A Theory of Role Composition

UCLA CSD Technical Report TR-080012

Jeffrey Fischer    Rupak Majumdar
University of California, Los Angeles
{fischer,rupak}@cs.ucla.edu

April 14, 2008

Abstract

We study the access control integration problem for web services. Organizations frequently use many services, each with its own access control policies, that must interoperate while maintaining secure access to information. The integration problem is to take the set of such services and to find a globally consistent access control policy that ensures no authorization failures or information disclosures for the system composed from the services. Currently, this is performed manually, and incurs high administrative overhead and high risks for errors. We give a sound and complete algorithm for access control integration by reducing the problem to Boolean constraint solving. We have implemented ROLEMATCHER, a tool to infer global role-based access control schemas for a set of services, and show on examples that it can quickly infer global roles for composed systems, or determine the absence of a globally consistent role schema.

1 Introduction

A key function in any enterprise security infrastructure is access control, which specifies the valid ways in which users can access resources and perform operations. Role-based access control (RBAC) is a frequently-used access control mechanism for enterprise systems [11]. In RBAC, roles represent functions within a given organization; authorizations for resource access are granted to roles rather than to individual users. Users “play” the roles, acquiring the privileges associated with the roles, and administrators grant or revoke role memberships.

While RBAC provides an elegant and scalable access control mechanism for a single system, organizations frequently deploy many interacting applications, each with its own RBAC policy. These individual policies must be integrated into a global policy for the organization. For example, web portal applications combine content from several back-end services in a single web interface. Portal frameworks generally provide their own RBAC policy enforcement, which must be configured to be consistent with the individual applications displayed on the portal. Currently, this global access control policy is designed manually, and this is both labor intensive and error-prone. The goal of our work is to automate this process.

In particular, we develop an algorithm that computes, if possible, a global RBAC policy from the RBAC policies of the different applications, such that the following incompatibilities cannot occur:

- In the processing of a request, a service in one component may call other services in different components. Even though the user had access rights to the original service, they might not have access to the called services, leading to indirect authorization errors. Such errors are difficult to prevent through testing, since access rights are usually assigned uniquely to each user.
- When confidential data is passed between ser-
vices, the receiver could potentially disclose that data to users and services which do not have the necessary access rights in the source system. This may violate the intent of the source system’s security policy.

We call this the global schema inference problem. A global schema is a system-wide notion of roles, and a mapping from local roles to (possibly multiple) global roles, such that with these global assignments of roles to users, there are no indirect authorization errors (the global roles are sufficient) or information leaks (the global roles are non-disclosing). Our main result is a constraint-based algorithm to infer, given the local role assignments for a set of components, whether there exists a sufficient and non-disclosing global schema that is consistent with the local roles in each component.

We make the following contributions in this paper. First, we study the integration problem abstractly by defining an interface formalism for components that specify both the roles required to access each service and the other services that may be invoked by a service, as well as a simple language of service invocations. We define the operational semantics of the language, formalizing indirect authorization errors. Given two components, each with their independent notion of roles, we define global role schemas that group together roles from each service such that there are no indirect authorization errors. Global role schemas are inferred using a constraint-based algorithm that reduces the inference problem to Boolean constraint solving. In addition to requiring sufficiency, the constraints also ensure other desired properties: separation, role ascription, and minimality. Separation ensures that two different roles from the same component are never merged in a global role (i.e., local role semantics are maintained). Role ascription allows the administrator to specify that certain roles across components should be merged. Minimality ensures the principle of least privilege: each global role must contain only required local roles and no extra roles. As might be expected, the global schema inference problem is NP-complete.

Next, we augment service components with information flow that additionally models information flow between services. We extend our language of service invocations with requests for information and responses containing information, and formalize information disclosure by providing the operational semantics of service calls in the presence of system-wide information flow. We provide a polynomial time static analysis that tracks information flow in the system. Using this analysis, we extend the global schema inference algorithm to produce minimal global role schemas that are sufficient as well as non-disclosing.

A possible alternative to global compatibility is to have one global role scheme managed and enforced centrally in the organization. In practice, this can be difficult, or impossible, since each application can be locally administered by different groups, or even be legacy applications where the notion of access control is built into the system. Also, external enforcement approaches may not necessarily capture all the places where an application discloses sensitive data (e.g., data included in other pages or disclosed through integrations). Alternate, ad hoc mechanisms to ensure interoperability do not provide security guarantees and can lead either to breaches of confidentiality or violation of least privilege.

We have implemented ROLEMATCHER, a tool for global role schema inference. Our implementation shows that our algorithm infers global role schemes within a few seconds for small examples, and is expected to scale well for large collections of systems. In addition, we performed a case study inspired by an industrial problem where an IT management company wanted to integrate two applications, one (App A) with a fixed RBAC scheme, and another (App B) in which the customer builds site-specific roles. For this problem, ROLEMATCHER can produce, for each customer site, a globally consistent view for the entire system that unifies customer-specified roles in App B, the fixed roles in App A, and the role schemes of any site-specific applications that the customer decides to install.

2 Example

We demonstrate our techniques on a hypothetical healthcare information system at a medical clinic.
The clinic has three applications:

- Clinical management: this application manages the scheduling of patients and captures the actions performed by doctors and nurses.
- Laboratory information system: this application tracks the tests to be performed and their results.
- Patient records: this application maintains historical data about each patient’s health.

Each system provides one or more web services, which expose a set of callable methods and encapsulates access to the underlying data and application functionality. These services are protected by Role-Based Access Control (RBAC). A set of roles is associated with each service and with each user. To access a service, the set of active roles for the current user must include at least one of the roles required by the service.

Unfortunately, each application has its own RBAC schema, so integrating these systems requires reconciling multiple schemas. Figure 1 lists the services provided by each application and the roles required for accessing these services. We prefix application-local role names with a unique letter for each application (C, L, or P). We call the roles defined within a given application local roles.

Suppose the clinic adds a new web portal which provides convenient web access across the other three applications (Figure 2(a)). The application does not store any confidential data locally. Instead, when the user requests a page, the portal makes service calls to the other applications using the requesting user’s login.

Consider the PatientMedData service of the portal which permits doctors and nurses to see the relevant medical information for a patient. Figure 2(a) shows the services called by PatientMedData: the Vitals and CareOrders services of Clinical Management, the TestResults service of Laboratory, and the PatientHistory service of Patient Records. The CareOrders service, in turn, calls the TestOrders service to retrieve details of tests that have been ordered. This service is accessible to users with either the W:Doctor or W:Nurse roles.

Global Roles. Since each application has its own notion of roles, it is difficult to determine whether a given user will have all the access permissions needed to retrieve the data for each page. This is the global role compatibility problem: does there exist some global set of roles that represent sets of local roles from the different applications that can be used to maintain consistent user to role mappings? The global role compatibility problem takes as input the applications and their role requirements, and produces, if possible, a set of global roles and a mapping from each global role to a set of local roles such that the following constraints are satisfied:

1. [Separation] No two local roles from the same application should be mapped to the same global role. This ensures that the semantics of authorization within an application is not changed by the global map. Otherwise, administrators will be unable to independently assign the two roles to users. Also, this restriction prevents degenerate solutions, such as assigning all local roles from a given application to the same group.

2. [Sufficiency] For each service of the web portal application, services that it (transitively) calls must include all of the global roles required by the web portal service. Thus, if a user has any one of the required roles for the PatientMedData service, they will have a required role for each called service. This ensures no indirect au-
3. **[Ascriptions]** Administrators may optionally specify sets of local roles that must map to the same global role. This permits the representation of semantic constraints specific to an application domain.

