

Decidable Administrative Controls based on Security Properties

Jon A. Solworth and Robert H. Sloan
University of Illinois at Chicago

Abstract

A security property is a high-level statement about what may occur (is authorized) within a system. One of the oldest such security properties is information flow confidentiality.

Given a security property p , it is a desirable goal for an authorization model to be expressive for p (enabling p to be both enforced and violated in different parts of the system), robust (enabling the authorization state to change without invalidating p where it holds), and analyzable (so it can be understood where p holds). Of particular interest in analyzing an authorization model is the decidability of security properties. If the system is not analyzable, how does one know what protections are being provided?

Protections can be provided at two levels: the ordinary privileges and the ability to change the system via administrative controls. Administrative controls provide a graceful means to perform the inevitable modifications to the system, that is to provide robust authorization systems.

To date, existing authorization systems are known to achieve at most two of expressibility, robustness, and decidability with respect to a security property. This paper proves that a previously proposed authorization model with administrative controls is decidable with respect to information flow confidentiality, thus simultaneously achieving all three of these goals.

1 Introduction

Security properties are high-level abstract statements about the protection of the system. Examples of security properties are information flow confidentiality, information flow integrity, separation-of-duty, and executable constraints. Security properties are high-level because they can be understood without a technical background and in fact, these properties have their genesis prior to computer systems¹. If a system can be analyzed in terms of its security properties, then non-technical management can understand the security provided.

In this paper we focus on one of the oldest security properties, information flow confidentiality: Information labeled l_0 may not flow to an object labeled l_1 if at any time after that l_1 's readership is not contained within l_0 's readership. The previous statement could be written without referring to an "object" and using classification rather than "label": It is written in this form to better segue into the theory that follows². As an example of the usefulness of such a security property, one

¹It may seem that executable constraints are technical, but we argue that they are standard procedural constraints to ensure that, for example, accounts are only updated through correctness preserving transactions.

²Alternative information flow security property formulations are possible, but we believe this formulation captures reasonably well the intuition.

may want to ensure that one's credit card number is never sent anywhere except to an authorized vendor.

In general, the security properties are always safe but not always appropriate. For example, it is necessary to *violate* information flow confidentiality under at least two different conditions: The first is to *declassify* information, for example if the information became public through other means. The second is because information has been *sanitized*, perhaps by redacting names. Declassification and sanitization are often mandated: For example, a public corporation may be legally required to publish quarterly public financial reports from highly confidential accounting information.

We consider next the **authorization model** which, given an **authorization state** allows or denies a program's requested operations based, in part, on the user who requests them—in other words a type of Reference Monitor [And72] or Enforcement Mechanism [Sch00]. The purpose of the authorization model is to ensure safety properties of the underlying computer **system**. An example of an authorization model is ARBAC'97 [SBM99], and the authorization state a particular ARBAC'97 specification.

Let p be a security property of interest. An authorization model is **expressive for** p if there exists an authorization state in which p can be selectively enforced or violated in different parts of the same system. In expressive authorization models, security properties are not, in general, safety properties since they do not hold universally in the system.

An authorization model is **robust for** p if there exists authorization state s such that (1) in every successor authorization state p holds where it holds in s and (2) there exist successor authorization states in which p can be made to hold in an arbitrary number of additional parts of the system. An authorization model which is not robust is **fragile**. The purpose of (2) is to ensure that the authorization state is not static but can be grown to arbitrary size. Robustness means that ordinary permissions can be changed *and* that the degree of change can be bounded: The mechanisms to change the ordinary permissions are themselves part of the permission structure and are called **administrative controls**. With well defined administrative controls, the need to restructure the system and to (re)verify its security properties from scratch is dramatically reduced or eliminated.

An authorization model is **analyzable for** p if for every authorization state it can be understood where p holds. After all, if one doesn't know what protections are provided how does one know that anything is protected? There are different notions of analyzability. One is ease of understanding by humans. Another is the ability of a program to determine whether a particular property holds, that is the decidability of the property. Decidability is the focus of this paper.

Unfortunately, it has been an elusive goal to design authorization models which simultaneously achieve all three qualities of expressiveness, robustness, and decidability for a security property p . In particular, in many systems, properties are undecidable as was first discovered by Harrison, Ruzzo, and Ullman (HRU) for the low-level property of safety [HRU75].

In ESORICS 2004, we introduced Security Property Based Administrative Controls (SPBAC) authorization model for (overt) information flow security properties [SS04b] of confidentiality information flow and for integrity information flow. The SPBAC is expressive with respect to information flow since any write of x after a read of y must be explicitly allowed via a *mayFlow* permission. The SPBAC is robust because, given an initial authorization state, it is possible to prevent violation of security properties where they initially hold and to arbitrarily grow the number of places the security property holds. In that paper, we showed that it was decidable, given an SPBAC state, to determine whether the definition of a particular *mayFlow* violated any of the information

flow security properties; this, in turn, required the algorithm to determine whether any sequences of *ordinary* operations (consisting of read an object, write an object, add a new user or change membership in a group) could violate the property.

In this paper, we show that it is decidable, given an SPBAC state whether *any* sequence of actions would cause information to flow from l to l' . The question involves not only the ordinary operations, but also the creation of new labels, new groups, and new *mayFlows*. This question is much broader and technically more challenging: Unlike the previous result it involves decidability of the administrative controls. Generalized Biba integrity is similar, but due to space limitations the reader is referred to our technical report [SS05].

The paper is organized as follows: Section 2 describes related work. Sections 3 and 4 together define the (slightly abridged) SPBAC authorization model of [SS04b]; they are included here to make this paper more self-contained. In terms of the layered design of access controls of [SS04a], an authorization model is “layer one.” In Section 5 we show that information flow is decidable for any particular SPBAC state. Finally in Section 6 we conclude.

2 Related Work

It has long been known that in sufficiently dynamic authorization models, analyzing security properties can be undecidable. Harrison, Ruzzo, and Ullman first showed that a low-level property, *safety* was undecidable in their model [HRU75].

