A Secure, Constraint-Aware Role-Based Access Control Interoperation Framework

Nathalie Baracaldo, Amirreza Masoumzadeh, and James Joshi
School of Information Sciences
University of Pittsburgh
Email: [nab62@, amirreza@sis., jjoshi@sis.] pitt.edu

Abstract—With the growing needs for and the benefits of sharing resources and information among different organizations, an interoperation framework that automatically integrates policies to facilitate such cross-domain sharing in a secure way is becoming increasingly important. To avoid security breaches, such policies must enforce the policy constraints of the individual domains. Such constraints may include temporal constraints that limit the times when the users can access the resources, and separation of duty (SoD) constraints. Existing interoperation solutions do not address such cross-domain temporal access control and SoDs requirements. In this paper, we propose a role-based framework to facilitate secure interoperation among multiple domains by ensuring the enforcement of temporal and SoD constraints of individual domains. To support interoperation, we do not modify the internal policies, as most of the current approaches do. We present experimental results to demonstrate our proposed framework is effective and easily realizable.

I. INTRODUCTION

Nowadays, sharing computing and information resources between organizations is becoming more and more important. In fact, several businesses base their competitive advantage on being able to share relevant information and resources with their partners. Unfortunately, security concerns are responsible for the reluctance of some organizations to join such environments [6]. In multi-domain environments, it is necessary that each of the domains involved can control the extent to which it wants to share its resources with other domains. It is important that the internal policy of each organization is not violated during interoperation. If such violations occur, vital information might be lost, modified or disclosed through unauthorized accesses. Therefore, when two or more domains want to interoperate, an interoperation policy should be created. In existing approaches (e.g. [13]), an interoperation policy is usually created manually, which is cumbersome and error-prone. It is desirable to create interoperation policies automatically to avoid security breaches introduced by human mistakes while reducing the amount of time required to enable the sharing activities.

A secure interoperation policy follows the principles of autonomy and security [16]. The principle of autonomy states that accesses permitted within a domain before the integration should still be allowed after the integration. On the other hand, the principle of security says that accesses forbidden within a domain before the integration should still be forbidden after the integration. To ensure these, the policy constraints of individual domains need to be enforced when sharing resources with external domains. Among the constraints that should be enforced are the separation of duty constraints [5], least privilege constraints [12] and temporal constraints [9].

Role Based Access Control (RBAC) [4] has been shown to encompass discretionary and mandatory access control models and support organization or user-specific requirements [10]. Because of the benefits offered by RBAC, it is considered as a promising approach for expressing domain policies; in this paper, we assume that all the involved domains employ extended RBAC models. To facilitate cross domain accesses in such a multidomain environment, it is first necessary to establish role to role mappings between domains, so that users from a domain can acquire the desired permissions from other domains[3]. Several algorithms have been proposed to solve the role mapping problem [16], [3], [22], [2], [21], [17]. However, these solutions have three key limitations. The first limitation is the limited capability to handle Static SoD (SSoD) constraints. In [18], a role mapping algorithm that respects SSoD of the internal policy was proposed. However, this approach assumes that the policies of both domains are static and are completely known to each other before the interoperation takes place. This assumption is too strong, as the policies of the domains involved can change at any time and some organizations may not trust their partners enough to reveal their policies. The second limitation is related to handling temporal constraints while creating an interoperation policy. The role mapping algorithm proposed in [14] attempts to minimize the number of roles to be mapped to a given request. However, it does not consider the temporal constraints. As a result, mapped roles may not be available at desired times. The third limitation of existing work consists of requiring that the roles selected by the role mapping algorithm only have authorization for the exact set of required permissions. To achieve this objective, some proposed solutions either modify the policy of the domains [18], [14], [20], [19] or grant extra privileges; however, neither of these two solutions are adequate, as the former increases the policy administration difficulty and the latter violates the principle of least privilege. Furthermore, none address these three issues together.

In this paper, we propose a role based framework that addresses aforementioned limitations of existing approaches. The key contributions of this paper are:

- We propose a secure interoperation framework and formally show that it enforces temporal, SSoD and DSoD constraints of individual domains involved.

- We introduce temporal coverage as the objective function of role mapping to maximize interoperation based on temporal requirements.

- We use the notion of filter roles in the interoperation
policy to ensure the enforcement of the least privilege principle, and support secure interoperation even when the internal policy evolves.

The paper is organized as follows. In Section II, we present an overview of GTRBAC model, hybrid hierarchy and SoD constraints. In Section III, we present the requirements of the system. In Section IV, we present the proposed secure interoperation framework. We show that the framework complies with the requirements in Section V. In Section VI, we present the proposed role mapping algorithm and implementation results. In Section VII, we discuss related work and in Section VIII, we conclude the paper and discuss future directions.

