^T, \Psi^T, Q^T, \vdash^T \right\rangle$. Each TBA state $\gamma^T \in \Gamma^T$ is defined by $\langle S, O, tag, P\rangle$, where S is the set of all subjects; O is the set of all objects; $tag \subseteq (S \cup O) \times T$ is

```
create_object($s$, $o$)
  $O = O \cup \{o\}$
  $tag = tag \cup \{\langle o, o.id \rangle\}$

destroy_object($s$, $o$)
  if($s$ has $delete$ for $o$)
    $O = O - \{o\}$
    $tag = tag - \{\langle o, o.id \rangle\}$

create_subject($s_1$, $s_2$)
  if($s_1$ can create subjects)
    $S = S \cup \{s_2\}$
    $tag = tag \cup \{\langle s_2, s_2.id \rangle\}$

destroy_subject($s_1$, $s_2$)
  if($s_1$ can delete subject $s_2$)
    $S = S - \{s_2\}$
    $tag = tag - \{\langle s_2, s_2.id \rangle\}$

assign_tags($s$, $x$, $T_1$)
  if($s$ can edit tags for $x$)
    for each $t \in T_1$
      $tag = tag \cup \{\langle x, t \rangle\}$

revoke_tags($s$, $x$, $T_1$)
  if ($u$ can edit tags for $x$)
    for each $t \in T_1$
      $tag = tag - \{\langle x, t \rangle\}$

transform_policy($s$, $P_+$, $P_-$)
  if($s$ can change $P$)
    $P = P - P_- \cup P_+$
```

Fig. 1. Command templates for TBA.

the tag relation, and contains the pair $\langle s, t \rangle$ for each subject s that has tag t and the pair $\langle o, t \rangle$ for each object o that has tag t; and P is a set of policy sentences, written in language L. Ψ^T is then defined using the commands in Figure 1. Q^T includes all queries of the following forms: (1) "Does subject s exist?", (2) "Does subject s have tag t?", (3) "Does object o have tag t?", (4) "Is policy sentence p in the policy?", and (5) "Does subject s have access i to object o?". $\vdash^T$ is defined as follows for queries of each of the forms above: (1) $true$ if and only if $s \in S$, (2) $true$ if and only if $\langle s, t \rangle \in tag$, (3) $true$ if and only if $\langle o, t \rangle \in tag$, (4) $true$ if and only if $p \in P$, and (5) $true$ if and only if $\exists T_1 \subseteq T, T_2 \subseteq T : \forall t_1 \in T_1, \langle o, t_1 \rangle \in tag \wedge \forall t_2 \in T_2, \langle s, t_2 \rangle \in tag \wedge \exists p \in P$ that grants subjects with tag set T_2 access i to objects with tag set T_1 under language L.

We now present theorems comparing TBA to a number of well-known systems in terms of state-matching reductions. We give the full proof for SDCO, while proofs for the other systems are available in a technical report version of this paper. [13] We first show that TBA is at least as expressive as a common discretionary access control scheme (SDCO), a common role-based access control scheme (ARBAC97), and a common mandatory access control scheme (the Bell-La Padula model).

```
create_object($s$, $o$)
  $O = O \cup \{o\}$
  $M[s, o] = own$

destroy_object($s$, $o$)
  if $own \in M[s, o]$
    $O = O - \{o\}$

grant_own($s$, $s'$, $o$)
  if $own \in M[s, o]$
    $M[s', o] = M[s', o] \cup \{own\}$
    $M[s, o] = M[s, o] - \{own\}$

grant_$i$($s$, $s'$, $o$)
  if $own \in M[s, o]$
    $M[s', o] = M[s', o] \cup \{i\}$

revoke_$i$($s$, $s'$, $o$)
  if $own \in M[s, o]$
    $M[s', o] = M[s', o] - \{i\}$
```

Fig. 2. Command templates for SDCO access control scheme. Commands with i in the name exist for each $i \in (I - own)$.

In SDCO, we assume the existence of I, the set of access rights, including own. SDCO is then defined as the state-transition system $\langle \Gamma^S, \Psi^S, Q^S, \vdash^S \rangle$. Each SDCO state $\gamma^S \in \Gamma^S$ is defined by $\langle S, O, M \rangle$, where S is the set of subjects, O is the set of objects, and $M : S \times O \to 2^I$ is the access matrix. Ψ^S is defined using the commands in Figure 2. Q^S includes all queries of the form "Does subject s have access i to object o?". $\vdash^S$ is defined as $true$ if and only if $i \in M[s, o]$.