Sandhu’s Typed Access Model (TAM) [San92] associates a fixed type with each subject and each object. Although TAM has the same undecidable safety property as HRU, Sandhu showed that if TAM is restricted to be *monotonic*—meaning that privileges can never be removed—(and also have another minor restriction), then the problem is decidable. More recently, Soshi [Sos00] showed that a different, non-monotonic restriction, Dynamic TAM, which allows the types of subjects and objects to change, also has a decidable safety property, under the restriction that only a fixed number of objects can ever be created in the lifetime of the system.

As the above work predates much of the work on administrative controls, they appear not to have been evaluated for robustness, and we are unaware of any work which evaluates their expressiveness. Of course, reference monitors [And72] are expressive but are neither decidable nor robust.

Lattice-based authorization models [Wei69, BL73, Bib77, Den76, San93] are both decidable and robust in the sense that new categories or compartments can be added without affecting existing protections. But they are not expressive for our information flow confidentiality security property, since the security property cannot be violated.

Type Enforcement (TE) [BK85, OR91] is expressive for information flow confidentiality as well as other security properties—for example, restricting the executable which operates on an object. It is also decidable. However, TE provides does not control changes to its authorization state, and hence such changes are unconstrained, so that TE is not robust.

Full Role-Based Access Controls (RBACs) are expressive and robust. RBAC models have traditionally been constructed using either first order predicate logic or graph transformation rules. Unfortunately, either of these constructions can lead to undecidability results. For example, both RBAC’96 [SCFY96] and ARBAC’97 [SBM99] are undecidable [MS99, Cra02]. The most vexing problems for decidability seem to arise from administrative controls, which are the hallmark of RBAC.

Koch and colleagues described an RBAC model based on graph transformations, and showed that it was decidable if no step both added and deleted parts of the graph [KMPP02b, KMPP02a]. This means that no command may both remove and add privileges. Thus, for example, a command to change a user's group, which usually means that the user simultaneously loses and gains privileges, would not be permitted. This restriction can be viewed as a somewhat milder form of monotonicity. Take-Grant [LS77] also obeys this restriction.

Tidswell and Jaeger created Dynamic Typed Access Control (DTAC) which extends TE to dynamic types and is capable of implementing administrative controls [TJ00a, TJ00b]. These were implemented as runtime checks in the operating system to ensure that various safety properties are not violated [JT01]. In this paper, we show how security properties can be analyzed over all operations for the entire system.

We have recently shown that administrative controls for classical Discretionary Access Controls [OSM00] can be implemented in a decidable language [SS04a]. The decidability question they addressed was the safety property. We introduced the SPBAC model in [SS04b] and gave an algorithm to determine the approvals needed for a *single* administrative action. In this paper, we prove the decidability of information flow in the SPBAC model over all possible sequences of administrative and non-administrative actions.

As with Foley, Gong, and Qian [FGQ96] SPBACs make extensive use of relabeling to encode authorization state.

We consider here only overt information flow, not the covert flows which in any event are tied to the execution model [Lam73, GM82].

3 Ordinary Privileges

In this section we describe the ordinary privileges of the SPBAC authorization model, and in the next we describe the administrative privileges. The ordinary entities in the SPBAC authorization model are (1) users, (2) group sets, (3) groups, (4) objects, (5) labels, and (6) permissions.

Each *user* is authenticated and individuals who can use the system are one-to-one with users. A process derives its authority to perform an operation from the user on whose behalf the process executes.

3.1 Group sets and groups

A *group set* is a collection of one or more *groups*; a group is a set of users. Every group is in exactly one group set. A group set is described by a set of group objects and a group set definition.

The *group objects* are used to track group membership. Each group object has a *group label* of the form $\langle U, G \rangle$, where U is a user ID and G is a *group tag* defined in the group definition. The first components of the group labels (i.e. the users) are unique within a group set. As we shall see, G is used to determine what groups U is a member of.

The group set definition \mathcal{G} consists of the following:

Group definitions Each group tag maps to a set of groups, possibly empty. For example, if there is a group object $\langle U, G \rangle$ and $G \rightarrow \{g_1, g_3\}$ then U is a member of g_1 and of g_3 . The group tags are unique to the group set.

Relabel permissions For any two group tags G_1, G_2 in the group set, a group relabel permission $Relabel(G_1, G_2) = g$ enables a member of a group g to change an object labeled $\langle U, G_1 \rangle$ to

$\langle U, G_2 \rangle$, for any U . An undefined $Relabel(G_1, G_2)$ is equivalent to defining $Relabel(G_1, G_2)$ to be the empty group. The group g is either defined in \mathcal{G} or a previously defined group set definition.

Group g is called a **membership secretary** since changing U 's group tag changes the groups to which U belongs. All the **membership secretaries** of a group set are drawn from the same **(membership secretary) group set**.

The membership secretary mechanism which constrains group membership is entirely independent of the security property administrative controls discussed in Section 4.

Creation of group objects Group objects can be created *only* by the group mechanism as follows:

Initial group labels A set of group labels $\langle U, G \rangle$ where each first component is a unique existing user.

New user tag A tag G_n which specifies that when a user U is added to the system, a new group object $\langle U, G_n \rangle$ is created in the group set.

Group set evolution A group set's complete definition must be given at its creation.

In addition to the above group sets, for each user U there exists a singleton group denoted $\{U\}$.

While the above does not describe removal of users, it is trivial to do so; for example, by removing the ability to authenticate the user. In any event, removal operations play no role in the decidability proofs.

Note that group membership can be constrained by relabel rules, for example, to allow groups which only grow or only shrink. Moreover, the tag to group mapping enables groups to have relative structure, such as hierarchy (one group always contains another) or partition (a user cannot be simultaneously a member of two separate groups).

That completes the definition of groups but we now need to describe one issue which arises from their use. A user may be removed from a group via a relabel. To ensure that relative structure properties on group sets hold, a process run on behalf of U which has used privileges which depend on being a member of a group is killed if U is removed from that group.