II. BACKGROUND

Temporal Constraints Generalized Temporal Role Based Access Control (GTRBAC) [9] is an extension of RBAC [4] with temporal constraints that restrict the periods of time during which a user can acquire certain permissions. It allows several types of temporal constraints on different components of the model such as roles, role-permission assignment, etc. In this paper, we only consider temporal constraints for enabling and disabling roles. In GTRBAC, a role can be activated by an authorized user only if it is enabled. A disabled role cannot be activated. To express temporal constraints, GTRBAC uses periodic time expressions, which are of form \((\text{begin}, \text{end})\), where \(P\) is a periodic expression denoting an infinite set of periodic time instants, and \([\text{begin}, \text{end}]\) is a time interval denoting the lower and upper bounds for the instants in \(P\). We refer interested readers to [9] for further details. We define \(\text{overlapRatio}(t_{c1}, t_{c2})\) as the ratio of the time instants that the interval \(t_{c1}\) has in common with \(t_{c2}\) to the interval \(t_{c1}\).

GTRBAC also introduces the concept of hybrid hierarchy [9]. Roles \(r_1\) and \(r_2\) can be hierarchically related in one of the following ways. (1) I-senior \((r_1 \geq_I r_2)\) where \(r_1\) inherits the permissions of \(r_2\); (2) A-senior \((r_1 \geq_A r_2)\) where users assigned to \(r_1\) can activate \(r_2\); however, \(r_1\) does not inherit \(r_2\)’s permissions; (3) IA-senior \((r_1 \geq_{IA} r_2)\), in this case \(r_1\) is I-senior and A-senior of \(r_2\).

In order to deal with the semantics of temporal constraints in presence of hybrid hierarchy, Joshi et al. propose two variations of hierarchical relations: weak and strong [8]. A weak hierarchical relation allows a senior role to activate a junior role or acquire its permissions even when the junior role is not enabled. In contrast, a strong hierarchical relation requires that both the senior and the junior roles be enabled for the inheritance semantics to be valid. In this paper, we use the function \(P_{au}(r, t)\) to denote the permissions that can be acquired through a role \(r\), including permissions inherited through hierarchical relations.

Separation of duty constraints Separation of duty constraints (SoD) are used to prevent fraudulent activities within an organization by preventing a unique user from playing two or more conflicting roles. There are two types of SoD constraints. The Static SoD (SSoD) and the Dynamic SoD (DSoD), SSoD restricts the authorization of users to conflicting roles [1], [5], [12], [11]. An SoD constraint is denoted by \((RS, k)\) where \(RS = \{r_1, r_2, r_3, ..., r_n\}\) is a set of roles in conflict and \(2 \leq k \leq n\). We use \(ssod(RS, k)\) to express SSoD constraints, which states that a user can be authorized to at most \(k - 1\) roles in \(RS\). Similarly, a DSoD constraint \(dsod(RS, k)\) states that a user can activate at most \(k - 1\) roles in \(RS\).

III. REQUIREMENTS FOR SECURE INTEROPERATION

An interoperation policy indicates which roles in an external domain will be able to acquire which privileges within a local domain. We propose a framework whose main goal is to establish an interoperation policy that respects the policies of individual domains. Here, by external and internal domains we mean the domain that issues access requests and the domain that grants/denies these requests, respectively. We assume that the domains involved use RBAC policies extended with SoDs, temporal constraints and hybrid hierarchy. We illustrate the requirements through the following example.

Example 1: Consider that the Treasurer Office (TO) wants to interoperate with the County Clerk Office (CCO) to increase the effectiveness of detecting tax evasions and have up-to-date and accurate information [16]. Part of TO’s policy is shown in Figure 1, where each role has been directly assigned to it the permissions shown in the rectangles and the permissions that each role can acquire are shown underneath the policy. TO’s policy has the following roles: Chief auditor \((r_{CA})\), Process auditor \((r_{PA})\), Refund auditor \((r_{RA})\), Tax collector \((r_{TC})\), Fraud manager \((r_{FM})\), Tax assessor \((r_{TA})\), Tax bill approver \((r_{TBA})\) and Treasurer \((r_{TS})\). TO’s policy contains important information on how to manage SoD conflicts. For instance, it contains the SSoD constraint \(ssod((r_{TS}, r_{CA}), 2)\) that specifies that \(r_{TS}\) and \(r_{CA}\) cannot be assigned to the same user; i.e., the person in charge of performing the operations should not be her own auditor. In addition, DSoD constraints specify how to follow the principle of least privilege. The constraint \(dsod((\{r_{EL}, r_{TA}, r_{TBA}\}), 3)\) specifies that the same user cannot activate 2 of those roles simultaneously. Finally, it also contains information on the times during which the roles are enabled; e.g., \(r_{TBA}\) can only be activated during Mondays to Thursdays and role \(r_{TA}\) can only be activated during office hours (7am to 7pm) Monday to Friday. These SSoD, DSoD and temporal constraints within a local domain also need to be enforced when users from external domains are given access through a secure interoperation policy.