A global role mapping is the *minimal* mapping which satisfies the constraints: if there are no calls combining two roles on separate systems, then the roles should be mapped to two separate global roles. For example, the C:Receptionist and L:Billing roles are unrelated to each other or any other roles through calls. Thus, they should be mapped to unique global roles. Minimality ensures a form of least privilege — a global role mapping should not give users any more access rights than strictly necessary to accomplish their objectives.

Note that a given local role can map to more than one global role. This may occur when one system has a less precise security model than the others. In our example, the Laboratory system has only a clinician role for both doctors and nurses, while the Clinical Management application distinguishes between doctors and nurses. By requirement 1 above, we need to maintain two separate global roles for doctors and nurses. If the Vitals service of Clinical Management is used in conjunction with the TestResults service of Laboratory, we need to map Clinician to both of these global roles.

**Global Schema Inference.** We solve the global schema inference problem by formulating it as a Boolean constraint satisfaction problem. Notice that requirement 1 constrains W:Doctor and W:Nurse to be in different global roles (Constraint (1)), and similarly C:Doctor and C:Nurse must be in different global roles (Constraint (2)). Since PatientMedData invokes CareOrders, the set of global roles for PatientMedData must be included in the set of global roles for CareOrders (Constraint (3)). To reflect the idea that doctor and nurse roles should be common across applications, we add an ascription to force the roles W:Doctor and C:Doctor to map to the same global role (Constraint (4)) and, similarly, an ascription to force the roles W:Nurse and C:Nurse to the same global role (Constraint (5)).
Surprisingly, it is not possible to find a mapping which satisfies all these constraints. From Constraints (1) and (2), the Doctor and Nurse roles within each service must be mapped to different global roles. Constraint (3) forces C:Doctor to map to both W:Doctor and W:Nurse. However, this violates the ascription constraints, as W:Nurse should be mapped to role C:Nurse. Thus, our initial portal design does not admit global role schemas. With a tool to infer global role schemas, we can catch such problems at design time rather than when users are assigned to roles and attempt to use the system (as common with ad hoc approaches to access control integration).

Now consider an alternative design of the web portal where we split the PatientMedData service into two services: PatientMedDataD, which contains all the data from the original PatientMedData, but is only accessible to the W:Doctor role, and PatientMedDataN, which does not include data from the CareOrders service, and is accessible to the W:Nurse role. In this case, we obtain the (global) role mappings from Figure 3(a) that maintains consistent global authorization across applications.

Roles and Information Flow. We now extend the role mapping algorithm in the presence of information flow. If one application keeps a copy of protected data from another application, it must control access to this copied data. Otherwise, a user that does not have access to the original application may be able to retrieve the same data through the target application. In our example, the Patient Records application maintains an archive of data from the Clinical Management and Laboratory applications. Thus, Patient Records must deny access to any users which do not have access to both the Clinical Management and Laboratory applications.

As shown in figure 2(b), the Patient Records application is populated with data in the following manner. The DataSync process periodically calls the Vitals and CareOrders services to retrieve patient clinical data older than a certain age, saves this data to the PatientHistory service, and then deletes the original copies from Clinical Management. This DataSync process runs as a super user on both systems, and thus does not have any issues with accessing the appropriate services. The TestResults service periodically connects directly to PatientHistory as a super user and saves any new test results since the last update. Finally, as with the web portal example, the CareOrders service makes a call to the TestOrders, but only relays the resulting data to its caller, without saving it locally. When computing the global roles for this scenario, we enforce the same properties as listed before. In addition, we want to ensure that the users which can access the target service of a data synchronization are always a subset of the users which can access the source service. To implement this, we use sets of roles as a proxy for sets of users and thus add a new requirement:

4. [Information Flow] Whenever data flows from one service to another, the target service’s global roles must be a subset of the source service’s roles.

This ensures that the target service provides at least the same level of access control for the data as its originating service. Note that information may flow in the opposite direction as a call (e.g., the calls to Vitals and CareOrders by DataSync).

For the data synchronization scenario, we obtain the following additional constraints for the doctor and nurse related roles from information flow considerations:

1. From the information flow from Vitals to PatientHistory, the global roles for P:Clinician must be a subset of the global roles for the set \{ C:Nurse, C:Doctor \}.
2. From the information flow from CareOrders to PatientHistory, the global roles for P:Clinician must be a subset of the global roles for C.Doctor.

3. From the information flow from TestResults to PatientHistory, the global roles for P:Clinician must be a subset of the global roles for L:Clinician.

If we solve these constraints to find a set of global roles, we obtain the mapping from Figure 3(b). Note that this mapping shuts nurses out from accessing the CareOrders service (by excluding role P:Clinician) — it exposes data from CareOrders, which is only visible to doctors. A naive mapping of roles that this mapping uses from accessing the PatientHistory service (by excluding role P:Clinician) — it exposes data from CareOrders, which is only visible to doctors. A naive mapping of roles that allows both doctors and nurses to access PatientHistory would subvert the access controls applied to CareOrders. This could be a serious issue, violating privacy regulations (such as HIPAA) and opening the system to abuse.

3 Semantics of Roles

We now define an interface describing an application’s services, the roles which may access each service, and the possible outbound calls made by each service. We then describe the runtime behavior of these interfaces using a small-step operational semantics. This allows us to prove properties relating the static structure and dynamic behavior of such systems.

3.1 Services and their Semantics

Let Names be a set of web service names and Users a set of users. A web application \( A = (\text{Roles}, \text{Services}, \text{Perm}) \) consists of a set of roles \( \text{Roles} \), a set of services \( \text{Services} \), and a user permission mapping \( \text{Perm} : \text{Users} \rightarrow 2^{\text{Roles}} \) from the (global) set of users to subsets of roles in \( \text{Roles} \). A (web) service \( S = (n, R, C, M) \) in \( \text{Services} \) consists of a name \( n \in \text{Names} \), a subset of roles \( R \subseteq \text{Roles} \) denoting the required permissions to call \( n \), a set of called service names \( C \subseteq \text{Names} \), and a mapping \( M : C \rightarrow 2^{R} \) from the services in \( C \) to subsets of \( R \). We write \( A.\text{Roles} \), \( A.\text{Services} \), and \( A.\text{Perm} \) to refer to the roles, services, and user maps of \( A \), respectively, and for a service \( S \), we write \( S.n \), \( S.R \), \( S.C \), and \( S.M \) to reference the components of a service. We assume that there is exactly one service with name \( n \), and we write \( \text{Svc}.n \) for that service. With abuse of notation, we identify a service \( S \) with its name \( S.n \), and say, e.g., that a service \( n \) is in an application \( A \).

The required roles are disjunctive — one of the roles must be satisfied to call the service. The mapping \( M \) represents a more precise subset of the roles known to be active when calling a service.

A system \( Sys \) consists of a set of applications, where we assume that the services and roles of the applications are pairwise disjoint. With abuse of notation, we speak of a service or role in a system for a service or role in an application in the system. Thus, we speak of a service \( S \in \text{Sys} \) if there is an application \( A \in \text{Sys} \) such that \( S \in A.\text{Services} \). We write \( \text{AllRoles} = \bigcup_{A \in \text{Sys}} A.\text{Roles} \) for the set of all (local) roles in system \( \text{Sys} \).

We say that a system \( \text{Sys} \) is well-formed if, (a) [service names are unique] for each \( n \in \text{Names} \), there is at most one service \( S \in \text{Sys} \) with \( S.n = n \), (b) [all called services exist] for each application \( A \in \text{Sys} \), each service \( S \in A.\text{Services} \), and each called service name \( n \in S.C \), there exists an application \( A' \in \text{Sys} \) and a service \( S' \in A'.S \) such that \( S'.n = n \). We say that a system has non-redundant roles if no two roles are assigned to the same subset of the services, formally, if there does not exist an application \( A \) and roles \( r_1, r_2 \in A.\text{Roles} \) such that for all services \( S \in A.\text{S} \), we have \( r_1 \in S.R \) iff \( r_2 \in S.R \). Well-formedness and non-redundancy are syntactic checks, and henceforth we assume all systems have both properties.