3.2 Objects, labels, and permissions

Each *object* has a *label* (sometimes called an “ordinary label” to make it clear that we are not referring to group labels). There are four permissions, three standard unary ones and a novel binary permission unique to SPBAC.

3.2.1 Unary Permissions

Privileges to access an object are based on the label of that object. Each unary permission p maps a label l to a group $p(l)$ which has privilege p on objects with label l ; p is either read (r), write (w), or execute (x).³ Thus, for example, $r(l)$ is the group that can read objects labeled l . This mapping is defined when the label is created and thus the group $p(l)$ is *fixed*, although the membership of $p(l)$ can change.

³Write privileges do not imply read privileges. Execute privileges are included here for completeness; they do not play any further role in this paper.

3.2.2 *mayFlow*

We now describe *mayFlow*, a binary permission which controls the information flow.

$mayFlow(l_0, l_1)$ is the group which can write l_1 and after having read l_0 .

Having user $U \in mayFlow(l_0, l_1)$ is a necessary but not sufficient condition for one-step information flow, since *mayFlow* does not include privileges to read l_0 or write l_1 . In order for a process executing on behalf of user U to read l_0 and then write l_1 , we must have

$$U \in r(l_0) \cap w(l_1) \cap mayFlow(l_0, l_1) . \quad (1)$$

Note that condition (1) must hold for *every* label l_0 read prior to writing l_1 .

The *mayFlow* relation need not be defined on *all* label pairs. For pairs on which *mayFlow* is not yet defined, the specified flow may not occur. Moreover, unlike lattice models, *mayFlow* is not transitive, so each allowable flow must be individually specified.

Information flow with downgrade example. Consider the following for labels L and H , $r(H) = w(H) = mayFlow(L, H) = g_2$; $mayFlow(H, L) = g_1$; and $r(L) = w(L) = g_3$ where it is always the case that $g_1 \subset g_2 \subset g_3$.

Then information flow confidentiality holds from L to H and is violated from H to L . The smallest group in this example, g_1 is used to go from H to L since violating confidentiality is the most sensitive operation. Note that in a flow from L to H the information flow confidentiality security property is enforced; in a flow from H to L it is violated. (Other examples are described in [SS04b]).

4 Administrative Privileges

Entities such as groups, labels, objects, permissions, and users may be created at any time.

Conceptually, the system is configured, verified, and then made operational, or goes “live”. To create entities after the system goes live, security property approval is needed where a change violates a security property; if there are no *additional* security property violation then no approval is needed.

We distinguish three separate levels of actions that can be performed in our model:

Ordinary These actions which are governed by the mechanism described in Section 3 and include (1) read and write of objects; and (2) change membership of groups (including the addition of new users)⁴.

SysAdmin Actions Actions which are not ordinary but that do not violate any security property. These actions include the (1) creation of new group sets (and hence new groups); (2) definition of new *mayFlows* not requiring security property approval; and (3) the creation of new labels.

⁴The rationale is that since any user could be allowed to be a membership secretary and therefore change group membership, and since new users gain privileges only through group membership, it is logically consistent to have in the same category everything about how a group evolves.

Security Property (*mayFlow*) Approvals Actions performed by administrators which require approval because they directly or indirectly affect security properties. When security property approvals are requested, an administrator is given all the information needed to make a decision. In this paper, the only security property approvals are of *mayFlows* that violate existing security properties.

The idea here is that SysAdmin actions can be delegated to technical administrators because they do not affect the core security properties of the system.

The SPBAC hierarchy is layered from low to high: (1) groups, (2) ordinary permissions and (3) administrative permissions. The layered design ensures that to implement a given layer only lower layers are used; hence there can be no loops in the layer dependencies. As a consequence of our layered design, administrative controls have no effect on groups, and in particular over group membership. Hence, security property approvals need to be resilient across all future group memberships. (The group membership mechanism restricts group membership, so this is a meaningful goal).

Our focus shall be on the effect of defining new *mayFlows*. (Note that existing permissions cannot be changed⁵ once established).

In subsection 4.1 we will describe the administrative entities that need to be added, beyond what is necessary for ordinary permission, to support changes to the information flow properties of the system. In subsection 4.2 we describe how we keep track of the relevant past actions. In subsection 4.3, we describe the conditions under which security property approvals are required.

4.1 Administrative entities

We now introduce the entities to support administration of information flow security properties: administrative permissions associated with labels.

For information flow, all security property approval is associated with labels. In particular, for each label we define two administrative permissions at the time of label creation:

ac(l) The (administrative) group which can approve exceptions to confidentiality for label l ;

af(l) The (administrative) group which can approve flows into or out of l .

The $ac(l)$ permission is for security property approvals while the other permission, $af(l)$ does not affect security properties.

We shall require that **administrative groups**—those used for administrative permissions—be distinct from **ordinary groups**—those used for ordinary permissions. Furthermore each administrative group is the only group defined in its group set. These properties hold not only for the group set, but also for its membership secretaries (and recursively for their membership secretaries, etc.). The term **administrative group closure** (resp. ordinary group closure) is used to refer to all these groups and their membership secretaries, recursively. The purpose of these restrictions on groups is to ensure that (i) administrative permissions don't interact with ordinary permissions and (ii) that there is no interaction between administrative groups in which one group being nonempty requires another group to be empty. Decidability holds for any mechanism which satisfies these criteria.

⁵However, existing permissions can be added incrementally by defining a new *mayFlow*.

4.2 Tracking past actions

The decidability analysis provided in Section 5 must consider all future actions. In order to track *all* flows, we also need to track actual past information flows. Thus we track:

didFlow(l_1, l_2): the set of labels which could have actually flowed across $mayFlow(l_1, l_2)$. For all pairs of labels l_1 and l_2 the set $didFlow(l_1, l_2)$ is initialized to be the empty set (at the time the second of the labels is created). Let

$$flowed(l) = \bigcup_{l' \in \text{system}} didFlow(l', l) \cup \{l\} .$$

Then the set $didFlow(l_1, l_2)$ is updated every time a process first reads l_1 and then writes l_2 .

$$didFlow(l_1, l_2) \leftarrow didFlow(l_1, l_2) \cup flowed(l_1)$$

Note that the granularity of information is at the label level—not the object level—and hence forms an upper bound on information flow.