Theorem 1. *There exists a state-matching reduction from SDCO, BLP, ARBAC97 to* TBA.

Proof. (SDCO only) By construction. Presented is a mapping, and proof that the mapping satisfies the two properties for it to be a state-matching reduction by Tripunitara and Li's definition 7. The mapping, σ, needs to be able to map every $\langle \gamma, \psi \rangle$ in SDCO to $\sigma(\langle \gamma, \psi \rangle) = \langle \gamma^T, \psi^T \rangle$ in TBA, as well as every q in SDCO to $\sigma(q) = q^T$ in TBA.

Let $\sigma(\gamma) = \gamma^T = \langle S_\gamma, O_\gamma, tag_\gamma, P_\gamma \rangle$ where $S_\gamma = S \cup \{sim\_admin\}$, $O_\gamma = O$, $tag_\gamma = \{\forall s \in S : \langle s, s.id \rangle\} \cup \{\forall o \in O : \langle o, o.id \rangle\}$, $P_\gamma = \{\forall s \in S, o \in O, \forall i \in M[s, o] : \text{"}s.id : i : o.id\text{"}\}$.

Here, L_γ is the set of sentences of the form "$t_1 : i : t_2$" where $t_1, t_2 \in T$ and $i \in I$. A policy P is consistent only if $\forall o \in O \exists s \in S$: "$s.id : own : o.id$" $\in P \wedge t \in T \neq s.id \implies$ "$t : own : o.id$" $\notin P$. The inference procedure for L_γ is as follows. The sentence "$t_1 : i : t_2$" grants any subject with the tag t_1 the right i to objects with the tag t_2. Since queries in SDCO have the same form as form-(4) queries in TBA, let $\sigma(q) = q^T = q$.

Let γ_0 be a start state in SDCO. Produce γ_0^T in TBA using σ. Given γ_k such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma_k$, we show that there exists γ_k^T such that $\gamma_0^T \overset{*}{\mapsto}_{\psi^T} \gamma_k^T$ where, for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. Consider the case where $\gamma_k = \gamma_0$, then let $\gamma_k^T = \gamma_0^T$. In $\gamma_0^T = \sigma(\gamma_0)$, s will be given right i over o only using the following sentence in P: "$s.id : i : o.id$". Such a line will be entered if and only if $i \in M[s, o]$, so for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. Next, consider some arbitrary γ_k reachable from γ_0. We construct γ_k^T that is reachable from γ_0^T and that answers every q^T the same way γ_k answers q, as follows. Consider each state-transition in the sequence $\gamma_0 \mapsto_\psi \gamma_1 \mapsto_\psi \ldots \mapsto_\psi \gamma_k$ in the SDCO system. If the state-transition in SDCO is the execution of `create_object(`s`, `o`)`, we execute `transform_policy(`sim_admin`, `$\{\text{"}s.id : own : o.id\text{"}\}$`, {})` followed by `create_object(`$s$`, `$o$`)`. If the state-transition in SDCO is the execution of `destroy_object(`s`, `o`)`, we execute `destroy_object(`s`, `o`)`, followed by `transform_policy(`sim_admin`, {}, `$\{\forall t, i : \text{"}t : i : o.id\text{"}\}$`)`. If the state-transition in SDCO is the execution of `grant_own(`s`, `s'`, `o`)`, we execute `transform_policy(`sim_admin`, `$\{\text{"}s'.id : own : o.id\text{"}\}$`, `$\{\text{"}s.id : own : o.id\text{"}\}$`)`. If the state-transition in SDCO is the execution of `grant_`i`(`s`, `s'`, `o`)`, then we execute `transform_policy(`sim_admin`, `$\{\text{"}s'.id : i : o.id\text{"}\}$`, {})`. If the state-transition in SDCO is the execution of `revoke_`i`(`s`, `s'`, `o`)`, then we execute `transform_policy(`sim_admin`, {}, `$\{\text{"}s'.id : i : o.id\text{"}\}$`)`. Now, consider each possible query q. Since q is of the form "Does subject s have access i to object o?", q^T is also "Does subject s have access i to object o?". In this case, $\gamma_k \vdash q$ if and only if i has been granted to s by the owner of o. This is true if and only if we have added the policy sentence "$s.id : i : o.id$" to P. Thus, $\gamma_k \vdash q$ if and only if $\gamma_k^T \vdash q^T$. Therefore, we've proven property (1) for state-matching reductions.