Operational Semantics. A system represents the composition of several web service applications, each with its own notion of roles and web services. Users can invoke a service in the system. The roles determine if the user has sufficient permissions to use the service as well as services transitively invoked by the called service. That is, before executing a user-invoked service request, every service \( S \) first checks if the initiator of the request (the user, or the service invoking the call) has appropriate permissions (roles) — determined by the roles \( S.R \) — to execute the re-
quest. If the initiator has permissions to call the service, it is executed to completion (this might involve calls to other services), otherwise there is an authentication failure. We formalize the runtime behavior of service invocations using service call expressions and their operational semantics.

Service call expressions are generated by the following grammar:

\[ e ::= \text{Call}(n, u) \mid \text{Eval}(n, u) \mid \text{Done} \mid \text{AuthFail} \mid e; e \]

A Call expression represents a service call before being checked for permissions; an Eval expression represents the evaluation of a service, the symbol Done represents the successful completion of a service call; the symbol AuthFail represents early termination of a service call due to an authentication failure. The sequential composition of two expressions is represented by \( e_1; e_2 \). The variable \( n \) ranges over service names and \( u \) ranges over users. For clarity, we omit other control structures from this core language, they do not introduce additional conceptual difficulties.

Figure 4 defines the small step operational semantics of service calls, formalized by a binary relation \( \rightarrow \) on service call expressions.

Evaluation starts with a single Call expression. Rules E-CALLEVAL and E-CALLFAIL represent the checking of the user’s roles against those required for the service. If the check is successful, the call steps to the evaluation of the service. Otherwise, evaluation stops with the AuthFail symbol. An Eval expression may step to a service call (rule E-EVALLCALL), successful termination of the subcomputation (rule E-EVALDONE), or a sequence of two Eval expressions (rule E-EVALSEQ). Duplication of Eval expressions captures the nondeterministic nature of services: a service can call any service in its call set zero or more times. In addition, a service call is not guaranteed to terminate. The E-SEQRED and E-SEQDONE rules permit reduction of expression sequences by evaluating the first expression in the sequence.

We write \( \rightarrow^* \) for the reflexive transitive closure of the \( \rightarrow \) relation. A direct service invocation is an expression of the form \( \text{Call}(n, u) \) representing the direct call of a service by a user.

Proposition 1 (Evaluation) Let \( \text{Sys} \) be a well-formed system, \( n \in \text{Name} \text{s} \) a service in \( \text{Sys} \), and \( u \in \text{Users} \) a user. Then, the evaluation via \( \rightarrow^* \) of the call \( \text{Call}(n, u) \) either diverges or eventually terminates with Done or AuthFail.

Accessibility and Sufficiency. Let \( \text{Sys} \) be a system, \( n \in \text{Name} \text{s} \) a service in \( \text{Sys} \), and \( u \in \text{Users} \) a user. The call \( \text{Call}(n, u) \) is accepted by \( \text{Sys} \) if there is a service \( S \in \text{Sys} \) with \( S.n = n \) and \( S.R \cap A_i.\text{Perm}.u \neq \emptyset \). Otherwise \( \text{Call}(n, u) \) is rejected. Intuitively, a call to service \( n \) by user \( u \) is accepted if the user has at least one of the roles required to execute the service named \( n \). In this case, the call evaluates in one step (by rule E-CALLEVAL) to \( \text{Eval}(n, u) \).

For a system \( \text{Sys} \), the function \( \text{rolesOf} : \text{Users} \rightarrow 2^\text{AllRoles} \) maps each user \( u \) to the set of roles available to \( u \), that is, \( \text{rolesOf} u = \bigcup_{A \in \text{Sys}} A.\text{Perm}.u \). A set of services \( \text{Services} \) is accessible to a set of roles \( R \) if for each service \( S \in \text{Services} \), there exists an \( r \in R \) such that \( r \in S.R \). Similarly, a set of services \( \text{Services} \) is accessible to a user \( u \in \text{Users} \) if \( \text{Services} \) is accessible to \( \text{rolesOf} u \). In this case, all calls by \( u \) to any service in \( \text{Services} \) will be accepted; however, transitive calls made by these services may cause authentication failures.

We wish to ensure that if a call is accepted by a system, no further authorization errors can occur. This is provided by the stronger notion of sufficiency. We say that a set of roles \( R \) is sufficient if, for every user \( u \) with \( \text{rolesOf} u = R \) and service \( S \) accessible to \( R \), there is no trajectory \( \text{Call}(S.n, u) \rightarrow^* \text{AuthFail} \). A system is sufficient if for all users \( u \in \text{Users} \), we have that \( \text{rolesOf} u \) is sufficient.

3.2 Role Compatibility

For systems with a single application, sufficiency can be checked by a dataflow analysis [24]. In general though, each application in a system comes with its own notion of local roles, and a call in application \( A_1 \) to service \( S \) in application \( A_2 \) only provides information about roles in \( A_1 \) held at the call point, not the roles in application \( A_2 \). Thus, in order to check sufficiency, we must somehow “convert” the local roles in each application to a global set of roles. We introduce global role schemas to do this.
Global Role Schema. Let $\text{Sys}$ be a system. A global role schema $\text{Grs} = (R, G)$ consists of a set $R$ of global role names and a mapping $G : R \rightarrow 2^{\text{AllRoles}}$ that maps global role names to sets of local roles in the system $\text{Sys}$. This schema guides role assignments for individual users: if a user is assigned to a global role $g \in R$, then that user must also be assigned all local roles in the set $G.g$.

We can also take an existing set of user assignments and see whether it corresponds to our global role schema. We say that the assignment of roles $\{A.\text{Perm} \mid A \in \text{Sys}\}$ conforms to a global role schema $\text{Grs}$ if there exists a user to global role assignment $\text{Perm} : \text{Users} \rightarrow 2^R$ such that for all users $u$

$$\bigcup_{g \in \text{Perm}, u} G.g = \text{rolesOf}.u$$

That is, there is a mapping of users to global roles such that the set of local roles designated by the global role schema to each user $u$ is exactly the set of local roles $\text{rolesOf}.u$ assigned to the user.

Sufficiency. A global role schema $\text{Grs}$ is fully sufficient if, for each global role $g \in R$, the set of local roles $G.g$ is sufficient. Given a user-role assignment that conforms to a fully sufficient role schema, any service call by an arbitrary user will either be rejected or executed without authentication failure.

Separation. A global role schema $\text{Grs} = (R, G)$ has role separation if no two roles from the same application map to the same global role, that is, for all $g \in R$ and $A \in \text{Sys}$, we have $|G.g \cap A.\text{Roles}| \leq 1$.

Role separation ensures that the roles of each application can be assigned to users independently. If multiple roles of an application appear in the same global role, these roles are effectively combined, potentially violating the intent of the original roles (e.g., allowing users access to data they should not see).

Minimality. Minimality encodes the requirement that a global role schema should not grant access to more services than necessary to ensure sufficiency. A set of local roles $R$ is minimal if it is sufficient and there exists an $l \in R$ such that any subset of $R$ containing $l$ is not sufficient. We extend minimality to global role schemas as follows: a global role schema is minimal if there exists an injective mapping $\mu : R \rightarrow \text{AllRoles}$ from global roles to local roles $\text{AllRoles}$ such that (a) for all $g \in R$ we have $\mu.g \in G.g$, and (b) any subset of $G.g$ containing $\mu.g$ is not sufficient. These conditions ensure that each global role $g$ has unique local role which requires the local role set $G.g$ for sufficiency. Note that there may be more than one minimal global role schema.