In order to comply with the principle of security, all the constraints of the internal domain’s policy should be enforced

![Diagram of role hierarchy and permissions](image-url)
when it interoperates with external domains. Hence, it is important to create an interoperation policy that respects these constraints in local domains. This requires looking for the set of roles in the internal policy that can provide the permissions that need to be granted to users assigned to a role of the external domain without violating any constraint. When an internal domain has a large policy (in terms of the number of roles and constraints), performing this process manually is very tedious and error-prone, and hence may lead to violations of the internal policy. Our framework provides the means to automate this process.

To create an interoperation policy, the internal domain’s security administrator issues an interoperation query that contains a set of desired permissions, and a periodic expression during which those permissions should be available for a particular external role. Given a set of interoperation queries, the framework should construct an interoperation policy that fulfills the following requirements:

1. It should respect the SSoD, DSoD and temporal constraints of the internal domain.
2. The time during which interoperation can happen should be maximized.
3. It should be possible to establish additional temporal restrictions on the accesses to the internal resources.
4. It should provide only the required permissions.
5. Individual domains should not be required to disclose their policies to create the interoperation policy. We assume the external domain does not trust the internal domain enough to reveal its policy and vice versa.
6. The policy of the internal domain should not be modified in order to interoperate, but it may evolve.

IV. POLICY INTEROPERATION FRAMEWORK

Our approach consists of creating an intermediate layer of roles that forms the interoperation policy, which governs how a particular external domain can access the resources of the internal domain; one interoperation policy is created for each external domain and managed by the internal domain.

A. Preliminaries and Notations

We use the following notations. A GTRBAC policy is an 8-tuple \( \langle U, R, P, UA, PA, RH, TC, SSoD, DSoD \rangle \) where in addition to standard GTRBAC components, \( TC \) is a function assigning enabling time constraint to roles. We use subscripts to refer to these sets in a specific GTRBAC policy, e.g., \( SSoD_P \) refers to the set of SSoDs in policy \( P \).

We use \( P \) and \( E \) to denote the policies of internal and external domain, respectively. The interoperation between the two domains is a triple \( \langle Q, P, E \rangle \) where \( Q \) is a set of interoperation queries. An interoperation query is a triple \( \langle r, PS, \tau \rangle \), indicating role \( r \in R_Q \) (the external domain) requires access to the permission set \( PS \subseteq P_P \), during a periodic time \( \tau \).

The interoperation policy is constructed using a special type of roles called filter roles [7]. A filter role is a role that has associated with it an upper bound set (UBS), which is the maximum set of permissions that can be acquired through it. We formally define it as follows.

\[
\text{Candidate Solution} \quad P_{au}(r \in R_{sel}) \quad \text{Enabled time} \quad CR
\]

<table>
<thead>
<tr>
<th>Candidate</th>
<th>( P_{au}(r \in R_{sel}) )</th>
<th>Enabled time</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_{au}(r_1) = { p_1, p_2, p_3, p_4 } )</td>
<td>3pm-8pm</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>( P_{au}(r_2) = { p_3, p_4, p_5 } )</td>
<td>9am-4pm</td>
<td>0.625</td>
</tr>
<tr>
<td>3</td>
<td>( P_{au}(r_3) = { p_1, p_4 } )</td>
<td>8am-10pm</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>( P_{au}(r_4) = { p_5, p_6, p_7 } )</td>
<td>2pm-6pm</td>
<td>0.5</td>
</tr>
</tbody>
</table>

TABLE I: Sample Coverage for \( \{ p_1, p_2, p_3, p_4 \} \) and \( \tau = \{ 9am - 5pm, everyday \} \)

**Definition 1:** Let \( r_f \) be a filter role and UBS \( (r_f) \) indicate its upper bound permission set. The authorized permissions for \( r_f \) at time \( t \) is

\[
P_{au}(r_f, t) = \bigcup_{r \in \{ \text{juniors}(r_f) \}} P_{au}(r, t) \cap \text{UBS}(r_f)
\]

We use dot notation to refer to the elements of a tuple, e.g., \( c.RS \) refers to the role set in the constraint \( c = dso(d(RS, k)) \). Whenever a set is used in a formula where a single element should be normally used, we mean the union of the results of applying each member of the set in the formula. For instance, \( P_{au}(RS) \) refers to the union of sets of permissions authorized for individual roles in set \( RS \) and \( DSoD_P \). \( RS \) refers to the union of sets of roles involved in a DSoD constraint in \( P \).

B. The Role Selection Optimization Problem

To create a suitable interoperation policy, it is necessary to find the sets of roles that provides the desired permissions and that can also be activated during the requested periods of time. We define the notion of coverage ratio to indicate how much of the requested interval is covered by a particular solution \( R_{sel} \subseteq R_P \).