The information flow question studied in this paper is, for state s (see Section 5 for definition of state) and labels a and b

$$\mathbf{Flow}(s; a, b) = \text{“Is there any reachable state from state } s \text{ in which } a \in flowed(b)\text{?”} \quad (2)$$

We note that this may seem a more narrow version than the question posed in the introduction, but since first a new label can be created with either a particular existing user or an arbitrary new user (all new users are “born” identically) this also answers the more widely phrased question.

4.3 Security property approvals for *mayFlow* change

As described in Section 3.2.2, $mayFlow(l, l')$'s being defined is a *necessary (but not sufficient)* condition for *direct* (one-step) information flow from l to l' . To compute what flows are *actually possible using only ordinary actions*, we shall use the the term *can flow*.

Definition 1. Information **can flow from l to l'** if and only if there is a sequence of labels $[l = l_1, l_2, \dots, l_n = l']$ such that using only ordinary actions in sequential order for $i = 1 \dots n - 1$ some user u_i can read l_i and then write l_{i+1} . Such a sequence of labels is called a **can-flow path**.

Can-flow paths denote possible *future* flows. To capture all the information about flows, actual past and possible future flows must be combined. We define *flow along a path* to capture past flows.

Definition 2. Let \mathcal{P} be a sequence, or path, of object labels $[l_1, l_2, \dots, l_n]$. There was a **(past) flow along \mathcal{P}** if $l_1 \in didFlow(l_i, l_{i+1})$ for $1 \leq i < n$. There **has been flow from label a to label b** if there was a flow along some path starting at a and ending at b .

Definition 3. Given a past flow along $[l_1, l_2, \dots, l_k]$ and a can-flow path $[l_k, l_{k+1}, \dots, l_n]$ there is an **extended can-flow path** $[l_1, l_2, \dots, l_n]$, because that information has flowed from l_1 to l_k and could flow to l_n .

We now give the formal requirements for confidentiality security property approval which is required when the confidentiality security property is violated. Adding a new $mayFlow(l, l')$ gives rise to a new set of extended can-flow paths. If there exists a new extended can-flow path $\mathcal{P} = [l_1, l_2, \dots, l_n]$ then **Confidentiality approval** from $ac(l_1)$ is needed if it is possible that at some point $r(l_n) \not\subseteq r(l_1) \cup r(l_2) \cup \dots \cup r(l_{n-1})$. A justification for this mechanism and an algorithm to determine when security property approval is required is given in [SS04b].

In addition, approval is always required by $af(l)$ and by $af(l')$, but these are not security property approvals; they are merely SysAdmin actions.

5 Decidability

Decidability is shown by a bounded state space construction. The construction is involved since the natural state space is infinite.

5.1 State space

We first define the **state** which is the SPBAC authorization state.

Definition 4. A **state** of an SPBAC authorization model consists of the following:

1. the set of users;
2. the group set definitions;
3. the group labels of each group set;
4. the set of ordinary object labels;
5. each unary permission definition—that is, $r(l)$, $w(l)$, $x(l)$, $ac(l)$, and $af(l)$ —for every label l ;
6. the defined $mayFlow$ permissions; and
7. the values of all $didFlows$.

There is one type of *transition* between states for each of the following: (1) adding a new user, (2) defining a new group set, (3) changing a group label, (4) defining a new ordinary label, (5) defining a new $mayFlow$, and (6) adding to a $didFlow$ (as a result of doing a write after a read).

Space does not permit listing the initial state and the pre-condition and effect of each type of transition, but they follow fairly directly from the definitions in Sections 3 and 4. For instance, the initial value of $didFlow(l_0, l_1)$ is \emptyset for every two ordinary labels l_1 and l_2 defined in the initial state, and the precondition for a transition updating $didFlow(l_1, l_2) \leftarrow didFlow(l_1, l_2) \cup flowed(l_1)$ is that the intersection of groups $r(l_1) \cap w(l_2) \cap mayFlow(l_1, l_2)$ be nonempty. The last is determined by examining the groups associated with permissions, their group set definitions, and their current group labels.

Definition 5. For state s_0 , the **state space** $\mathcal{SS}(s_0)$ is a directed graph on states that includes s_0 and every state reachable from s_0 by a sequence of transitions, and has edge (s, s') exactly when there is a transition from s to s' .

The **ordinary state space** $\mathcal{SSo}(s_0)$ is the subset $\mathcal{SS}(s_0)$ reachable from s_0 using only ordinary actions. (i.e., not permitting transitions that define new group sets, define new ordinary labels, or define new $mayFlows$).

The set of all reachable states is infinite for three different reasons:

- An unbounded number of new users may be added.
- An unbounded number of new ordinary object labels can be created.
- An unbounded number of new group sets may be created.

We need to finitely bound the above three things, while retaining sufficient information to answer the information flow question $\mathbf{Flow}(s; a, b)$ described in Equation 2. Lemmas 12 and 13, together, show that a bounded number of new ordinary object labels and group sets suffice to answer the information flow question. The number of new users that need to be considered is bounded in Subsection 5.4 where a *restricted augmented* state space is defined and the subsection culminates with our central technical result, Theorem 17, which says that $\mathbf{Flow}(s; a, b)$ is decidable.

We next state a simple but useful proposition: loops can always be removed from flow paths.