3.3 Global Schema Inference

The global schema inference problem (GSI) takes as an input a system $\text{Sys}$ and asks if there is a minimal global schema $\text{Grs}$ which has separation and is fully sufficient. We omit the details of the following theorem (see [12]).

Theorem 1 GSI is NP-complete.
Given a global schema and a witness for minimality, one can check the properties in polynomial time. The hardness is by a reduction from one-in-three 3SAT. One-in-three 3SAT is a variant of 3SAT which determines whether, for a list of three-literal causes, there exists an assignment to the referenced boolean variables such that every clause contains exactly one true literal.

**Proof idea.** Given an instance of one-in-three 3SAT with \( N \) clauses, we first create an application \( A_L \) which defines a special local role \( L \) and contains a single service \( S_L \) that is accessible to role \( L \). For each distinct boolean variable \( v \), we create an application \( A_v \) with local roles \( v^+ \) and \( v^- \), corresponding to the literals \( v \) and \( \neg v \), respectively. Each of these applications contains three services: \( S^+_v \), accessible to role \( v^+ \), \( S^-_v \), accessible to role \( v^- \), and \( S_v \), accessible to both roles. Service \( S_v \) is called by service \( S_L \). These calls represent the constraint that each variable is either true or false.

For each clause \( i \), we also define an application \( A_i \) with three local roles and four services. The local roles are created using the following naming convention:

- If the clause \( i \) contains the positive literal \( v \), we define a local role \( v^+_i \).
- If the clause \( i \) contains the negative literal \( \neg v \), we define a local role \( v^-_i \).

The first service is named \( S_i \), is accessible to all three local roles, and is called by the service \( S_L \). These calls represent the constraint that only one literal is true in each clause. The other three services correspond to the local roles as follows:

- If the local role \( v^+_i \) is defined, then a service \( S^+_v \) is created and protected by this local role.
- If the local role \( v^-_i \) is defined, then a service \( S^-_v \) is created and protected by this local role.

Finally, each service \( S^+_v \) calls any services \( S^+_v \) defined above and each service \( S^-_v \) calls any services \( S^-_v \) above.

If we solve the global role schema for a group containing global role \( L \), we obtain a set of local roles that includes one of \( v^+ \) or \( v^- \) for each boolean variable \( v \) in the original one-in-three 3SAT problem. If we assign \( \text{true} \) to those variables for which role \( v^+ \) is in the group and \( \text{false} \) to those variables for which role \( v^- \) is in the group, we obtain a solution to the one-in-three 3SAT problem. If no global role schema is found, then no solution exists to the satisfiability problem either.

### 4 Constraint-Based Inference

We solve the global schema inference problem through Boolean constraint solving. First, notice that, due to our minimality requirement, the number of global roles is at most the total number of roles in \( \text{AllRoles} \). Although each local role can be in one or more global roles, each global role must be sufficient for at least one local role. If the number of global roles is larger than the number of local roles \( \text{AllRoles} \), then global roles can be eliminated while still ensuring a sufficient global role for each local role.

We generate a set of global roles that include a local role in the following way. Fix a global role \( g \). For each local role \( r \in \text{AllRoles} \), we define an atomic predicate \( r^g \) which is \( \text{true} \) if the role \( r \) is included in the global role \( g \) and \( \text{false} \) otherwise. The predicates \( r^g \) satisfy the following constraints.

1. **[Separation Constraints]** No two local roles from the same application should be mapped to the same global role. That is, for each \( A \in \text{Sys} \), at most one local role \( r \in A.\text{Roles} \) can be in \( g \). Thus, for each application \( A \in \text{Sys} \), we have (considering each \( r^g \) to be a 0-1 variable) \( \sum_{r \in A.\text{Roles}} r^g \leq 1 \), or equivalently,

\[
\bigwedge_{A \in \text{Sys}} \bigwedge_{r_1, r_2 \in A.\text{Roles}, r_1 \neq r_2} (-r^g_1 \vee -r^g_2)
\]

2. **[Sufficiency Constraints]** The sufficiency constraints dictate that for each service \( S \) and each service \( c \in S.C \) called from \( S \), if one of the roles in \( S.M.c \) is mapped to the global roles \( g \), then one of the roles in \( Svc.c.R \) must also be mapped to \( g \). That is,

\[
\bigwedge_{A \in \text{Sys}} \bigwedge_{S \in A.\text{Services}} \bigwedge_{r \in S.M.c} (\bigvee_{r^g} r^g) \rightarrow (\bigvee_{r \in Svc.c.R} \hat{r}^g) \quad (1)
\]
Let $\phi_{\text{Sys}}$ be the conjunction of the constraints from Equation 1 and Equation 1. Clearly, $\phi_{\text{Sys}}$ is polynomial in the size of $\text{Sys}$. A satisfying assignment for $\phi$ is a function mapping each $r^g$ to true or false such that $\phi$ evaluates to true.

**Theorem 2** Let $\rho$ be a satisfying assignment to $\phi_{\text{Sys}}$. Then the set of roles $\{r \mid \rho.r^g = \text{true}\}$ is a global role which is fully sufficient and has role separation.

Given the constraints, we can find a global group containing local role $r$ by conjoin $\phi_{\text{Sys}}$ with $r^g$. To construct a global role schema $G$, we iterate through the set of local roles $\text{AllRoles}$, finding a global role group for each local role. This is done in Algorithm $\text{solve_full}$.

The function $\text{SAT}(\phi)$ returns a set of local roles which are assigned true in a satisfying assignment for the constraint $\phi$. The resulting set of roles is then passed to $\text{minimize}$ (described below), which removes any local roles not required for full sufficiency, while keeping $r$ in the group. The roles from the resulting minimized group are then removed from the workset $\mathcal{R}$, and another role is selected for solving. When $\mathcal{R}$ is empty, all the necessary global groups has been created.

If there is no satisfying assignment to $\phi_{\text{Sys}} \land r^g$, where $r$ is one of the local roles in $\mathcal{R}$, then $\text{SAT}(\phi_{\text{Sys}} \land r^g)$ will return $\emptyset$ and $\text{solve_full}$ will stop with no_solution.

The $\text{minimize}$ function is called with the role group $\mathcal{R}_g$, computed from the boolean constraints in $\text{solve_full}$, and $\mathcal{R}_{\text{req}}$, a subset of $\mathcal{R}_g$ roles which must be present in the final minimized group $\mathcal{R}_{\text{min}}$. Three sets are maintained: $\mathcal{R}_{\text{min}}$ is the minimized group, initialized to $\mathcal{R}_{\text{req}}$, $\mathcal{R}_{\text{new}}$ contains the roles added by the previous iteration, initialized to $\mathcal{R}_g$, and $\mathcal{S}_{\text{min}}$ contains the services accessible, given the set of roles $\mathcal{R}_{\text{min}}$. For each iteration, the services directly callable from $\mathcal{S}_{\text{new}}$ (the services added the previous iteration) are added to $\mathcal{S}_{\text{min}}$. Then, any roles needed to make these callable services accessible are added to $\mathcal{R}_{\text{min}}$. These roles are selected from the role group $\mathcal{R}_g$ by taking the intersection between $\mathcal{R}_g$ and each service’s role set. At the fixpoint, $\mathcal{R}_g$ is both fully sufficient and minimal.