**Definition 2 (Coverage Ratio (CR)):** Given a candidate solution \( R_{sel} \) for the interoperation query \( \langle r, PS, \tau \rangle \), \( CR(R_{sel}, PS, \tau) \) is defined as:

\[
\text{overlapRatio}(\tau, \bigcup_{p \in PS} \{ t | p \in P_{au}(r_i, t) \})
\]

The objective of our role selection process is to choose a solution that maximizes coverage ratio. In addition, it needs to consider the restrictions imposed by SoD constraints. We define the role selection problem as follows.

**Definition 3:** The Role Selection Problem (RSP) for a query \( q = \langle r, PS, \tau \rangle \) is to find a solution, \( S(q) \), for the following optimization problem:

\[
\text{Max}_{R_{sel} \in R_P} \quad CR(R_{sel}, PS, \tau)
\]

\[
\text{s.t.} \forall dso(d(RS_i, k_i) \in DSoD : |R_{sel} \cap RS_i| < k_i \quad \forall ssod(RS_i, k_i) \in SSoD_P : |R_{sel} \cap RS_i| < k_i \quad P_{au}(R_{sel}) \geq PS
\]

If \( S(q) \) has a coverage of zero for a given query \( q \), it means that no set of roles in \( R_P \) can satisfy the query. i.e., the interoperation request is denied.

**Example 2:** Suppose that Table I lists feasible solutions for the role selection problem for query \( \langle r, \{ p_1, p_2, p_3, p_4 \}, 9am - 5pm, everyday \rangle \) and their calculated CR. Solution 2 provides
In this section, we explain and prove that the proposed LAP provides only the required permissions, respects DSODs, and temporal constraints of the internal policy.

**Conforming with the Requirements:**

- **DSOAs on the Constrained Roles to External Users:** In order to allow secure interoperation (Q, P), we augment the interoperation-augmented policy. For interoperation roles that are not strong-seniors of the corresponding internal user, we cannot assume any knowledge of the external's domain policy, and therefore, we consider the worst case scenario, where external users have only the required permissions, respecting the principle of non-disclosure of the external's domain policy, and the following temporal constraints. In the above definition, we consider a unique external user u, in the external domain is assumed to have access to all the external users.

- **Equivalent DSODs on the Interoperation Roles:** In order to allow secure interoperation (Q, P), we augment the interoperation-augmented policy. We only consider the case where each layer represents a different type of filter roles, respectively. However, this solution is not suitable based on temporal requirements as it only covers 50% of the requested interval.

- **Equivalent Filter Roles:** Equivalent DSODs on the interoperation roles to external users. The correspondence is provided by mapping function Ψ. In the external domain is assumed to have access to all the external users.

In the above definition, we consider a unique external user u, in the external domain is assumed to have access to all the external users.

- **Filter Roles:** Equivalent DSODs on the interoperation roles to external users. The correspondence is provided by mapping function Ψ. In the external domain is assumed to have access to all the external users.

In the above definition, we consider a unique external user u, in the external domain is assumed to have access to all the external users.
Fig. 3: Example of an IAP system.

A. Providing Only the Requested Permissions

The permissions an external user \( u \) authorized for \( r \in R_Q \) can acquire in the internal domain are bounded by the set of queries created by the system’s administrator for role \( r_c \). However, \( u \) can acquire permissions in the internal domain by activating the roles in the interoperation policy that are juniors of \( r_c \). The following theorem shows IAP complies with these requirements.

Theorem 1: Let \( Q_e \) be a set of queries with the same external role \( r_e \), for permissions in internal policy \( P \). A user \( u_e \) in \( P \)'s IAP, can only acquire the permissions in \( Q_{e,PS} \) because of being authorized for \( r_e \).

Proof: Being authorized for \( r_e \) allows \( u_e \) to activate \( r_e \) and its A-juniors to acquire permissions. By activating \( r_e, \) \( u_e \) does not acquire any permission because, as per Definition 4, \( r_e \) does not have assigned any permissions. The A-juniors of \( r_e \), roles in \( R_e \), are filter roles that have their USB set to \( q,PS \), where \( q \in Q_e \). Therefore, \( u_e \) can acquire at most \( Q_{e,PS} \).

B. Respecting Internal Enabling Constraints

One of the requirements of the framework is to respect the temporal constraints of the internal policy. In what follows, we show that IAP enforces temporal constraints of the internal policy and is able to enforce additional constraints during interoperation with an external domain.

Theorem 2: For an IAP policy and a corresponding query \( q = \langle r, PS, \tau \rangle \), user \( u_e \) can only acquire the permissions in \( q,PS \) during time \( \tau \) without violating any temporal constraint of the internal domain.