Proposition 6. *Let s be a state such that $a \notin \text{flowed}(b)$, and such that there is a state $s' \in \mathcal{SS}(s)$ where there has been flow along a sequence of labels from a to b . Then there is some sequence of transition in $\mathcal{SS}(s)$ in which the flow from label a to label b is along a flow path with no repeated labels.*

Proof sketch: By our definition of flow along a path, which requires $a \in \text{didFlow}(l_i, l_{i+1})$ for every pair of adjacent labels l_i and l_{i+1} on the path, we can simply remove the loop. \square

5.2 Independence of group changes and destroying extended can-flow paths

We state here two lemmas about groups. The first says that changes in group membership are unaffected by any non-ordinary actions. The second says that changes in ordinary group membership can only remove previously possible future flows.

Lemma 7. *If there is a state in $\mathcal{SS}(s)$ where a particular group g in s is nonempty (resp. empty, or contains a particular member), then there is a state where g is nonempty (resp. empty, or contains a particular member) in $\mathcal{SSo}(s)$. Furthermore, that state can be reached by taking exactly the same ordinary actions as were taken in $\mathcal{SS}(s)$.*

Proof sketch: Actions that are not ordinary are the creation of new labels, the creation of new group sets, and the definition of *mayFlows*. None of those affects membership of groups in defined in s . \square

Lemma 8. *Changes in ordinary group membership can destroy an existing extended can-flow path or can-flow path, but can never create one.*

Proof sketch: The definition of can-flow path and extended can-flow path incorporates any sequence of ordinary actions, so includes all changes in ordinary (and non-ordinary) group membership.

To show the destruction of a path consider, as an example, a group $r(l)$ without a new user tag. Then all members of $r(l)$ could be removed (via group object relabels) destroying some path out of l . \square

5.3 Bridges

We next generalize the notion of defining $\text{mayFlow}(a, b)$ to what we call a *bridge*. The purpose of a bridge is to allow a flow between labels which exist in s . The decidability proof will rely heavily on both the types of bridges which need to be constructed and the order of their construction.

Definition 9. *A bridge from label a to label b relative to state s is an extended can-flow path from a to b whose interior nodes do not include any labels existing in s . We will use the term **span** to refer to an edge in a bridge.*

We are interested in a single extended can-flow path from a to b . A flow from a to b is possible iff it is possible to construct such a path. The following lemma restricts the form of bridges we need consider.

Lemma 10. *Let s be a state in which $a \notin \text{flowed}(b)$, and let $s' \in \mathcal{SS}(s)$ be a state in which there has been a flow along \mathcal{P} from a to b . Then there exists a sequence of transitions in the state space in whose final state there has been a flow along \mathcal{P} from a to b such that:*

1. *The only mayFlows added after s are those on \mathcal{P} ;*
2. *Only labels on \mathcal{P} are created;*
3. *Administrative group membership is never reduced;*
4. *New mayFlows along \mathcal{P} are defined in the order traversed; and*
5. *Any new users are added immediately after state s .*

Proof: due to space limitations the proof is in Appendix A.

We next examine the types of bridges that need to be considered. We show that if it is possible to construct such a bridge, it can be done using either a single span bridge or a triple span bridge. We also need to describe the groups used in such bridges, which turn out to be either existing groups, singleton groups, or a special group type we call a *valve* group, defined next.

Definition 11. *A group g is a **valve** iff (i) g can have any user as its only member and (ii) g can be made permanently empty.*

A valve can be constructed as follows: The group set for the valve group has two tags, G_1 and G_2 , and every new user is created with tag G_1 while existing users are created with the tag G_2 . The group tag maps are $G_1 \rightarrow \{\}$ and $G_2 \rightarrow \{g\}$. The only membership secretary in the group set is g , and hence $\text{Relabel}(G_1, G_2) = \text{Relabel}(G_2, G_1) = g$.

A valve is used as follows: First, make g consist of a single specified member u by moving u into g , if necessary, and moving all other members of g outside of g . Second, assign a new *mayFlow* permission to g . Third, u moves an object across the *mayFlow*. And finally, u moves itself out of g (thus ensuring g will be forever empty). The valve is then said to be (permanently) *closed*.

We next give the 3-Span Lemma:

Lemma 12. *If a multi-span bridge relative to state s can be built from l to l' , then a bridge from l to l' can be built with exactly three spans. Furthermore, the bridge can be constructed using only groups that exist in s and one (new) valve group.*

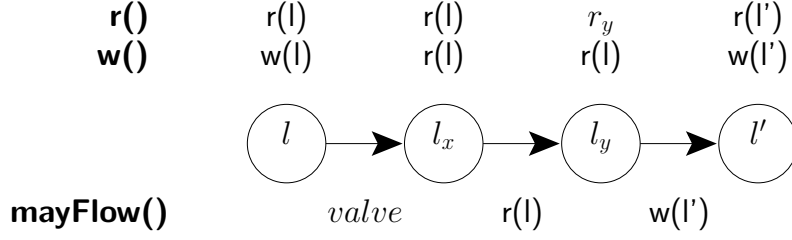


Figure 1: 3 span bridge construction used in the proof of Lemma 12. Read and write permissions are shown over the label and mayFlow permissions under the edge.

Proof. We first apply Lemma 10 to the bridge that must exist by the “if” part of this lemma to get a bridge $\mathcal{B} = [l = n_0, n_1, \dots, n_k, l']$, where the n_i for $i \geq 1$ are new labels, and \mathcal{B} 's *mayFlows* are defined in the order in which they are crossed. Since by hypothesis \mathcal{B} has multiple spans, it follows that $k \geq 1$.

The bridge we construct to prove this lemma is the path $\mathcal{P} = [l, l_x, l_y, l']$ as shown in Figure 1. The groups used to construct the bridge are the existing groups $r(l)$, $w(l)$, and $w(l')$; a new valve group; and a group denoted r_y . We show below that it is sufficient to select r_y to be a singleton user group. (Note that such groups are added automatically upon the addition of a new user.)