![Figure 5: Example requiring relaxed sufficiency](image)

**Theorem 3** If $\text{solve_full}$ returns a global role schema $G$ for $\text{Sys}$, then $G$ has role separation, is fully sufficient, is minimal with respect to the role signatures of $\text{Sys}$, and each local role appears in at least one global role. If $\text{solve_full}$ terminates with no_solution for $\text{Sys}$, then no such global role schema exists for $\text{Sys}$.

**Solving ascribed roles.** The administrator can specify a subset $R'$ of local roles such that there must be a global role $g$ with $R' \subseteq G.g$. The above algorithm does not address these role ascriptions. To extend $\text{solve_full}$ for ascribed roles, we define the following constraints.

- **[Ascription Constraints]** For each ascription $\{r_1, \ldots, r_k\}$, we have $r_1^g \iff r_2^g \iff \ldots \iff r_k^g$.

We first solve for each of the ascribed roles, conjoining the associated ascription constraint with $\phi_{\text{Sys}}$. $\text{minimize}$ is called for ascribed groups with $\mathcal{R}_{\text{req}} = \{r_1, \ldots, r_k\}$, keeping the ascribed roles in the minimized group. After solutions are found for each ascribed group, we then solve for the remaining roles without any ascription constraints.

Note that we permit ascribed groups to be extended as needed to achieve sufficiency. Due to minimize, we will not unnecessarily add roles. If all local roles are ascribed, then the problem is reduced to global schema checking, rather than global schema inference.

### 4.1 Relaxing Full Sufficiency

Requiring that $G.g$ is sufficient for each global role $g$ may be too strict for some situations. For exam-
we model this scenario using all the roles included in the associated global group. If a user needs access to a system, the administrator picks a local role for that system and then assigns to the user. To avoid indirect authorization errors, when a user has role r_b, then service S_3 is accessible. Since S_3 calls S_4, which requires role r_c, the global role must also contain r_c. But this violates separation.

In many situations, we only require a relaxed version of sufficiency. For example, in a “bottom-up” approach to role assignment, the systems are administered based on local roles and global roles are used to ensure interoperability. Each local role has an associated global role containing any remote roles needed to avoid indirect authorization errors. When a user needs access to a system, the administrator picks a local role for that system and then assigns to the user all the roles included in the associated global group. We model this scenario using subset sufficiency.

We say that global role g is subset sufficient for local role set R_s if R_s ⊆ G.g and R_s is sufficient. In this case, users requiring local roles in R_s can be assigned group g. With this role assignment, all direct calls to services accessible to R_s will not have authorization errors. Of course, any direct call to a service not accessible to R_s (but accessible to G.g) is not guaranteed to execute without authentication failure. Thus, instead of looking for fully sufficient solutions, we can look for global role schema such that for each user u, there is some set of global roles that is subset sufficient for rolesOf_u. Using subset sufficiency, and assuming the role sets \{r_a\}, \{r_b\}, and \{r_c\} for users, the above system has the solution: G_1 = \{r_a, r_b\} (subset sufficient for \{r_a\}), G_2 = \{r_b, r_c\} (subset sufficient for \{r_b\}), and G_3 = \{r_c\} (subset sufficient for \{r_c\}).

We adjust our definition of minimality to account for subset sufficiency. A set of local roles R is subset minimal if it is subset sufficient for a R_s ⊆ R and any subset R' ⊂ R where R_s ⊆ R' is not subset sufficient for R_s. We extend this definition to global role schemas as follows: a global role schema is subset minimal if there exists an injective mapping \(\mu : R \rightarrow \text{AllRoles}\) from global roles to local roles AllRoles such that (a) for all g ∈ R we have \(\mu.g \in G.g\), and (b) any R' such that \(\mu.g \in R'\) and \(R' \subset G.g\) is not subset sufficient for \(\{\mu.g\}\).

To infer a global role schema that only has subset sufficiency, we must adjust the boolean constraint \(\phi_{\text{sys}}\). For each service S in the system, we introduce an atomic predicate \(S^g\) which is true if S is transi-

---

**Algorithm 1** solve_full and minimize

```plaintext
function solve_full
input System Sys
\( R \leftarrow \{ r \mid r \in A, A \in \text{Sys} \} \)
\( \phi_{\text{sys}} \leftarrow \text{constraint_pred(Sys)} \)
G \leftarrow \text{Map.empty}
while \( R \neq \emptyset \) do
    r \leftarrow \text{choose}(R) \{ \text{Pick a role to solve} \}
    g \leftarrow \text{make_name}(r) \{ \text{Name the group for r} \}
    \( R_g \leftarrow \text{SAT}(\phi_{\text{sys}} \land r^g) \)
    if \( R_g \neq \emptyset \) then
        \( G.g \leftarrow \text{make_name}(\text{Sys}, R_g, \{r\}) \)
        \( R \leftarrow R \setminus G.g \)
    else
        return \text{no_solution}
    end if
end while
return G
```

---

function minimize
input System Sys, Role group R_g, Required Roles R_{req}
\( R_{\text{min}} \leftarrow R_{\text{req}} \)
\( R_{\text{new}} \leftarrow R_{\text{req}} \)
\( S_{\text{min}} \leftarrow \text{accessible}(R_g) \)
repeat
    \( S_{\text{new}} = \text{callable}(R_{\text{new}}) \setminus S_{\text{min}} \)
    \( R_{\text{new}} = \text{roles_for}(S_{\text{new}}, R_g) \)
    \( R_{\text{min}} = R_{\text{min}} \cup R_{\text{new}} \)
    \( S_{\text{min}} = S_{\text{min}} \cup S_{\text{new}} \)
until \( R_{\text{new}} = \emptyset \)
return \( R_{\text{min}} \)
tively callable from a service accessible from a group’s required roles. The sufficiency constraints are modified in the following way:

2’ [Subset Sufficiency Constraints] For each called service $c$ in the signature of a service $s$, we add:

$$(S^g \land \bigvee_{r \in S.M.c} r^g) \rightarrow c$$

Additionally, to ensure that callable services are accessible, for each service $S$, we add $S^g \rightarrow \bigvee_{r \in S.R} r^g$. To find a group for a specific role $r$, we call SAT with $\phi_{S,S} \land r \land \bigwedge_{S.I \in S.R} S^g$. We then minimize the result, using a modified version of minimize, which only adds those services callable from the services added in the previous iteration, rather than all services accessible to the newly added roles.

5 Services with Information Flow

We now extend our results to services and systems where we model flow of sensitive data between applications. We must now ensure that information that can only be accessed under some role constraints is not “disclosed” to applications that do not hold the required roles.

To model information flow, we extend the services to include a directed information flow graph. Thus, a service is now a 5-tuple $(n, R, C, M, I)$, where $(n, R, C, M)$ are as before, and $I \subseteq (C \cup \{n, \text{Caller}_{in}\} \times (C \cup \{n, \text{Caller}_{out}\})$ is a set of pairs of service names (or the special symbols $\text{Caller}_{in}$ and $\text{Caller}_{out}$ denoting the entry and exit points respectively of a caller of the service). A pair $(n_1, n_2) \in I$ represents an information flow from $n_1$ to $n_2$, which may occur when the service is called, and self-links of the form $(c, c)$ are not permitted. The information flow graph models two forms of information flow: synchronization, where data from one service is saved in another service, and disclosure, where data from a service is made available to callers of a (potentially different) service. Given a service $S$ and callee $c \in S.C$, the callee-to-self link $(c, S.n)$, represents a synchronization of data from service $c$ to $S$. The callee-to-called link $(c, \text{Caller}_{out})$ represents a disclosure of data from $c$ by $S$.