We prove that property (2) for a state-matching reduction is satisfied by our mapping also by construction. Let γ_0^T be the start-state in TBA corresponding to γ_0, the start-state in SDCO. Then, if γ_k^T is a state reachable from γ_0^T and q^T is a query in TBA whose corresponding query in SDCO is q, we construct γ_k, a state in SDCO reachable from γ_0 as follows. For each sentence in P of the form "$s.id : own : o.id$", we execute `create_object(`s`, `o`)`. Then, for each sentence in P of the form "$s'.id : i : o.id$" where $i \neq own$, we execute `grant_`i`(`s`, `s'`, `o`)`, where s is the owner of o. Since q is of the form "Does subject s have access i to object o?", q^T is also "Does subject s have access i to object o?", which means that $\gamma_k^T \vdash q^T$ iff "$s.id : i : o.id$" $\in P$. The condition that q^T is true is the only one in which we would have added the right i to $M[s, o]$, and therefore $\gamma_k \vdash q$ iff $\gamma_k^T \vdash q^T$. Therefore, we've proven property (2) for state-matching reductions, and thus our mapping σ is a state-matching reduction. $\square$

Since a state-matching reduction from A to B proves that B is at least as expressive as A, the above results show that TBA is at least as expressive as SDCO, ARBAC97, and BLP. The next results ensure that none of these schemes are as expressive as TBA.

Theorem 2. *There exists no state-matching reduction from TBA to SDCO, BLP, or ABAC97.*

Proof. (SDCO only) By contradiction. Assume there is a state-matching reduction from TBA to SDCO. In TBA, adopt as γ a state where $s \in S$, $o \in O$, $P = \{\}$, and L is as described in Theorem 1 except the inference procedure is augmented as follows. The sentence "$t_1 : i^* : t_2$", in addition to granting subjects with tag t_1 the right i^* over objects with tag t_2, also grants such subjects the right i' over the same objects, regardless of whether P also contains the sentence "$t_1 : i' : t_2$". Languages like this can be used to express a heirarchy of rights, *e.g.*, the *execute* right carries with it automatic *read* right. Let $q_1 =$ "Does s have right i^* to o?" and let $q_2 =$ "Does s have right i' to o?".

Observe that $\gamma \vdash \neg q_1 \wedge \neg q_2$. Consider the state γ^S in SDCO that is equivalent to γ (if there does not exist one, the contradiction of the existence of a state-matching reduction is found). We know that $\gamma^S \vdash \neg q_1^S \wedge \neg q_2^S$. Observe that, given $i^* \neq own$, there exists $\tilde{\gamma}^S$ reachable from γ^S such that $\tilde{\gamma}^S \vdash q_1^S \wedge \neg q_2^S$ via the execution of `grant_`i^*. However, the only $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1$ is that in which $\exists t_1, t_2 \in T$: "$t_1 : i^* : t_2$" $\in P \wedge \langle s, t_1 \rangle, \langle o, t_2 \rangle \in tag$. Due to the inference procedure of L, such a $\tilde{\gamma} \vdash q_1 \wedge q_2$, leaving no such $\tilde{\gamma} \vdash q_1 \wedge \neg q_2$, meaning there is no $\tilde{\gamma}$ that is equivalent to $\tilde{\gamma}^S$. This contradicts property (2) for state-matching reductions, giving us the needed contradiction and proof of the non-existence of a state-matching reduction from TBA to SDCO. $\square$

When a state-matching reduction from A to B is accompanied by the nonexistence of a state-matching reduction in the reverse direction, it proves that B is *strictly* more expressive than A. As a result, we have shown that TBA is strictly more expressive than all of SDCO, ARBAC97, and BLP.

5 Related Work

TBA has been studied informally in [15, 19], though that work allows tags on subjects but not objects. Section 4 compares TBA to several well-known authorization paradigms. We do not survey related work on tag ontologies, which have been studied extensively by the Semantic Web community, but as evidence of viability simply point to two organizations employing ontologies to handle large terminologies: the U.S. National Cancer Institute (NCI Thesaurus `http://www.cancer.gov/cancertopics/terminologyresources`) and the U.S. National Library of Medicine (SNOMED-CT `http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html`). In this section, we discuss work related to structured policies, conflict resolution, and delegation.

Policy Structure. Structured policies can be seen as combining two operations on policies: the "override" operator in [6] (also called "exceptions" in [4]) and the "scoping" operator in [6, 16]. Our choice to employ these two policy combination operators instead of a richer framework [6, 8] was driven by our desire for a conceptually simple and therefore highly usable framework; these two operators seemed to be the minimal necessary to support delegation.