Proof: Based on hierarchical relations in Definition 4, \( u_e \) can only acquire permissions by activating A-juniors of \( r_e \). All such roles that are added to the IAP policy for the query \( q \) are assigned the time constraint \( \tau \). Therefore, \( u_e \) can only activate these roles during \( \tau \). Since these roles are linked to the internal policy roles using I-strong-hierarchy, they do not inherit permissions unless the internal roles are enabled.

C. Respecting Dynamic Separation of Duty Constraints

In this section, we first provide some definitions and then formally prove that IAP respects DSoDs.

Definition 5: A user can acquire a role if the user can activate it or one of its I-seniors.

Definition 6: A GTRBAC system enforces constraint \( c = dsoD(RS, k) \) if and only if no user can simultaneously acquire \( k \) or more roles in \( RS \).

Definition 7: A GTRBAC system strictly enforces constraint \( c = dsoD(RS, k) \) if and only if no user can simultaneously acquire \( k \) or more roles in \( RS \).

Strict enforcement of DSoD constraints is a desired property in RBAC systems, and in fact, one of the main reasons that led to introduction of hybrid hierarchy [15]. We assume that in the internal policy, the roles involved in a DSoD constraint may only have A-senior roles, and no I-senior roles. This guarantees that when a GTRBAC system enforces a DSoD constraint, it is enforced strictly. We call such a policy DSoD-well-formed.

Definition 8: A policy \( P \) is DSoD-well-formed if and only if \( \forall r \in RS, c = dsoD(RS, k) \in DSoD_P \) if and only if \( r \geq r' \).

Theorem 3: Let \( P \) and \( P' \) be a DSoD-well-formed GTRBAC system and its IAP, respectively. \( P' \) strictly enforces constraints in \( DSoD_P \).

Proof: Assume \( c = dsoD(RS, k) \in DSoD_P \) is not strictly enforced in \( P' \), i.e., a user can acquire \( RK \subseteq RS \) with \( k \) roles in system \( P' \). Note that \( RS \subseteq RP \). Since the user assignments and their respective role structure for the users in \( U_P \) is not different in \( P \) and \( P' \), the user that violates strict enforcement has to be \( u_e \). Among the roles in \( RP \), \( u_e \) can only acquire \( S(Q) \), i.e., the set of selected roles for interoperation queries, and the roles I-junior to them. Since \( P \) is DSoD-well-formed, the roles I-junior to the roles in \( S(Q) \) must not be involved in a DSoD. Thus, we have \( RK \subseteq S(Q) \). Since roles in \( RK \) are involved in a DSoD, based on Definition 4, we have \( RK' = M_e(RK) \subseteq UC \). \( u_e \) needs to activate all roles in \( RK' \) in order to acquire their corresponding roles in \( RK \). According to Definition 4, there exists a constraint \( c' = dsoD(RS', k) \), which should be enforced in \( P' \). Since \( RK' \subseteq RS' \), hence, \( u_e \) needs to activate \( k \) roles in \( RS' \) which contradicts the enforcement of \( c' \). Consequently, our assumption about non-strict enforcement of \( c \) is false.

Example 4: Let \( q_1 = \langle \{p_6, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}, r_e \in R_e, \tau = EveryDay \rangle \) be an interoperation query. Here, roles \( r_{EL}, r_{TA}, \) and \( r_{TBA} \) should be selected, but that would violate the DSoD constraint of \( TO \)'s internal policy: \( dsoD(\{r_{EL}, r_{TA}, r_{TBA}\}, 3) \) (i.e., selected roles cannot be activated simultaneously). Hence, this request is denied.
Creating a filter role that inherits from \( r_{EL} \), \( r_{TA} \) and \( r_{TBA} \) would result in a \( P' \) that is not DSoD-well-formed.