Synchronization and disclosure are distinguished from benign non-disclosing transfers, where a service moves data between two other services without saving or disclosing it. Callee-to-callee and caller-to-callee links, where there is no additional link from the source to the current service or its caller, are all non-disclosing. For example, if the information flow graph for service $S$ contains $(c_1, c_2) \in S.I$, where $c_1, c_2 \in S.C$, and there are no links from $c_1$ or $c_2$ to $S.n$ in $S.I$, then $S$ facilitates an information flow from $c_1$ to $c_2$ without disclosure.

5.1 Operational semantics

To extend the dynamic semantics of services to include information flow effects, we extend the grammar of expressions as follows:

\[ e ::= \ldots \mid \text{Send}(n, u, e) \mid \text{Recv}(n, u, e) \mid \text{Request}(n, u) \mid \text{Reply}(n, e) \mid \text{Save}(n, e) \mid \text{Data } n \mid \text{NoData} \]

where $n$, $u$, and $n_d$ range over service names and $u$ ranges over users. The source of an information flow is represented by an expression of the form $\text{Data } n$ where $n$ is the name of the originating service. This expression may be passed between services until a call to $\text{Save}$ is made, creating a synchronization, or a call to $\text{Reply}$ is made, creating a disclosure.

Figure 6 lists the inference rules which define our operational semantics with information flow. Each step of the $\rightarrow$ relation may now include an optional information flow effect, which is written above the arrow, if present. We write $e \xrightarrow{n_1 \text{Send}} e'$ to indicate that the expression $e$ steps to expression $e'$ and as a side-effect, data from service $n_1$ is saved. We write $e \xrightarrow{n_1 \text{S}} e'$ to indicate that expression $e$ steps to expression $e'$ and service $n_d$ discloses information from service $n_1$.

A service invocation may take the form of an $\text{Call}$, $\text{Send}$, or $\text{Request}$ expression. As before, $\text{Call}$ does not assume any information flow between the caller and callee. A $\text{Send}$ expression represents the flow of information from the caller to the callee. The source
Figure 6: Semantics for service calls with information flow
of the flow is represented by the third parameter of the \texttt{Send}, which is an expression that should eventually step to a \texttt{Data} expression (by rule \texttt{E-SENDRED}). For direct user invocations of \texttt{Send}, we use the current service’s name as the data source (e.g., \texttt{Send}(n, u, n)).

A \texttt{Request} expression represents an information flow from the callee to the caller. This expression should eventually step to a \texttt{Data} expression. In the event that the callee does not have a corresponding information flow to its caller, a special \noData expression is returned instead (rule \texttt{E-REQNOData}).

Invocations of \texttt{Call}, \texttt{Send}, and \texttt{Request} all require an authorization check before evaluation of the service is performed. If these checks fail, evaluation stops with \texttt{AuthFail} (rules \texttt{E-CALLFAIL}, \texttt{E-SENDFAIL}, and \texttt{E-REQFAIL}).

If authorization is successful, \texttt{Call} steps to \texttt{Eval} (rule \texttt{E-CALLEVAL}). An \texttt{Eval} expression may step immediately to \texttt{Done} (rule \texttt{E-EVALDONE}), duplicate itself (rule \texttt{E-EVALSEQ}), or call other services (rules \texttt{E-EVALCALL}, \texttt{E-EVALSEND}, \texttt{E-EVALMOVE}, and \texttt{E-EVALSAVE}). The form of service invocation depends on the information flow graph and whether a given call is permitted for the user (based on the callee map \( M \)). If more than one invocation is possible, then one is chosen nondeterministically.

A \texttt{Send} expression steps to \texttt{Recv} upon successful authorization (rule \texttt{E-SENDRECV}). A \texttt{Recv} expression may step immediately to \texttt{Done} (rule \texttt{E-RECVDONE}), duplicate itself (rule \texttt{E-RECVSEQ}), or step to \texttt{Eval} (rule \texttt{E-RECVEVAL}). If the information graph contains a caller-to-callee link, then the \texttt{Recv} may step to a \texttt{Save} of the incoming data (rule \texttt{E-RECVSAVE}). If the information graph contains a caller-to-callee link, then the \texttt{Recv} may step to a \texttt{Send} of the incoming data to the associated callee (rule \texttt{E-RECVSEND}).

If a \texttt{Send} occurs within a service invocation, the associated data source expression is first reduced (rule \texttt{E-SENDRED}). If this steps to a \noData expression, the \texttt{Send} steps directly to \texttt{Done} without invoking the target service (rule \texttt{E-SENDNOData}).

A \texttt{Request} expression, upon successful authorization, steps to an \texttt{Eval} followed by a \texttt{Reply} (rules \texttt{E-REQDATA} and \texttt{E-REQREQ}), assuming the service has an information flow link terminating at the caller. If no such link is present, the \texttt{Request} steps to \noData (rule \texttt{E-REQNOData}).

Information flow effects are represented using the \texttt{Save} and \texttt{Reply} expressions. First, the data source parameter must be reduced to a \texttt{Data} expression by rules \texttt{E-SAVERED} and \texttt{E-REPLYRED}. Then, \texttt{Save} reduces to \texttt{Done} and \texttt{Reply} reduces to \texttt{Data}, emitting an information flow effect — either a synchronization from the data source to the current service (\texttt{Save}, via rule \texttt{E-SAVEDONE}) or a disclosure of the data source by the current service (\texttt{Reply}, via rule \texttt{E-REPLYDISC}). If the data source expression reduces to \noData, then the enclosing \texttt{Save} or \texttt{Reply} reduces with no information flow effect (rules \texttt{E-SAVENOData} and \texttt{E-REPLYNOData}).

We write \( \rightarrow^* \) to represent the transitive closure of the \( \rightarrow \) relation. A direct service invocation is an expression of the forms \texttt{Call}(n, u), \texttt{Send}(n, u, n), or \texttt{Request}(N, u) representing the direct call of a service by a user.

Proposition 2 (Evaluation) Let \( Sys \) be a well-formed system, \( n \) a service in \( Sys \), and \( u \in Users \) a user. The evaluation via \( \rightarrow^* \) of a direct service invocation of the forms \texttt{Call}(n, u) or \texttt{Send}(n, u, Data n) either diverges or eventually terminates with \texttt{Done} or \texttt{AuthFail}. The evaluation via \( \rightarrow^* \) of a direct service invocation \texttt{Request}(n, u) either diverges or terminates with \texttt{Done}, \texttt{AuthFail}, \texttt{Data n}, or \noData.

5.2 Sufficiency

We now extend our definition of sufficiency to include \texttt{Send} and \texttt{Request} service invocations. We say that a direct service invocation \texttt{Call}(n, u), \texttt{Send}(n, u, Data n), or \texttt{Request}(n, u) is accepted if \( Svc.n \in A_i \) for some \( A_i \in Sys \) and \( Svc.n.R \cap A_i.\text{Perm}.u \neq \emptyset \), i.e., if it does not evaluate in one step to \texttt{AuthFail}.

We say that a set of roles \( R \) is sufficient for \( Sys \) if, for every user \( u \) with \( \text{rolesOf}.u = R \), any direct service invocation in \( Sys \) by user \( u \) that is accepted does not evaluate, via \( \rightarrow \), to \texttt{AuthFail}.
5.3 Non-disclosing Global Schema

Informally, we say that a global role schema $G_r$s is non-disclosing for a conforming user assignment, if it does not permit the disclosure to a user $u \in Users$ of data originating at a service $S$ for which the user does not have access. This is the requirement that a user cannot subvert access control rules by exploiting information flow between services. To state this precisely with respect to our operational semantics, we define the following predicates (where $n, n' \in Names$ are service names and $u \in Users$ is a user).

