Enhancing Relationship-Based Access Control Policies with Negative Rule Mining

Ferhat Demirkiran
University at Albany
Albany, New York, USA
fdemirkiran@albany.edu

Amir Masoumzadeh
University at Albany
Albany, New York, USA
amasoumzadeh@albany.edu

Abstract

Relationship-based access control (ReBAC) policies often rely solely on positive authorization rules, implicitly denying all other requests by default. However, many scenarios require explicitly stating negative authorization rules to capture exceptions or special restrictions that are not naturally enforced by deny-by-default semantics. This work presents a systematic method to mine ReBAC policies that integrate both positive and negative authorization rules from observed authorizations. We formalize the mining problem, show its NP-hardness, and develop an approach that identifies minimal policies while accurately reflecting observed access decisions. We demonstrate the feasibility and effectiveness of our proposed approach through a set of experiments. Our experimental evaluations on representative datasets demonstrate that including negative rules leads to more concise and semantically complete policies, confirming the necessity of explicit negative authorizations in complex access control settings.

CCS Concepts

• **Security and privacy**;

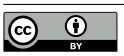
Keywords

relationship-based access control; policy mining; negative authorization; deny rules; policy optimization

ACM Reference Format:
Ferhat Demirkiran and Amir Masoumzadeh. 2025. Enhancing Relationship-Based Access Control Policies with Negative Rule Mining. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25), June 4–6, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3714393.3726510

1 Introduction

Relationship-based access control (ReBAC) has emerged as a flexible and expressive framework for managing access permissions. ReBAC policies enable more granular and context-aware access control decisions by utilizing the connections among users and resources [2]. In various domains, such as online social networks [11], healthcare systems [24], and collaborative platforms, such relationships are fundamental to defining who should have access to what resources.



CODASPY '25, June 4–6, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-1476-4/2025/06
https://doi.org/10.1145/3714393.3726510

Traditionally, the focus in the mining of ReBAC policies has been on extracting positive authorization rules, that is, rules that specify when access should be granted based on certain relationship patterns. Yet, effective policies often require not only stating when access is allowed but also when it must be explicitly denied [20]. Negative authorization rules are essential for capturing exceptions and prohibitions that cannot be expressed solely through deny-by-default semantics. Consider a social media platform where Alice and Bob share a friendship relationship, granting each other the permission to view each other's posts. However, when an explicit blocked relationship is established between Alice and Bob, this negative authorization rule overrides the existing friendship, ensuring that Alice is denied access to Bob's content despite their friendship connection. Without the incorporation of such negative rules, the ReBAC policies would rely solely on implicit denials, potentially leading to unauthorized access and compromising user privacy.

Despite the clear need, however, integrating negative authorization rules into ReBAC policy mining introduces unique challenges. One primary challenge is distinguishing between access denials that result from explicit negative authorization rules and those that arise from a deny-by-default policy, which is often implicitly enforced in access control systems [21]. Accurately attributing access denials to specific negative authorization rules requires sophisticated analysis and a deep understanding of the underlying relationship patterns. Furthermore, negative authorization rules often represent exceptions to general access patterns, necessitating to identify and categorize these exceptions without introducing inconsistencies or conflicts within the policy. Effectively addressing these challenges requires not only understanding of access control patterns, but also a focus on policy minimality, ensuring that the resulting policies capture the required access control decisions with the minimal set of rules. This balance between expressiveness and minimality is crucial for maintaining policies that are comprehensive and practical for real-world implementation [16].

Addressing these challenges, this paper presents the first systematic approach for mining ReBAC policies that incorporate both positive (`PERMIT`) and negative (`DENY`) authorization rules. Our approach thoroughly explores the authorization space, identifies relationship patterns for user-resource pairs, and applies a categorization and optimization process to distinguish implicit and explicit denial decisions and derive minimal policies. Our specific contributions in this work are as follows.

- We formulate the problem of mining ReBAC policies with both positive and negative authorization rules.
- We show that the proposed mining problem is NP-hard.
- We propose a two-stage solution that mines positive and negative authorization rules by translating each stage to

an instance of the classic set cover problem, leading to an approximately optimal result. We provide a comprehensive time complexity analysis of the solution as well.
- We experimentally validate our approach on datasets of varying complexity, demonstrating its correctness, scalability, and conciseness. Additionally, we provide a feasibility analysis that empirically establishes the necessity of explicit negative rules for representing complex access control requirements.