**Example 5:** Let \( q_1 = \langle \{p_7, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}, r_{e2} \in R_e, \text{EveryDay} > \rangle \) and \( q_2 = \langle \{p_6, r_{e3} \in R_e, \text{AllFridays} > \rangle \). Query \( q_1 \) can be granted by selecting roles \( r_{TA}, r_{TBA} \) and \( r_{TC} \). Note that role \( r_{TBA} \) can be activated only on Monday to Thursday. Although the request was issued for every day, because of the I-strong semantics, the activation constraint of \( r_{TBA} \) is respected and the external domain cannot inherit permission from \( r_{TBA} \) on Fridays, Saturdays and Sundays. To satisfy \( q_1 \), filter roles \( r_{o1 \rightarrow r_2}, r_{r_1 \rightarrow r_2} \) and \( r_{r_2 \rightarrow r_2} \) are created as shown in Figure 3. According to Definition 4, \( r_{o \rightarrow r_2} \in R_o \) and \( r_{r_1 \rightarrow r_2} \) and \( r_{r_2 \rightarrow r_2} \in R_e \) because the latter roles inherit from a role in DSoD. For query \( q_2 \), role \( r_{EL} \) should be selected. Two interoperation roles \( r_{o \rightarrow r_3} \in R_o \) and \( r_{r_2 \rightarrow r_3} \in R_e \) are created to satisfy this request as shown in Figure 3. Since \( r_{o \rightarrow r_3} \) is inheriting from \( r_{EL} \), which is in a DSoD constraint, a constraint \( \text{ssod}(\{r_{e1 \rightarrow r_2}, r_{r_2 \rightarrow r_2}, r_{r_2 \rightarrow r_3}\}, 3) \) is created. Hence, even if an external user is authorized for roles \( r_{r_2} \) and \( r_{r_3} \), he cannot activate them simultaneously, thus respecting the DSoD internal policy. Finally, note that filter roles created for \( q_1 \) and \( q_2 \) are initialized as follows: \( \text{UBS}(r_{o \rightarrow r_2}) = \text{UBS}(r_{o \rightarrow r_2}) = \text{UBS}(r_{o \rightarrow r_3}) = \tau_p \) and \( \text{UBS}(r_{o \rightarrow r_3}) = \text{UBS}(r_{o \rightarrow r_3}) = \{p_6\} \).

**D. Respecting Static Separation of Duty Constraints**

In this section, we show how the framework enforces SSoD constraints. According to the requirements presented in Section III, the policy of the external domain is not known. As a consequence, it is not possible to identify SSoD violations caused by the hierarchical structure or user assignments of the policy of the external domain. Furthermore, even if the policy of the external domain was known, in order to have a continuous enforcement of the internal SSoD constraints, the external domain should maintain its policy as it was when \( P' \) was constructed. Therefore, we propose to relax the SSoD to DSoD constraints to ensure that the permissions and roles in conflict are never acquired simultaneously by any user of the external domain. We capture this relaxation in Definition 4.

**Example 6:** Consider the policy presented in Figure 4 with \( \text{ssod}(\{r_3, r_7\}, 2) \). The hierarchy paths from \( r_{e1} \) to \( r_3 \) and from \( r_{e1} \) to \( r_7 \) represent a SSoD violation. This means that any user authorized for \( r_{e1} \) is authorized for \( r_3 \) and \( r_7 \). Moreover, user \( u_1 \) assigned to roles \( r_{e4} \) and \( r_{e6} \) violates the above-mentioned SSoD, by being authorized for \( r_3 \) and \( r_7 \). To mitigate this lack of control, we add \( \text{dsod}(\{r_{e3}, r_7\}, 2) \) to \( P' \), which ensures the relaxed enforcement of \( \text{ssod}(\{r_3, r_7\}, 2) \).

Now, we formally prove enforcement of SSoD constraints in our framework, based on the following definitions. We assume that there are no roles in the internal policy that inherit from the roles involved in SSoD constraints. We call such a policy SSoD-well-formed.

**Definition 9:** A GTRBAC system satisfies constraint \( c = \text{ssod}(r S, k) \) if and only if no user is authorized for \( k \) or more roles in \( R S \).

**Definition 10:** A policy \( P \) is SSoD-well-formed if and only if \( \forall r \in R S, c = \text{ssod}(r S, k) \in \text{SoD}_{\text{P}}[2r \in R P, r \geq 1, r] \).

**Theorem 4:** Let \( P \) satisfy \( SSoD_P \) and be SSoD-well-formed and DSoD-well-formed. Also, let \( P' \) be its IAP. \( P' \) strictly enforces constraints in \( SSoD_P \) as DSoDs.

**Proof:** As per Definition 4, there exists a constrained role and DSoD constraint for each selected role that is in a SSoD constraint. Therefore, constraints in \( SSoD_P \) are regarded as DSoDs and strictly enforced by \( P' \) as proved in Theorem 3.

VI. ALGORITHMS AND IMPLEMENTATION RESULTS

The interoperation policy is created in two phases, the role selection phase and the interoperation policy construction phase where the filter roles and the hierarchy of the interoperation policy are constructed. Algorithm 1 is used to process the queries, and Algorithm 2 presents the role mapping details. The role mapping works using a backtracking algorithm that greedily looks for the solution with the maximum possible coverage (defined in Section IV-B) given a request \( \langle P_{req}, \tau, r_{ex} \rangle \). In each iteration the role with maximum average coverage per queried permission is selected and, to respect the SSoD constraints, the search space is reduced by removing the roles that cannot be authorized any longer. For instance, given a constraint \( \text{ssod}(r S, k) \) if the algorithm has selected \( k - 1 \) roles in \( RS \), the search space is pruned accordingly. Similarly, the algorithm prunes the roles that cannot be activated at the same time due to DSoD constraints. If a solution for a request is found some post-processing is performed. Because the policy of the external domain is dynamic and unknown, we relax SSoD to DSoD as explained in Section V-D. The time complexity in the worst case is exponential to the number of roles in the internal policy, but in practice is less due to the pruning strategy used.