The predicate $Disclose(n, n')$ is true if there exists a direct service invocation which, when evaluated via $\rightarrow^+$, emits the information flow $n \rightarrow n'$. The predicate $Sync(n, n', u)$ is true if there exists a direct service invocation which, when evaluated via $\rightarrow^+$, emits the information flow synchronization $n \rightarrow n'$. Finally, $Flow$ is the reflexive and transitive closure of the union of $Disclose$ and $Sync$: $Flow = (Disclose \cup Sync)^*$. If $Flow(n, n')$, then there exists a sequence of direct service invocations which will result in the disclosure of data from $n$ at service $n'$.

A global role schema $G_r$s is non-disclosing if there does not exist services $S$ and $S'$, a global role $g$, and a user $u$ with $rolesOf.u = G.g$ such that (a) role $g$ does not have access to service $S$: $G.g \cap S.R = \emptyset$, (b) role $g$ has access to service $S'$: $G.g \cap S'.R \neq \emptyset$, and (c) $Flow(S.n, S'.n)$ is true.

A global information flow (GIF) graph $I_g$ for a system $Sys$ is a directed graph constructed from the local information flow graph of each service. For each service $S$, the GIF graph has the set of nodes $S.C \cup \{S.n, S.Caller_in, S.Caller_out\}$, consisting of a node for each service called by $S$, a node $S.n$ for the service $S$ itself. The set of all nodes in $I_g$ is the disjoint union of the set of nodes for each service. To distinguish a node $v$ from service $S$, we write $S.v$. For a service $S$, we create an edge $(S.v_1, S.v_2)$ if $(v_1, v_2) \in S.I$. For different services $S$ and $S'$ (with names $n$ and $n'$), we create the following additional edges:

- $S$ sends to $S'$: there is a link $(S.n, S'.Caller_in)$ if $(n, n') \in S.I$ (service $S$ sends data to $S'$) and $(Caller_in, v') \in S'.I$ for some $v'$.
- $S'$ requests from $S$: there is a link $(S.Caller_out, S'.v)$ if $(n, v) \in S'.I$ and there is a $v'$ with $(v', Caller_out) \in S.I$.

We can now define a static version of (dynamic) information flow, based on the global information flow graph: $StatFlow(S.v, S'.v')$ is true if there exists a path in $I_g$ from $S.v$ to $S'.Caller_out$. This function is an over-approximation of $Flow$: $Flow(n, n')$ implies $StatFlow(n, n')$, but it is possible to have a path in the global information flow graph that is not feasible due to authorization errors. However, there is no loss in precision if the role schema is sufficient.

**Theorem 4** [Disclosure] Given a sufficient global schema $G_r$s for system $Sys$, for any two services $S$ and $S'$ in $Sys$, $Flow(S.n, S'.n)$ iff $StatFlow(S.n, S'.n)$.

To compute global role schemas that are non-disclosing, we conjoin additional constraints with $\phi_{sys}$ to ensure only permitted information flow. For each pair of services $S, S'$ such that $StatFlow(S.n, S'.n)$ is true, we add the constraint:

$$\bigwedge_{r \in R} r^g \rightarrow (\bigvee_{r' \in R'} r'^g).$$

We now modify the function $constraint_{pred}$, called from $solve_{full}$ in algorithm 1, to include these extra information flow constraints. The resulting version of $solve_{full}$ will infer non-disclosing global role schemas.

**Theorem 5** If the modified $solve_{full}$ returns a global role schema $G_r$s for $Sys$, then $G_r$s has role separation, is non-disclosing, is fully sufficient, and is minimal with respect to the role signatures of $Sys$. If $solve_{full}$ terminates with no solution for $Sys$, then no such global role schema exists for $Sys$.

### 6 Experiences

We have implemented RoleMatcher, a tool to infer global role schemas. Our tool takes as input a textual representation of the $Sys$ definition described in section 3. It produces a global role schema via Algorithm 1, using the MiniSat [10] satisfiability solver to resolve the boolean constraints. Both fully sufficient and subset sufficient solutions may be obtained.
Figure 7: Performance results for RoleMatcher

<table>
<thead>
<tr>
<th>System</th>
<th>Num svcs</th>
<th>Num calls</th>
<th>Num grps</th>
<th>Max pred</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>portal1</td>
<td>8</td>
<td>5</td>
<td>N/A</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td>portal2</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>92</td>
<td>5</td>
</tr>
<tr>
<td>data sync</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>69</td>
<td>5</td>
</tr>
<tr>
<td>it mgt</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>94</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 7 summarizes the results of running our tool on several small examples, using a Dell PowerEdge 1800 with two 3.6Ghz Xeon processors and 5 GB of memory. The “Num svcs” and “Num calls” columns represent the total number of services in the system description and the total number of service calls, respectively. “Num grps” lists the number of groups in the inferred schema (or N/A if no solution was possible). “Max pred” is the size of the largest predicate passed to the solver and “Time” the elapsed time in milliseconds. portal1 and portal2 correspond to the clinic web portal of Figure 2(a), data sync corresponds to the data synchronization example of Figure 2(b), and it mgt represents the case study described below. Since the problem is NP-complete, the use of an exponential procedure is inevitable. Even though the algorithm involves SAT solving, this has not been a bottleneck. This is because the constraints are a combination of 2-literal clauses (for separation) and Horn clauses (for sufficiency), and Boolean constraint propagation and unit resolution heuristics in a modern SAT solver are particularly tuned for these types of clauses.

Case Study. To further evaluate our approach to role interoperability, we considered a real-world scenario described to us by an industrial colleague. An IT management applications company had several independently-developed products from companies it had acquired. The company wished to integrate these applications (including their security models) in order for customers to use them as an end-to-end solution to their IT management needs.

Figure 8 shows a (simplified) view of how two such applications might interact. The IT System Management (ITSM) application includes modules for incident, problem, and change management, as well as a database to track a company’s hardware and software assets. The Patch Management application gathers an inventory of the patches currently installed on the company’s computers and manages the application of new patches. The integrations between these systems are straightforward: the patch inventory data should be included in the ITSM asset database (the arrow from Discovery Data to Asset Management) and the application of patches should be controlled via the ITSM change management module. (the arrow from Change Management to Apply Patch).

Both systems use role-based access control, but with very different role models. The Patch Management application has a very simple model with three fixed roles: User, PowerUser, and Admin. The ITSM application allows system administrators at the customer to define their own roles and mappings to data/service access permissions. The roles shown in Figure 8 for the ITSM application are representative of a typical customer configuration. Thus, it is not feasible for the application vendor to ship a fixed role mapping. A better solution is to build a tool to extract role interfaces from the ITSM system’s role metadata and infer a global role schema as a part of application deployment and configuration.

The role schema for a simple system definition, like the one in Figure 8, can be computed by hand. A quick inspection shows that the Asset and Change ITSM roles should be mapped to either the Admin or PowerUser patch management roles. However, a real system is more complicated. Customers can add new integration points between the two applications (e.g. the dotted line from Asset Management to Discovery Reports). Also, the asset database is likely to contain data from several discovery applications. Thus, we believe that such customers would benefit from the use of a global role inference tool.

7 Related work

Access control for web services. The eXtensible Access Control Markup Language (XACML) [1] defines an access control policy language for web services which is flexible enough to express many access control models, including RBAC [2]. However,
XACML policy definition and enforcement are not tied to the underlying access policies of individual services. Thus, while clearly a useful tool for defining security policies, XACML does not address the basic issues addressed by global role schema inference. XACML policies could be generated from a global role schema. This would enable centralized enforcement while avoiding the problems associated with two independent policy layers.

[26] proposes an access policy language which can reference the past history of service invocations, using pure-past linear temporal logic (PPLTL). Like XACML, it uses a centralized approach to policy specification and enforcement. In order to reason about access policies across systems, the administrator must provide a role mapping. Thus, it can be used as a layer on top of global schema inference.