In the remainder of the paper, we formalize the ReBAC policy model and mining problem (Section 2), discuss our policy mining algorithms and their analysis (Section 3), present the results of our experimental evaluations (Section 4), review the closely related work (Section 5), and discuss the significance of our findings for future research and practice (Section 6).

2 Preliminaries and Problem Statement

ReBAC extends classical access control by incorporating the relationships among users and resources when making authorization decisions [15]. Specifically, ReBAC policies determine whether a user can access a resource by examining the path of relationships in a graph that connects user to resource.

The central problem addressed in this paper is the automated mining of ReBAC policies that incorporate both positive and negative authorization rules. Existing literature has primarily focused on discovering relationship patterns that justify granting permissions, implicitly treating all unpermitted accesses as denied. While this positive authorization approach can suffice for some scenarios, many systems require explicit negative authorization rules to enforce specific prohibitions. For example, an organization might grant contractors access to certain internal resources, while explicitly denying access to those who also contract with a competing organization. Without the ability to capture such negative rules it fails to capture explicit denials required in many real-world contexts.

To formally define the problem of mining optimal ReBAC policies, we first define a ReBAC system. In applications that support ReBAC policies, the system information is modeled as a system graph [13], where nodes represent users and resources, and edges denote the relationships between them.

Definition 2.1 (System Graph). Let U be the set of users and R be the set of resources within a system. Let L be the set of possible relationship labels between users and resources. The *System Graph* $G = \langle V, E \rangle$ is a directed graph where:

- $V = U \cup R$ is the set of vertices representing users and resources.
- $E \subseteq V \times V \times L$ is the set of directed, labeled edges that represent relationships between entities.

Building upon the system graph G, the ReBAC policies define authorization rules based on path-based relationship patterns specified about the graph.

Definition 2.2 (Relationship Pattern). A *Relationship Pattern* ϕ is a sequence $[l_1, l_2, \ldots, l_n]$, where each l_i is a relationship label in L and $n \geq 1$. We refer to n as the *length* of the relationship pattern, i.e., the number of relationship labels in that sequence. The domain of relationship patterns is denoted by Φ.

Suppose a system graph contains a path from u to r of the form:

$$u \xrightarrow{\text{friend}} x \xrightarrow{\text{colleague}} r.$$

Here, the path corresponds to the relationship pattern:

$$\phi = \langle \text{friend}, \text{colleague} \rangle.$$

Its length is 2 because it involves exactly two labels in the path.

Definition 2.3 (Authorization Rule). An *Authorization Rule* is a tuple $\langle \phi, d \rangle$ where ϕ is a relationship pattern and $d \in \{\texttt{PERMIT}, \texttt{DENY}\}$ is the associated access decision with ϕ.

To simplify our discussions in this paper, we assume that all authorization rules pertain to a singular right or action. Considering an additional component in the above definition that represents the applicable right would be trivial. We define a ReBAC policy as a collection of authorization rules.

Definition 2.4 (Policy). A *Policy* ρ is a set of authorization rules $\{\langle \phi_i, d_i \rangle\}$, where each $\langle \phi_i, d_i \rangle$ represents an authorization rule, i.e., a pair of relationship pattern and associated decision.

An access request is evaluated based on whether it matches any of the rules defined in the policy.

Definition 2.5 (Access Request Evaluation). An access request $\langle u, r \rangle$, where $u \in U$ and $r \in R$, matches a relationship pattern ϕ if there exists a path in G from u to r whose sequence of edge labels corresponds to ϕ. The access decision for $\langle u, r \rangle$ is determined by the policy ρ by considering the following rules in order:

(1) If there exists an authorization rule $\langle \phi, \texttt{DENY} \rangle \in \rho$ such that ϕ matches the access request, then $\langle u, r \rangle$ is denied.
(2) If there exists an authorization rule $\langle \phi, \texttt{PERMIT} \rangle \in \rho$ such that ϕ matches the access request, then $\langle u, r \rangle$ is permitted.
(3) Otherwise, $\langle u, r \rangle$ is denied by default.

If both PERMIT and DENY rules match the same access request, the above process ensures that the DENY decision takes precedence. This prioritization avoids ambiguity and ensures a secure outcome in case of conflicting rules.

Our last preliminary definition captures a collection of access request evaluations that forms the input to a policy mining process.