The bridge is constructed in the following sequence:

1. Define new labels l_x and l_y , with $r(l_x) = w(l_x) = r(l)$; $r(l_y) = r_y$; and $w(l_y) = r(l)$.
Assign an arbitrary permanently nonempty administrative group to all the administrative permissions for l_x and l_y .
2. Perform all the group membership changes on group sets in state s , made when constructing \mathcal{B} prior to the flow across \mathcal{B} 's *first* span. (Among other things, this will ensure that $r(l)$ is nonempty). No approvals are needed to make $r(l)$ nonempty, by Lemma 7.
3. Construct the first span with valve group by defining $\mathit{mayFlow}(l, l_x) = \mathit{valve}$. No confidentiality approval is needed for the *mayFlow*, since l_x 's readership is the same as l 's readership.
4. Perform ordinary flow from l to l_x using any one member of $r(l)$, which is possible because of step 2.
5. Close the valve so that it cannot be used again. Note that now $\mathit{flowed}(l_x) = \mathit{flowed}(l) \cup \{l_x\}$.
6. Perform all the group membership changes on group sets in s , made when constructing \mathcal{B} prior to the flow across its *penultimate* span, (n_{k-1}, n_k) . (Among other things, this will ensure that $w(l')$ is nonempty). Note that these may also be necessary to destroy some extended can-flow paths which would otherwise require approvals.
7. Define $\mathit{mayFlow}(l_x, l_y) = r(l)$.

For confidentiality approval, choose r_y to be the singleton group containing the user u who caused the flow across the *last* span of \mathcal{B} . At the time that the flow across that final span occurred, $u \in r(n_k)$. Therefore, at the time the penultimate span of \mathcal{B} was approved, the

confidentiality approvals required for flow into n_k based on the group $r(n_k)$ included all confidentiality approvals needed for the labels in $flowed(l)$ to be read by a group potentially including u . Here we will need confidentiality approval from at most those same ac sets plus $ac(l_x)$, which is nonempty.

8. Perform ordinary flow from l_x to l_y , and then any group membership changes that occurred between the definition of the penultimate and last spans of \mathcal{B} in \mathcal{B} 's construction.
9. Define $mayFlow(l_y, l') = w(l')$.

The confidentiality approvals required must have been given when \mathcal{B} was constructed. We need confidentiality approval for the group $r(l')$ at the time that the last span of \mathcal{B} was approved to read everything in $flowed(l) \cup \{l_x, l_y\}$. Confidentiality approvals for (at least) $flowed(l)$ had to be given in the construction of \mathcal{B} , and $ac(l_x)$ and $ac(l_y)$ are nonempty.

□

The previous lemma bounds the length of multi-span bridges and the groups and users used in their construction. We next limit the set of single span bridges that we need to consider. The next lemma says that if it is possible to define $mayFlow(l, l')$ at all, then it can be done without introducing any new groups.

Lemma 13. *Let s' be a state reachable from s without defining any new $mayFlows$. If $mayFlow(l, l')$ can be defined in state s' to create a bridge from l to l' , then $mayFlow(l, l') = g$ can be defined to create a bridge from l to l' from state s where g is an existing group.*

Proof. Say in state s' the definition $mayFlow(l, l') = g$ is approved for some g that does not exist in s . Notice that g must be in a group set that does not exist in s , since all the groups in a group set are defined at once. Notice also that $g \cap r(l) \cap w(l')$ can be made nonempty, since a bridge is created, which therefore must be a can-flow path.

Now any new extended can-flow paths that are created when $mayFlow(l, l') = g$ is defined would also be created by defining $mayFlow(l, l')$ to be, say, $r(l)$. This is because there are only two types of choices for the $mayFlow$ group which limit the number of extended can-flow paths. One is if there is no flow possible at all over the $mayFlow$, for example by defining it to be the empty group. However, that is not possible here. The other is if the group could potentially “block” or be “blocked” by some other group on an extended can-flow path; for instance, if this group must be made empty in order to make another group on the path nonempty. However, that relation is possible only between two groups in the same group set, and hence the group would need to be part of an existing group set. □

5.4 Decidability of information flow

We now introduce the *augmented state* which can succinctly represent an arbitrary number of new users, denoted \top , added to the system. The symbol \top is used to avoid a tedious counting argument bounding the number of new users required to determine whether a membership secretary can be made nonempty. The augmented state differs from the regular state by additional group labels:

Definition 14. *An augmented state is the same as a regular state with the addition that for each group tag G , $\langle \top, G \rangle$ can be a group label. Let s^\top denote the augmented state formed from*

regular state s and (1) adding a group label $\langle \top, G_i \rangle$ for each group tag G_i that is a new user tag of some group set in s and (2) adding $n - 1$ new users, where n is the number of labels in s .

Next we introduce the restricted augmented state space which is a bounded state space central to our decidability proof. It is *restricted* since it allows neither new group sets, new labels, nor new users to be added. Since it is augmented, an additional rule is needed beyond those defined in the regular state space: If there is a definition $Relabel(G, G') = g$, g is not empty, and there is a group label $\langle \top, G \rangle$ but not $\langle \top, G' \rangle$, then there is a successor state which adds $\langle \top, G' \rangle$.

Definition 15. *The **restricted augmented state space** $\mathcal{SS}^\top(s^\top)$, is the set of augmented states reachable from s^\top either by transitions that are in $\mathcal{SS}(s)$ —except for those which create new group sets, create new labels, or add new users—or by the relabel transition described above. The **restricted regular state space from s** $\mathcal{SS}^r(s)$ are states reachable from s in $\mathcal{SS}(s)$ without defining either new labels or new group sets.*

Note that the restricted augmented state space is finite, having removed the three issues that make the state space infinite. The next lemma says that the restricted regular state space and restricted augmented state space answer the same information flow question.

Lemma 16. *Information flow from label a to label b is possible in $\mathcal{SS}^r(s)$ if and only if it is possible in $\mathcal{SS}^\top(s^\top)$.*

Proof: due to space limitations the proof is in Appendix A.

Now we give the decidability result, whose proof consists of showing that restricted regular state space transitions are sufficient to answer the information flow questions which means that the restricted augmented state space is sufficient to answer these questions. And since the restricted augmented state space is bound, the information flow question is decidable.