Other approaches to ensuring access control constraints are possible, e.g., in situations where multiple providers for a service are available, a broker can dynamically select among the available providers to satisfy security requirements [6].

Other standards address orthogonal issues to access control: Security Assertions Markup Language (SAML) provides a framework for querying authentication and authorization statements across security domains, WS-Security defines how encryption and digital signatures may be applied to web service messages, and WS-Policy establishes a format for services to advertise their security requirements.

Theory of access control policy interoperation. [13] investigates the theory behind interoperability of ACL (Access Control List) based security models. The desired interoperation between systems is specified as a set of accessibility links between principals of the individual systems and a set of restrictions between principals. It is shown that finding a maximal subset of the accessibility links, such that the security constraints of the individual systems are not violated, is NP-Complete. [5] shows a similar result for the interoperation of partial order based security models.

The Ismene policy language [18] uses a predicate-based notation for specifying general security policies. When a collection of entities wish to interact in a distributed system, their individual policies are reconciled to create a policy instance which governs the current session. Reconciliation involves finding a satisfying assignment to the conjunction of the individual policies. The access control policies implied by role interfaces could be represented in Ismene. However, the reconciliation of Ismene policies occurs at runtime, and the transitive nature of role composition makes runtime evaluation an impractical approach.

A related problem is decentralized trust management [4], where credentials from independent systems are combined according to a well-defined policy. Trust management has been implemented using a role-based model in the policy language RT [16, 17].

Relation to standard RBAC models. [25] defines three RBAC models, which form the basis of American National Standard 359-2004 for RBAC.
RBAC\(_0\) models users, roles, permissions and sessions. The model described in this paper captures all of RBAC\(_0\), except for sessions. A \textit{session} is a map from a user to a subset of their roles and captures the idea that user need not activate all of their roles for a given task. Since the roles of a session are at the user’s discretion, they are not relevant to the computation of global role schemas. However, if a session does not activate all of a users roles, indirect authorization errors can still occur. Thus, the role schema may be a useful guide to the user on which roles need to be activated to accomplish a given task.

RBAC\(_1\) models a hierarchy of roles, where granting a role gives a user all the permissions associated with any role dominated by the granted role. While we do not currently include role hierarchies, it is straightforward to represent such a hierarchy in our boolean constraints and to extend the inference algorithm to find the \textit{lowest} role in a hierarchy which satisfies a given constraint, keeping with the principle of least privilege. This is left as future work.

RBAC\(_2\) extends RBAC\(_0\) by adding constraints on the sets of roles assigned to users. These constraints include \textit{mutual exclusion} constraints (e.g., a user may not be assigned roles A and B together), \textit{prerequisite roles} (e.g., to have role B, a user must also be assigned role A), and \textit{cardinality constraints} (e.g., a user may be assigned at most one role). Clearly, a global role schema should not violate any role assignment constraints. Our separation constraints are just a form of mutual exclusion constraint. It is trivial to extend our model to represent arbitrary boolean role constraints, including mutually exclusive and prerequisite roles. Cardinality constraints are less critical to global role schemas, as they mainly concern user-role and role-permission associations, rather than the relations between roles.

\textbf{Role mapping.} A role-based access control policy may be seen as an abstraction of an underlying ACL security policy. \textit{Role mapping} [27, 28] attempts to find this abstraction automatically by finding a minimal set of roles for a single system which captures the underlying relationship between users and the resources they may access. [27] shows that this problem is NP-Complete and provides algorithms for both full and approximate solutions. Alternatively, roles may be constructed using data mining techniques which attempt to infer the functional responsibilities of users, based on patterns in an organization’s access rights [15].

Role mapping may be extended to address the interoperability of RBAC systems. In this context, it is assumed that an inter-system call will include the set of underlying permissions required on the target system. \textit{Inter-domain role mapping} (IDRM) attempts to find the minimal set of target system roles which satisfy the requested permissions [23, 9, 7]. [3] presents a static version of IDRM where roles are directly mapped between systems. Links between roles are determined by grouping similar objects across systems (e.g., accounts, insurance claims, etc.) and then linking their underlying permissions (e.g., access to accounts on system A implies access to accounts on system B). Mappings between roles attempt to satisfy as many of these links as possible while avoiding the subversion of the individual systems’ access policies. This problem is formulated as a system of integer programming constraints.

When inferring a global schema, we use each service interface’s set of called services as a proxy for required access permissions. This is appropriate and necessary for systems whose internals are encapsulated by services. Our approach globally optimizes the role mappings to minimize the number of mappings needed while preserving interoperation.

\textbf{Static analysis of RBAC systems.} [24] describes a static analysis for roles within a single Java Enterprise Edition application. This analysis checks for three types of errors: indirect authorization errors, redundant role definitions, and the subversion of an RBAC policy by exploiting unchecked intra-component calls. Our work can be viewed as extending these static checks across systems.

\textbf{Information flow.} [19] describes a static program analysis to compute information flow in a modular fashion. All variables, arguments, and procedure return values are labeled with a lattice element. Each lattice element represents the set of owners for data flowing into a variable and the set of readers to which the variable may eventually flow. Distributed com-
munication is modeled using channels, which are also labeled with their information flow properties. This work has inspired the use of decentralized information flow security in programming languages [20], operating systems [14], and web service compositions [21]. Rather than use information flow as the access control mechanism, we use information flow to inform a standard access control policy, leveraging the use of existing RBAC infrastructure. Our information flow constraints can be viewed as an instantiation of the standard lattice model [8] by defining a lattice whose elements consist of sets of global roles, where the top element is the set of all global roles and the bottom element is the empty set. Each service is assigned a lattice element corresponding to the set of global roles by which it is accessible. If a service B discloses data from a service A, it must have an element equal to the lattice element computed for A or an element lower in the lattice. Information flow has also been studied in the context of RBAC. [22] computes the information flow for a single system’s RBAC policy due to two causes: 1) the ability to pass data between two objects protected by the same role, and 2) the ability to pass data between objects protected by different roles, when those roles may be simultaneously activated.

8 Discussions

Access control integration for systems interacting via service calls is an important industrial problem. Current solutions (e.g., from Securent, Vaau, and Aveksa) either perform expensive and error-prone manual integration, or ad hoc mining of access control rules from logs that are then centrally managed and enforced. In contrast, our global constraint analysis, together with our precise formulation of the semantics, allows us to make precise claims of correctness with respect to sufficiency and confidentiality.

We are addressing some limitations in current work. In the event that no global role schema satisfying our constraints exists for a given system, we currently just report an error. Going forward, we would like to provide more guidance to the user, e.g., by providing feedback about which services and constraints actually cause the infeasibility.

We can also attempt to relax constraints or drop inter-system calls and return to the user a solution which satisfies as many of the constraints as possible. This approach has been proposed for other access control interoperability situations [13]. For example, upon finding that no full solution is possible, one might selectively relax the role separation constraints until a sufficient global schema is found.

In a large collection of systems, the appropriate mappings between roles may not be obvious. If several candidate roles can satisfy the constraints for a given group, the system arbitrary picks one of them. The administrator may influence this process by using role ascriptions. In fact, our approach permits the administrator to pick any point on the spectrum from full role inference to the validation of a completely specified global role schema. However, manually providing ascriptions for all the roles in a large enterprise may be quite tedious. One solution is to use existing role mining techniques [27, 28, 3], to automatically compute an initial set of ascriptions and run our algorithm using this initial guess. If these ascriptions do not admit a global role schema, the solver could relax them and consider other role mappings as needed.

Finally, we are currently scaling our implementation to allow us to apply our algorithm on real industrial case studies from the healthcare space.

References