Definition 2.6 (Access Log). Given the system graph G, an *access log* $\mathcal{A}$ is a set of tuples $\mathcal{A} = \{(u, r, d)\}$ where:

(1) $u \in U$ is a user requesting access.
(2) $r \in R$ is the resource being accessed.
(3) $d \in \{\texttt{PERMIT}, \texttt{DENY}\}$ is the access decision outcome for the request.

Given an access log defined, our objective is to construct a ReBAC policy ρ, composed of both PERMIT and DENY authorization rules, that accurately reflects the observed decisions. In practice, this means that the policy must be consistent with the access log: for every access request (u, r, d) recorded in the log, the policy must yield the same decision. If the log indicates that a particular user-resource pair is permitted, the policy should allow it; if the log

indicates a denial, the policy should likewise deny that access. In addition to consistency, we favor policies that are minimal. Among all policies that satisfy the consistency requirement, a minimal policy is one that uses the fewest possible authorization rules. This focus on minimality reduces redundancy and enhances manageability by limiting the number of rules that need to be maintained.

Definition 2.7 (Optimal ReBAC Policy Mining Problem). Given a system graph $G = \langle V, E \rangle$ and an access log $\mathcal{A}$, find an optimal ReBAC policy ρ consisting of both positive and negative authorization rules. The policy ρ must satisfy the following criteria:

(1) **Consistency**: For every access request $(u, r, d) \in \mathcal{A}$, the policy ρ must yield a decision consistent with d.
(2) **Minimality**: Let Σ denote the set of all policies that are consistent with $\mathcal{A}$. Then an optimal policy ρ^* satisfies $|\rho^*| \leq |\rho|$ for every $\rho \in \Sigma$; equivalently, there exists no consistent policy ρ' with $|\rho'| < |\rho^*|$, where $|\rho|$ denotes the number of rules in the policy ρ.

By ensuring consistency and minimality, the resulting policy ρ becomes both manageable and interpretable while faithfully representing the observed access decisions.

3 Methodology

This section presents a systematic approach for mining ReBAC policies that incorporate both positive (`PERMIT`) and negative (`DENY`) authorization rules. The methodology directly addresses the challenges outlined in the problem statement, including accurately attributing access denials, handling policy exceptions, and achieving policy minimality. The approach consists of the following steps:

(1) **Pattern Extraction and Mapping**: We identify all relationship patterns of length up to a specified bound k that relate users to resources, and map each pattern to the set of user-resource pairs it matches.
(2) **Categorization and Refinement of Relationship Patterns**: We categorize relationship patterns based on the observed access decisions, distinguishing those that could serve as `PERMIT` rules, `DENY` rules, or require refinement to resolve conflicts.
(3) **Policy Optimization**: We optimize the selection of authorization rules to find minimal sets of rules that accurately represent the desired access control decisions.

The remainder of this section provides a detailed discussion of each step, along with their associated algorithms and complexity considerations.

3.1 Pattern Extraction and Mapping

The initial step involves extracting all applicable relationship patterns from the system graph and creating a mapping between those patterns and their applicable set of user-resource pairs. A pattern is applicable to a user-resource pair if a path exist between them correspondig to the pattern.

Since the number of possible paths can be large, we limit our search to paths of length at most k, a user-specified parameter. This length constraint ensures computational feasibility while capturing all possible relationship patterns up to a maximum length that is practical. Thus, we perform a breadth-first traversal from each user node u, exploring all reachable resources within k hops. For each discovered path, we record the relationship pattern ϕ defined by its edge labels, and associate ϕ with the corresponding user-resource pairs $\langle u, r \rangle$. Algorithm 1 outlines the construction of mapping $\mathcal{M}$ from relationship patterns to sets of user-resource pairs.