Theorem 17. *There is an algorithm to decide whether $\mathbf{Flow}(s; a, b)$ is true for any given a state s and two labels a and b that exist in s .*

Proof. If there is an extended can-flow path from a to b in state s , then the algorithm answers yes. An algorithm to determine whether there is an extended can-flow path is given in [SS04b].

Next consider the case where there is not an extended can-flow path from a to b . Assume for now that information flow from a to b is possible; we will give a finite construction that is guaranteed to find such a flow. By Proposition 6, we need consider only loop-free sequences of labels consisting of labels existing in s and new labels. We consider all possible orderings that begin with a and end with b of any subset of the labels existing in s . By Lemma 12, for each such sequence, we need add at most a bounded number of new labels (less than two new labels per existing label in s to create bridges). By Lemmas 12 and 13, only a bounded number of new group sets are needed (one valve group per bridge, hence less than the number of labels existing in s).

By the “only if” part of Lemma 16, for a sequence of labels that corresponds to the flow in the regular state space, we will find a flow in the restricted augmented state space.

By the “if” part of Lemma 16, if we find flow in the restricted augmented state space, then flow really is possible. \square

6 Conclusion

Security properties are high-level statements about the protections provided by a computer system’s authorization model. We have focused here on one of the oldest, and best studied of the security properties, information flow confidentiality.

We have previously presented *Security Property Based Access Controls (SPBAC)* [SS04b]. In an SPBAC, the focus is on security properties. SPBACs are designed around the principle that it is always safe to honor security properties, but violating them—while necessary in real systems—requires care and administrative approval from those who are charged with overseeing the system security.

An SPBAC uses commonplace specifications. Labels, simple patterns, unary and binary permissions on labels, relabels, and groups are the base mechanisms. But despite the simplicity of the building blocks, it is both expressive and robust.

Of course, SPBAC is not the first authorization model to be both expressive and robust, but achieving systems which are simultaneously expressive, robust, and decidable for a security property has been elusive. Although there are systems which are known to support two of these goals, until now there have been none which support all three.

The contribution of this paper is to show that these information flow properties are decidable, meaning we can algorithmically answer the question of whether information contained in an object with label l_1 can ever flow into an object labeled with l_n . Since a new label can be computed with a particular existing user or an arbitrary new user (all new users are “born” identically) this also answers the more widely phrased information flow question about whether a user who could not view the information as originally classified can view it after it flows to some other label, that is, can there be a loss of confidentiality? We have further shown in a technical report [SS05] that it also holds for information flow integrity.

We believe that an SPBAC can therefore combine the best properties of lattice-based access controls (decidable, property based, and extensible), type enforcement (decidable and selective enforcement and violation of security properties), and role-based access controls (flexibility and support for administrative controls).

References

- [And72] James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, AFSC, Hanscom AFB, Bedford, MA, October 1972. AD-758 206, ESD/AFSC.
- [Bib77] K. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, MITRE Corp, Bedford, MA, 1977.
- [BK85] W. E. Boebert and R. Kain. A practical alternative to hierarchical integrity policies. In *8th National Computer Security Conference*, pages 18–27, 1985.
- [BL73] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, Mitre Corporation, Bedford MA, 1973.
- [Cra02] Jason Crampton. *Authorizations and Antichians*. PhD thesis, Birkbeck College, Univ. of London, UK, 2002.

- [Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM (CACM)*, 19(5):236–243, 1976.
- [FGQ96] Simon Foley, Li Gong, and Xiaolei Qian. A security model of dynamic labeling providing a tiered approach to verification. In *Proc. IEEE Symp. Security and Privacy*, pages 142–154, 1996.
- [GM82] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. Security and Privacy*, pages 11–20, 1982.
- [HRU75] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating system. In *Symposium on Operating Systems Principles*, pages 14–24, 1975.
- [JT01] Trent Jaeger and Jonathon E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security (TISSEC)*, 4(2):158–190, 2001.
- [KMPP02a] Koch, Mancini, and Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proc. European Symp. Research in Computer Security (ESORICS)*, pages 229–243. LNCS, Springer-Verlag, 2002.
- [KMPP02b] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):332–365, 2002.
- [Lam73] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM (CACM)*, 16(10):613–615, 1973.
- [LS77] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.
- [MS99] Qamar Munawer and Ravi Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *INFOSEC99: International Conference on Information Security*, 1999.
- [OR91] R. O’Brien and C. Rogers. Developing applications on LOCK. In *Proc. 14th NIST-NCSC National Computer Security Conference*, pages 147–156, 1991.
- [OSM00] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.
- [San92] Ravi S. Sandhu. The typed access matrix model. In *Proc. IEEE Symp. Security and Privacy*, pages 122–136, 1992.
- [San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.
- [SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):105–135, 1999.

- [SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [Sch00] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.
- [Sos00] Masakazu Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proc. European Symp. Research in Computer Security (ESORICS)*, volume 1895 of *Lecture Notes in Computer Science*, pages 106–121. Springer-Verlag, 2000.
- [SS04a] Jon A. Solworth and Robert H. Sloan. A layered design of discretionary access controls with decidable properties. In *Proc. IEEE Symp. Security and Privacy*, pages 56–67, 2004.
- [SS04b] Jon A. Solworth and Robert H. Sloan. Security property-based administrative controls. In *Proc. European Symp. Research in Computer Security (ESORICS)*, volume 3139 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2004.
- [SS05] Jon A. Solworth and Robert H. Sloan. On the decidability of administrative controls for information flow confidentiality and integrity. Technical report, University of Illinois at Chicago, 2005.
- [TJ00a] Jonathan F. Tidswell and Trent Jaeger. Integrated constraints and inheritance in DTAC. In *Proc. of the ACM Workshop on Role-Based Access Controls (RBAC)*, pages 93–102, 2000.
- [TJ00b] Jonathon Tidswell and Trent Jaeger. An access control model for simplifying constraint expression. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 154–163, 2000.
- [Wei69] Clark Weissman. Security controls in the ADEPT-50 time-sharing system. *Proc. FJCC, AFIPS*, 35, 1969.