Conflict resolution. Conflict resolution is important in the context of structured policies, where conflicts must be resolved within basic policies as well as across policies. We make no commitment to a particular scheme but provide a framework for implementing other proposals in the literature. Our framework is based on the premise of a fixed global operator such as [3, 9, 16], though user-settable conflict resolution schemes for each policy such as in [2, 11, 20] can be achieved by building them into the entailment relations for the individual policies.

Trust Management and delegation. Trust management [5] is concerned with distributed authorization and therefore focuses extensively on delegation. TBA's delegation functionality was designed for simplicity, and as shown in Section 4 is less powerful than RT's delegation primitives. This decision was made to improve usability, with an acknowledged decrease of flexibility and expressiveness.

6 Conclusion

Logical access control systems are attractive for their power and flexibility, while extensional access control systems are known for their simplicity and the ease with which relatively untrained users can contribute. Tag-based authorization combines these qualities into a single system. Subjects and objects are assigned tags, and access is decided by a policy over those tags. TBA is powerful and flexible through its logical policy, and achieves nearly the expressiveness of existing logical access control systems. At the same time, it is simple to describe and allows relatively untrained personnel to assign tags to objects, more fully utilizing a diverse workforce. Tag ontologies can further simplify both object tagging and policy writing. In addition, our approach to delegation externalizes the mechanism through which policies are combined, enabling different sub-policies to be written in distinct languages.

To evaluate TBA, we explored its ability to express common access control policy idioms and its formal expressive power. We show via simple example instantiations that TBA is capable of expressing the access matrix, attribute-based, role-based, discretionary and mandatory access control paradigms. Then, by utilizing the reduction framework of Tripunitara and Li, we show that TBA is strictly more expressive than specific, common implementations of these paradigms, namely SDCO (a common access matrix system), ARBAC97 (a common role-based system), and BLP (the U.S. military's extended mandatory system). Thus, TBA is not only much more intuitive to describe and use than other current logical authorization systems, but also strictly more expressive than current extensional access control systems, making it a true hybrid of these two types of access control metaphors.

References

1. Moritz Y. Becker, Cedric Y. Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *JCS*, 2009.
2. Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM TISSEC*, 6(1):71–127, 2003.
3. Elisa Bertino, Elena Ferrari, Francesco Buccafurri, and Pasquale Rullo. A logical framework for reasoning on data access control policies. In *IEEE CSFW*, 1999.
4. Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM TISSEC*, 17(2):101–140, 1999.
5. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
6. Piero A. Bonatti, Sabrina D. di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *ACM CCS*, pages 164–173, 2000.
7. Kevin D. Bowers, Lujo Bauer, Deepak Garg, Frank Pfenning, and Michael K. Reiter. Consumable credentials in logic-based access-control systems. In *NDSS*, pages 143–157, 2007.
8. Glenn Bruns and Michael Huth. Access-control policies via belnap logic: Effective and efficient composition and analysis. In *IEEE CSF*, 2008.
9. Laurence Cholvy and Frederic Cuppens. Analyzing consistency of security policies. In *IEEE S&P*, 1997.
10. Jason Crampton. Understanding and developing role-based administrative models. In *ACM CCS*, pages 158–167, 2005.
11. Frederic Cuppens, Laurence Cholvy, Claire Saurel, and Jerome Carrere. Merging security policies: analysis of a practical example. In *IEEE CSFW*, 1998.
12. Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *IEEE CSFW*, 2003.
13. Timothy Hinrichs, William Garrison, Adam Lee, Skip Saunders, and John Mitchell. TBA: A hybrid of logic and extensional access control systems (Extended version). Technical Report TR-11-182, University of Pittsburgh, October 2011.
14. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *IEEE S&P*, 2002.
15. Maryam Najafian Razavi and Lee Iverson. Supporting selective information sharing with people-tagging. In *CHI Extended Abstracts*, pages 3423–3428, 2008.
16. Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. SPL: An access control language for security policies with complex constraints. In *NDSS*, 2001.
17. Mahesh V. Tripunitara and Ninghui Li. A theory for comparing the expressive power of access control models. *JCS*, 15(2):231–272, 2007.
18. U.S. Air Force Scientific Advisory Board. Networking to enable coalition operations. Technical report, MITRE Corporation, 2004.
19. Qihua Wang, Hongxia Jin, and Ninghui Li. Usable access control in collaborative environments: Authorization based on people-tagging. In *ESORICS*, pages 268–284, 2009.
20. Duminda Wijesekera and Sushil Jajodia. Policy algebras for access control - the predicate case. In *ACM CCS*, pages 171–180, 2001.