Algorithm 1 Mapping Relationship Patterns to User-Resource Pairs

Input: System Graph $G = \langle U \cup R, E \rangle$, Maximum Pattern Length k
Output: Mapping $\mathcal{M} : \Phi \to \{U \times R\}$

```
M[φ ∈ Φ] ← ∅
for all u ∈ U do
    Create a queue Q and enqueue ⟨u, [ ]⟩, where [ ] is the empty pattern
    while Q is not empty do
        Dequeue ⟨v, φ'⟩ from Q
        if v ∈ R then
            M[φ'] ← M[φ'] ∪ ⟨u, v⟩
        if |φ'| < k then
            for all (v, w, ℓ) ∈ E where v is the source node do
                φ'' ← φ' ∘ [ℓ]                ▷ Append edge label ℓ
                Enqueue ⟨w, φ''⟩ into Q
```

Complexity Analysis of Algorithm 1. The complexity of the breadth-first traversal from a node is dependent on how many edges may be followed from each vertex in the graph. Theoretically, that would be $O(|V| \cdot |L|)$. However, such vertex out-degree will be much smaller in real-world system graphs. Let d be the maximum out-degree of a vertex in the system graph. Given the maximum length of patterns, k, each breadth-first traversal explores up to d^k paths. Thus, the overall worst-case complexity of the algorithm is $O(|V| \cdot d^k)$.

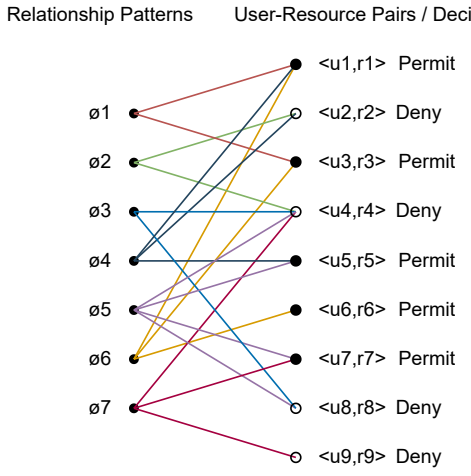


Figure 1: Mapping of Each Relationship Patterns ϕ_i to Matched User-Resource Pairs $\langle u, r \rangle$ in the System Graph. Empty Circles Correspond to `DENY` pairs.

Example 3.1. Figure 1 illustrates the correspondence of relationship patterns and access requests in the log in the context of a small

system graph that will be used as a running example in the rest of this section. There are seven relationship patterns, $\{\phi_1, \phi_2, \ldots, \phi_7\}$, each pattern ϕ_i matches a set of $\langle u, r\rangle$ pairs. For example, ϕ_1 matches $\langle u_1, r_1\rangle$ and $\langle u_3, r_3\rangle$, whereas ϕ_2 matches $\langle u_2, r_2\rangle$ and $\langle u_4, r_4\rangle$. These sets collectively form the output of the mapping step and will be the foundation for subsequent categorization and refinement.

3.2 Categorization and Refinement of Relationship Patterns

After constructing $\mathcal{M}$, we obtain a set of relationship patterns Φ along with the corresponding user-resource pairs that each pattern matches. Utilizing the access log $\mathcal{A}$, which records observed (u, r, d) triples (where $d \in \{\texttt{PERMIT}, \texttt{DENY}\}$), we categorize each pattern $\phi \in \Phi$ based on the decisions associated with its matched user-resource pairs as follows:

- **Candidate `PERMIT` Patterns without Conflicts** (P^+): All $\langle u, r\rangle \in \mathcal{M}[\phi]$ are associated with a `PERMIT` decision in $\mathcal{A}$.
- **Candidate `DENY` Patterns without Conflicts** (P^-): All $\langle u, r\rangle \in \mathcal{M}[\phi]$ are associated with a `DENY` decision in $\mathcal{A}$.
- **Candidate `PERMIT` Patterns with Conflicts** ($P^\pm$): Some $\langle u, r\rangle \in \mathcal{M}[\phi]$ are permitted while others are denied.

We also track the set of user-resource pairs that are denied by P^- patterns (D_{UR}). If a denial is not part of D_{UR}, we can confidently attribute it to the default deny policy, as it cannot be explained by any potential `DENY` rules. Algorithm 2 outlines the process for categorizing patterns.