A. Implementation Results

We have implemented a prototype system to test the performance of the proposed algorithm. We generated 30 random policies and averaged the time required to process a query. Each role was randomly assigned a periodic time constraint. The ratios of role to SSoD and DSoD were set to 0.1, and of role to temporal constraints to 0.4. The ratio of roles to authorized permissions was kept 1:24. A randomly generated request was created. The number of requested permissions was kept to five, the number of hierarchical levels to seven and the
Algorithm 1 createInteroperationPolicy($r, PS, \tau$)

1: $R_{avail} \leftarrow R_P \setminus \text{juniors}(r \in SSoD_P)$ \{Candidate roles\}
2: $R_{sel} \leftarrow \emptyset$ \{Selected roles so far\}
3: $P_{rem} \leftarrow P$ \{Set of permissions that haven’t been found\}
4: $P_{part} \leftarrow \emptyset$ \{Permissions partially covered\}
5: $R_b \leftarrow \emptyset$ \{Global variable to store the best found solution\}
6: for all $p \in PS$ do
7: \quad add to $UT$ the pair $[p, \tau]$ \{Initialize uncovered time per permission\}
8: selectRolesMaxCoverage($P_{rem}, R_{avail}, R_{sel}, P_{part}, UT$) \{See Algorithm 2\}
9: if $R_b \neq \emptyset$ then
10: \quad Follow Definition 4
11: \quad return \{Could not find a solution for the query\}

Algorithm 2 selectRolesMaxCoverage($P_{rem}, R_{avail}, R_{sel}, P_{part}, UT$)

1: if $P_{rem} = \emptyset$ then
2: \quad if $R_b = \emptyset$ then
3: \quad \quad $R_b \leftarrow R_{sel}$
4: \quad else
5: \quad \quad if $CR(R_b, PS, \tau) < CR(R_{sel}, PS, \tau)$ then
6: \quad \quad \quad $R_b \leftarrow R_{sel}$
7: \quad \quad else if $(CR(R_b, PS, \tau) = CR(R_{sel}, PS, \tau)) \land \ R_b \cap R_{sel} \neq \emptyset$ then
8: \quad \quad \quad $R_b \leftarrow R_{sel}$
9: \quad \quad return \{Found candidate solution\}
10: \quad return \{Objective fulfilled.\}
11: for all $ssod(RS, k) \in SSoD_P$ do
12: \quad if $|R_{sel} \cap RS| = (k - 1)$ then
13: \quad \quad $R_{avail} \leftarrow R_{avail} \setminus (RS \setminus (R_{sel} \cap RS))$
14: \quad \quad if simultaneous then
15: \quad \quad \quad for all $dsod(RS, t) \in DSoD_P$ do
16: \quad \quad \quad \quad if $|R_{sel} \cap RS| = (k - 1)$ then
17: \quad \quad \quad \quad \quad $R_{avail} \leftarrow R_{avail} \setminus (RS \setminus (R_{sel} \cap RS))$
18: \quad \quad \quad \quad for all $ri \in R_{avail}$ do
19: \quad \quad \quad \quad \quad if $P_{rem} \cap P_{au}(ri) = \emptyset$ then
20: \quad \quad \quad \quad \quad \quad $R_{avail} \leftarrow R_{avail} \setminus ri$
21: \quad \quad \quad \quad \quad \quad if $R_{avail} = \emptyset$ then
22: \quad \quad \quad \quad \quad \quad \quad return
23: \quad \quad \quad \quad \quad \quad $r_{best} \leftarrow \text{roleMaxCoverage}(P_{rem}, R_{avail}, UT)$
24: \quad \quad \quad \quad \quad \quad $R_{avail} \leftarrow R_{avail} \setminus r_{best}$
25: \quad \quad \quad \quad \quad \quad newUT $\leftarrow UT$ \{Create a copy\}
26: \quad \quad \quad \quad \quad \quad updated newUT according to $r_{best}$ activation constraint
27: \quad \quad \quad \quad \quad \quad $P_{newPart} \leftarrow P_{part} \cup P_{au}(r_{best})$
28: \quad \quad \quad \quad \quad \quad $P_{newRem} \leftarrow P_{rem}$
29: \quad \quad \quad \quad \quad \quad for all $p \in P_{rem}$ do
30: \quad \quad \quad \quad \quad \quad \quad if $[p, \tau] \in newUT$ has $\tau = 0$ then
31: \quad \quad \quad \quad \quad \quad \quad \quad $P_{newRem} \leftarrow P_{newRem} \setminus p$
32: \quad \quad \quad \quad \quad \quad \quad else if $\forall r \in R_{avail} :$ coverage(newUT($p$), $r$) = 0 \land ($p \in P_{newPart}$) then
33: \quad \quad \quad \quad \quad \quad \quad \quad $P_{newRem} \leftarrow P_{newRem} \setminus p$
34: \quad \quad \quad \quad \quad \quad selectRolesMaxCoverage($P_{newRem}, R_{avail}, (R_{sel} \cup \{r_{best}\}), P_{newPart}, newUT$)
35: \quad \quad \quad \quad \quad \quad selectRolesMaxCoverage($P_{rem}, R_{avail}, R_{sel}, P_{part}, UT$)
36: \quad \quad \quad \quad \quad \quad return