A Proofs of lemmas

Lemma 10. *Let s be a state in which $a \notin \text{flowed}(b)$, and let $s' \in \mathcal{SS}(s)$ be a state in which there has been a flow along \mathcal{P} from a to b . Then there exists a sequence of transitions in the state space in whose final state there has been a flow along \mathcal{P} from a to b such that:*

1. *The only `mayFlows` added after s are those on \mathcal{P} ;*
2. *Only labels on \mathcal{P} are created;*
3. *Administrative group membership is never reduced;*
4. *New `mayFlows` along \mathcal{P} are defined in the order traversed; and*
5. *Any new users are added immediately after state s .*

Proof. By Proposition 6, assume WLOG that \mathcal{P} has no loops. For each of constraints 1– 5, we show that it is possible to modify the flow to conform to the constraint without removing the flow.

(1) `mayFlows` not on \mathcal{P} are obviously not used for flow on \mathcal{P} . Newly created `mayFlows` can only introduce more extended can-flow paths, making it more difficult to obtain future *ac* approvals of `mayFlows`. So remove all such `mayFlow` definitions.

(2) Once those `mayFlow` definitions are removed, we can remove new label definitions not on \mathcal{P} , since they will have no `mayFlow` defined in or out, and hence will not have any effect.

(3) Next, delete any transitions that reduced the membership in any administrative group (*ac* or *af*), or the membership in any other group in the administrative group closure. This is possible because (a) administrative groups are separate from ordinary groups and (b) administrative groups do not interact with each other as a consequence of only one group per group set being used.

Constraint (4) is the most subtle. We are going to move certain `mayFlow` definitions later in time. Let nm_1, \dots, nm_k be the new `mayFlows` on \mathcal{P} in the order *traversed*. Constraint (3) means that if the definition of nm_i is moved to be later in the sequence, then the administrative groups that approved its definition at the original time will still be nonempty. We need therefore only to worry about *additional* approvals that may be needed because of the change.

Now consider all the `mayFlow` definitions that are made. We argue that we can move all definitions of nm_i for $i > 1$ later in the sequence of events so that they all are defined immediately after the definition nm_1 . In particular, say nm_{i^*} was the last new `mayFlow` defined before nm_1 was defined in the original chronological ordering, and consider what happens if we move the definition of nm_{i^*} to immediately follow the definition of nm_1 . Approval of the definition of nm_1 is not affected, since having fewer other `mayFlows` defined could only reduce the number of extended can-flow paths, and hence require fewer approvals.

Now, what about the approval of nm_{i^*} ? Two things may be different with its chronologically later approval. First, some additional changes in membership in *r*, *w*, and/or `mayFlow` groups may have taken place before the definition of nm_{i^*} in this new ordering, but by Lemma 8, such changes could only reduce the number of approvals needed. Second, there may be some new extended can-flow path that uses both nm_1 and nm_{i^*} that exists only when approving nm_{i^*} in the new order, because nm_1 is already defined. However, this same path would otherwise have been introduced at the request to define nm_1 . This is because we have exactly the same `mayFlow` definitions and group memberships at the point where nm_{i^*} is defined in this new ordering with

the definition of nm_{i^*} immediately following the definition of nm_1 as we had in the old ordering at the point where nm_1 was defined. Any approvals needed for such new extended can-flow paths can be obtained now, since administrative group memberships are the same as at the time of the definition of nm_1 in the original order.

By a similar argument, we can next move the second to last new *mayFlow* definition made before the definition of nm_1 to a position immediately after the definition of nm_1 (and hence immediately before the definition of nm_{i^*}). Continuing in this fashion, we can slide all *mayFlow* definitions that originally came chronologically before nm_1 so that they occur after nm_1 .

We then repeat the process to move all nm_i with $i > 2$ so that they occur immediately after the definition of nm_2 , and then continue in a similar way until we have all the nm_i defined in the desired order.

Finally, (5) holds since the only transition which requires the *absence* of users is a *mayFlow* definition, which considers the absence only after the addition of all possible users (i.e., after allowing for any possible ordinary action, including the addition of new users), and hence their earlier existence is immaterial. \square

Lemma 16. *Information flow from label a to label b is possible in $\mathcal{SS}^r(s)$ if and only if it is possible in $\mathcal{SS}^\top(s^\top)$.*

Proof. \Rightarrow :

Let $\mathcal{P} = [a = l_1, \dots, l_n = b]$ be the flow path in $\mathcal{SS}^r(s)$ (n must be at most the number of labels existing in state s). Let u_i be the user who carries out the flow from l_i to l_{i+1} . If u_i is not a user in state s , WLOG let us give the name u_i to one of the new users added to s^\top (all new users are initially the same). At most $n - 1$ users participate in this flow, so that the $n - 1$ new users added to state s^\top suffice.

Now, in general, make the same transitions in the augmented state space $\mathcal{SS}^\top(s^\top)$ as were made in $\mathcal{SS}^r(s)$ to accomplish the flow, with the following exceptions.

1. Do not add any new users.
2. Omit any reads or writes that are not part of the flow path.
3. Call a user in some state of $\mathcal{SS}^r(s)$ “added” if that user did not exist in state s .

For any transition in $\mathcal{SS}^r(s)$ that relabels an added user’s group tag from G to G' , in $\mathcal{SS}^\top(s^\top)$ follow the transition that adds group object $\langle \top, G' \rangle$ if it does not already exist. By our construction, $\langle \top, G \rangle$ will be present to be relabeled.

Observe that the same flow will take place in the augmented state space as did in $\mathcal{SS}^r(s)$.

\Leftarrow : The “extra” thing that could cause information flow to occur in the augmented state space is that \top could be used to make some membership secretary nonempty, permitting it to relabel something. However, the augmented state space is constructed so that if \top is making a group nonempty, then any number of regular, named new users could have been added and made that group nonempty as well. \square