Algorithm 2 Categorizing Relationship Patterns

Input: Set of Relationship Patterns Φ; Mapping $\mathcal{M}$; Access Log $\mathcal{A}$
Output: Sets of Candidate Non-Conflicting, `PERMIT` Patterns P^+, `DENY` Patterns P^-, Conflicting `PERMIT` Patterns $P^\pm$; User-Resource Pairs Corresponding to Candidate `DENY` Patterns D_{UR}

1: $P^+ \leftarrow \emptyset$; $P^- \leftarrow \emptyset$; $P^\pm \leftarrow \emptyset$; $D_{UR} \leftarrow \emptyset$
2: **for all** $\phi \in \Phi$ **do**
3: $\quad D \leftarrow \{d \mid \langle u, r, d\rangle \in \mathcal{A}, \langle u, r\rangle \in \mathcal{M}[\phi]\}$
4: $\quad$ **if** D = {`PERMIT`} **then**
5: $\quad\quad P^+ \leftarrow P^+ \cup \{\phi\}$
6: $\quad$ **else if** D = {`DENY`} **then**
7: $\quad\quad P^- \leftarrow P^- \cup \{\phi\}$
8: $\quad\quad D_{UR} \leftarrow D_{UR} \cup \mathcal{M}[\phi]$
9: $\quad$ **else** ▷ *Pattern with mixed decisions*
10: $\quad\quad P^\pm \leftarrow P^\pm \cup \{\phi\}$

Complexity Analysis of Algorithm 2. The loop in the algorithm iterates over $|\Phi|$, which will be $O(|L|^k)$, where k is the user-specified, maximum length of patterns considered. Constructing D in every iteration of the loop on Line 3 essentially requires calculating the intersection of the user-resource pairs in the log records and those associated with a pattern. Assuming that those sets are sorted by user-resource pairs, the intersection operation will take $O(|V|^2)$ (more accurately, $O(|U| \cdot |R|)$. Assuming a log-linear complexity for a sort operation, the overall complexity of the algorithm is $O(|L|^k \cdot |V|^2 \log |V|)$.

After categorizing the relationship patterns into sets P^+, P^-, and $P^\pm$, we proceed to refine $P^\pm$ to ensure that it is not associated with any `DENY` decision that cannot be explained by patterns in P^-. Any denials not explained by P^- has to result from the deny-by-default policy. Therefore, $P^\pm$ patterns with such `DENY` decisions cannot be considered as potential `PERMIT` pattern. Specifically, for each pattern in $P^\pm$, we examine whether it has a denied user-resource pair not included in the user-resource pairs D_{UR} (those that correspond to patterns in P^-, collected in Algorithm 2). Algorithm 3 illustrates the refinement process.

Algorithm 3 Refinement of Candidate `PERMIT` Patterns with Conflicts

Input: Candidate Conflicting `PERMIT` Patterns $P^\pm$; User-Resource Pairs Corresponding to Candidate `DENY` Patterns D_{UR}; Mapping $\mathcal{M}$; Access Log $\mathcal{A}$
Output: Refined Candidate `PERMIT` Patterns with Conflicts $P^\pm_{\text{refined}}$

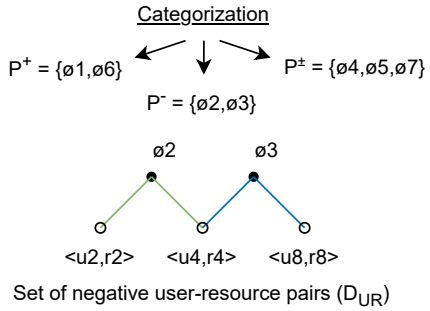
1: $P^\pm_{\text{refined}} \leftarrow \emptyset$
2: **for all** $\phi \in P^\pm$ **do**
3: $\quad \mathcal{U}^\phi_{\texttt{DENY}} = \{\langle u, r\rangle \mid \langle u, r\rangle \in \mathcal{M}[\phi], (u, r, \texttt{DENY}) \in \mathcal{A}\}$
4: $\quad \mathcal{U}^\phi_{\text{Unexplained-DENY}} = \mathcal{D}^\phi_{\texttt{DENY}} \setminus D_{UR}$
5: $\quad$ **if** $\mathcal{U}^\phi_{\text{Unexplained-DENY}} = \emptyset$ **then**
6: $\quad\quad P^\pm_{\text{refined}} \leftarrow P^\pm_{\text{refined}} \cup \{\phi\}$

Complexity Analysis of Algorithm 3. In each iteration of the loop, matching user-resource pairs with `DENY` decisions in the log on Line 3 takes $O(|V|^2 \log |V|)$ (similar to the analysis of Line 3 in Algorithm 2). The subsequent lines take at most $O(|V|^2)$. Thus, the overall complexity of the algorithm is $O(|L|^k \cdot |V|^2 \log |V|)$.