process the requests because of the proposed greedy heuristic for processing different requests against different policies. The number of requested permissions does not seem to influence per se the time required to find a solution. The effect of augmenting the number of roles in $P$ while keeping the remaining variables constant is shown in Figure 5b. The results suggest that the time required to find a solution does not increase considerably when the number of roles in the internal policy is augmented and the remaining parameters are maintained unchanged. Finally, we find that the performance of the algorithm depends on the configuration of $P$, as Figure 5c suggests. The experiment was run increasing the maximum number of junior roles in $P$ and maintaining the other variables fixed. Each line represents the results for policies of 3 and 7 hierarchy levels. As can be seen, augmenting the number of junior roles increases the time required to find a solution for an interoperation query. This is due to the increase of density of commonly authorized permissions per role, caused by the inheritance of permissions, which in turn increases the size of the search space. The experimental results suggest that the algorithm can find solutions in reasonable time.

VII. RELATED WORK

Several algorithms related to the role mapping problem exist, but they do not consider temporal constraints. In [3], simulated annealing and genetic algorithms have been used to find a suboptimal mapping. In [2], an alternative objective function for the role mapping algorithm has been proposed to consider the least privilege principle by penalizing solutions that contain roles with extra permissions. In [3], [22], [2],
[21], various algorithms have been proposed to solve the role mapping problem in presence of hybrid hierarchy - none, however, consider temporal, SSoD and DSoD constraints.

In [14], an approach to consider temporal constraints has been proposed. Their approach may override the internal temporal constraints as the configuration of the internal domain is not considered. Wickramarachchi et. al [19] propose heuristics to solve the role mapping problem considering the DSoD constraints. However, their work does not consider the SSoD or temporal constraints, and their algorithms may provide more permissions than those strictly required for the interoperation. Algorithms proposed in [18] aim to respect several constraints in presence of hybrid hierarchy, yet their approach has some important limitations. The roles of the internal and the external domains are mapped directly. Since they do not use an intermediate layer, additional constraints cannot be established on the interoperation. In addition, to respect SSoD constraints, their algorithm assumes that the policy of the external domain is known and static and may modify their policies during interoperation periods.

Using filter roles allows our proposed approach to provide only the permission needed without modifying the internal policy. To the best of our knowledge, two alternative solutions exist. In [18], [14], [20], the roles of the internal policy are split to a degree in which the selection of a particular role provides the exact set of required permissions. However, splitting the roles affects the organization and the original semantics of the hierarchy making it unmanageable. In the approach proposed in [22], [21], [19] the principle of least privilege is sacrificed when interoperation needs are more important. The solution we propose avoids having to make such a compromise by filtering out the permissions that should not be provided to the external domain, without modifying the policy of the internal domain. In [7], a similar notion of filter role is used, yet it is limited to the semantics of the intra-domain delegation. To the best of our knowledge, this paper is the first to introduce filter roles in the context of multi-domain environments.

In addition to practical and secure support for SoD and temporal constraints, our framework is distinguished from the existing approaches in the following aspects: 1) The proposed framework respects the principle of least privilege, as it provides the exact set of required permissions, unlike other approaches [22], [21], [19] that provide more permissions than those requested. 2) The proposed solution maintains the internal policy unmodified, while the existing approaches [18], [14], [20] can potentially modify the internal policy for every interoperation request. 3) To the best of our knowledge, our work is the first one to propose an algorithm that respects the temporal constraints of the internal domain.

VIII. CONCLUSIONS

In this paper, we have proposed a secure role-based framework that allows time-constrained interoperation between domains, and complies with SoD and temporal constraints of the internal domain policy. In addition, we consider a number of other requirements such as maximizing the available interoperation time, assuming lack of knowledge and control over the policy of the external domain, and avoiding changing in the internal policy as the result of interoperation. We also prove formally that our framework meets our security requirements such as conforming with the constraints. As future work, we plan to investigate reuse of the same interoperation policy for multiple interoperations, in order to minimize the enforcement load on the internal domain. We also plan to incorporate additional temporal constraints proposed in GTRBAC.

ACKNOWLEDGMENT

This research has been supported by the US National Science Foundation award IIS-0545912.

REFERENCES