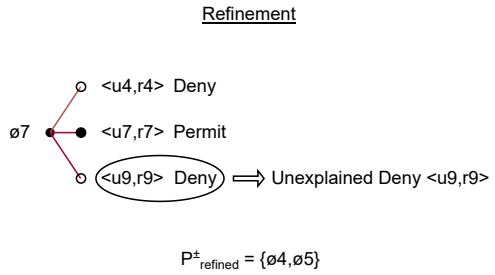
Example 3.2. Figure 2 demonstrates how the seven relationship patterns from Example 3.1 are divided into different sets based on the observed `PERMIT` or `DENY` decisions. First, each pattern is assigned to one of the three sets P^+, P^-, or $P^\pm$ (Figure 2a). Patterns ϕ_1 and ϕ_6 are in P^+, as they only match user-resource pairs with `PERMIT` decisions, while ϕ_2 and ϕ_3 are placed in P^-, as they match only `DENY` decisions. Patterns ϕ_4, ϕ_5, and ϕ_7 are in $P^\pm$, as they each match some `PERMIT` and some `DENY` decisions. Figure 2b illustrates the refinement algorithm, which inspects each relationship pattern in $P^\pm$. If a relationship pattern explains a `DENY` decision *not* accounted for by P^- (i.e., not in D_{UR}), that relationship pattern cannot be a candidate for final set of `PERMIT` patterns. Consider relationship pattern ϕ_7 and the corresponding pair $\langle u_9, r_9\rangle$, which has a `DENY` decision. However, this denial cannot be explained by any negative relationship pattern in P^- (Note that $\langle u_9, r_9\rangle$ is not present in D_{UR}). Therefore, it must result from the system's deny-by-default policy. Consequently, ϕ_7 is excluded from $P^\pm_{\text{refined}}$, and will not be considered as a candidate `PERMIT` pattern in the policy optimization step.

3.3 Policy Optimization

Having identified P^+, P^-, and refined $P^\pm$, we combine P^+ and $P^\pm_{\text{refined}}$ to form a candidate set of patterns for `PERMIT` rules. Similarly, P^- defines candidate patterns for `DENY` rules. The objective of



(a) Categorization of Relationship Patterns Into P^+, P^-, and $P^\pm$.



(b) Refinement of $P^\pm$, Excluding Relationship Patterns Corresponding To Deny-by-Default Decisions.

Figure 2: Categorization and Refinement Processes.

ReBAC policy mining optimization is to select a minimal set of rules that fully and precisely represents the observed access decisions. In this section, we formalize the optimization of the ReBAC policy mining process involving both PERMIT and DENY decisions. We first show that the proposed mining problem is NP-hard.

Theorem 3.3. *The ReBAC policy optimization problem, as defined above, is NP-hard.*

Proof. We prove NP-hardness by showing a polynomial-time reduction from the *set cover* problem to our ReBAC policy optimization problem.

Set Cover Problem: In the classic *set cover* problem, we are given:

- A universe $\mathcal{U}$ of elements.
- A family of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ such that $\bigcup_{i=1}^{m} S_i = \mathcal{U}$.

The objective is to find the minimum number of subsets $\mathcal{S}' \subseteq \mathcal{S}$ whose union still covers $\mathcal{U}$.

Reduction Construction: Given an instance of the *set cover* problem $(\mathcal{U}, \mathcal{S})$, we construct an instance of our ReBAC policy optimization problem as follows:

(1) Universe to User-Resource Pairs: For each element $e \in \mathcal{U}$, introduce a unique user-resource pair $\langle u_e, r_e \rangle$. We assume that the access log $\mathcal{A}$ grants PERMIT decisions for exactly these pairs. Thus, the set of all permitted pairs $\mathcal{U}_{\text{permit}}$ corresponds directly to the universe $\mathcal{U}$.
(2) Subsets to Candidate Patterns: For each subset $S_i \in \mathcal{S}$, define a corresponding Candidate PERMIT pattern $\phi_i \in P^+$. Define mapping $\mathcal{M}[\phi_i]$ as the set of all user-resource pairs $\langle u_e, r_e \rangle$ corresponding to elements $e \in S_i$. Hence, the domain covered by rule ϕ_i directly matches S_i in the original *set cover* instance. Also, consider $P^- = P^\pm_{\text{refined}} = \emptyset$.
(3) Minimal Coverage: Our objective in the ReBAC instance is to select the minimal subset of rules $P_{\min} \subseteq P^+ \cup P^\pm_{\text{refined}}$ such that:
$$\bigcup_{\phi \in P_{\min}} \mathcal{U}_\phi = \mathcal{U}_{\text{permit}},$$
mirroring the requirement in the *set cover* problem that a minimal subcollection of subsets covers $\mathcal{U}$.

Correctness of the Reduction: The construction ensures a one-to-one correspondence between elements of $\mathcal{U}$ and permitted user-resource pairs, and between subsets S_i and candidate PERMIT patterns ϕ_i. A solution that selects a minimal set of rules $P_{\min}$ covering all permitted pairs $\mathcal{U}_{\text{permit}}$ in our ReBAC instance directly corresponds to a minimal set cover $\mathcal{S}'$ of $\mathcal{U}$ in the original problem. Likewise, any minimal set cover $\mathcal{S}'$ translates into a minimal set of authorization rules $P_{\min}$.

NP-hardness: The *set cover* problem is well-known to be NP-hard. Since our reduction is polynomial-time and faithfully translates *set cover* instances into ReBAC policy optimization instances, the NP-hardness of *set cover* implies that the ReBAC policy optimization problem is also NP-hard. □

The above NP-hardness proof also provides us an intuitive approach to solve our mining problem. The set cover problem has been studied extensively in the literature including proposals for approximation algorithms to solve it. We propose a two-stage solution for the mining problem that relies on solving a set cover problem in each stage. This approach ensures reaching a relatively optimal solution in polynomial time.

The pseudocode for mining authorization rules is shown in Algorithm 4. In the first stage, we concentrate on mining PERMIT authorization rules. Our objective is to select the minimal subset of candidate PERMIT patterns, including those without conflicts (P^+) and with conflicts ($P^\pm_{\text{refined}}$), such that all PERMIT cases in the authorization log are covered. We model this stage of the solution as a set cover problem as follows. The universe is the set of all user-resource pairs with PERMIT decision (Line 2). The subsets include user-resource pairs corresponding to each pattern in $P^+ \cup P^\pm_{\text{refined}}$ with PERMIT decision (Line 4). The solution of this set cover instance determines the patterns for PERMIT authorization rules in the policy (Line 5). Selected patterns are guaranteed to cover the expected PERMIT cases in the log. However, if they include any of the patterns in $P^\pm_{\text{refined}}$, they result in extra PERMIT cases that are denied in the authorization log. In the second stage, we concentrate on mining DENY authorization rules that lead to denying those extra-permitted cases. We first form the universe of our second instance of set cover problem as all user-resource pairs that are supposed to be denied, but would be permitted by the previously selected $P^\pm_{\text{refined}}$ in the minimal PERMIT patterns (Line 9). The subsets include user-resource pairs corresponding to the P^- patterns that overlap with this universe. The solution to this second set cover instance achieves minimal patterns for DENY authorization rules in the policy.

Algorithm 4 Mine Minimal Policy

Input: Sets of Candidate Non-Conflicting PERMIT Patterns P^+, Refined Conflicting Patterns $P^{\pm}_{\text{refined}}$, DENY Patterns P^-; User-Resource Pairs Corresponding to Candidate DENY Patterns D_{UR}; Mapping $\mathcal{M}$; Access Log $\mathcal{A}$

Output: Minimal Subset of Patterns for PERMIT and DENY Rules: $P^+_{\text{min}}, P^-_{\text{min}}$

▷ *Mine* PERMIT *rules* ◁

$\mathcal{U}_{\texttt{PERMIT}} \leftarrow \{\langle u, r\rangle \mid \langle u, r, \texttt{PERMIT}\rangle \in \mathcal{A}\}$

for all $\phi_i \in (P^+ \cup P^{\pm}_{\text{refined}})$ **do**

  $S^+_i \leftarrow \mathcal{M}[\phi_i] \cap \mathcal{U}_{\texttt{PERMIT}}$

$P^+_{\text{min}} = \{\phi_i\}$ where ϕ_i corresponds to S^+_i selected by solving Set Cover problem for universe $\mathcal{U}_{\texttt{PERMIT}}$ and subsets $\{S^+_i\}$

▷ *Mine* DENY *rules* ◁

$\mathcal{U}_{\texttt{DENY}} \leftarrow \emptyset$

for all $\phi \in (P^+_{\text{min}} \cap P^{\pm}_{\text{refined}})$ **do**

  $\mathcal{U}_{\texttt{DENY}} \leftarrow \mathcal{U}_{\texttt{DENY}} \cup (\mathcal{M}[\phi] \cap D_{UR})$

for all $\phi_i \in P^-$ **do**

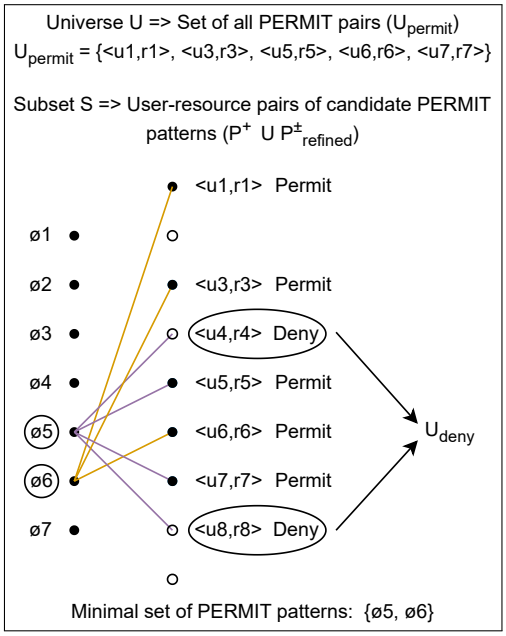
  $S^-_i \leftarrow \mathcal{M}[\phi_i] \cap \mathcal{U}_{\texttt{DENY}}$

$P^-_{\text{min}} = \{\phi_i\}$ where ϕ_i corresponds to S^-_i selected by solving Set Cover problem for universe $\mathcal{U}_{\texttt{DENY}}$ and subsets $\{S^-_i\}$

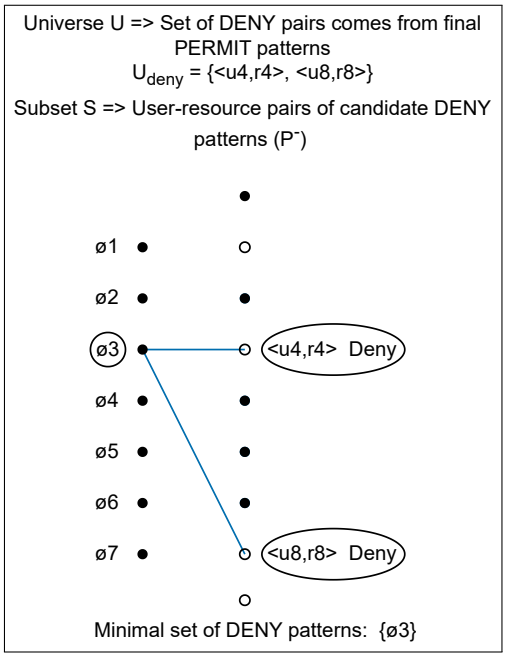
Complexity Analysis of Algorithm 4. Preparing the universe and subset inputs for each instance of set cover takes $O(|V|^2)$ and $O(|L|^k \cdot |V|^2 \log |V|)$, respectively. Note that preparing subsets involve sorting and intersection operations. The complexity of our algorithm is clearly dependent on the complexity of the set cover solution that is used as well. The well-known greedy algorithm for set cover [12] achieves an approximation factor of $\ln n$, where n is the size of the universe. It has been also shown that, unless $P = NP$, no polynomial-time algorithm can attain an approximation factor significantly better than $(1 - o(1)) \ln n$ [14]. This highlights the inherent difficulty in surpassing the greedy algorithm's performance in the worst-case scenario. The complexity of the greedy solution is $O(m \cdot n \log n)$ where m is the number of subsets and n is the size of the universe. In our setup, m and n will be $|\Phi|$ and $O(|V|^2)$, respectively. Therefore, the complexity of Algorithm 4 is $O(|L|^k \cdot |V|^2 \log |V|)$.

Overall Complexity Consideration: Considering the complexity of all algorithms discussed, the time complexity of our proposed solution will be $O(|L|^k \cdot |V|^2 \log |V| + |V| \cdot d^k)$ where d is the maximum out-degree of a vertex in the system graph and k is the maximum length of patterns to be considered. Considering that the practical choices for pattern lengths will be relatively small (e.g., we consider $k = 5$ in our experiments), the algorithm will be of polynomial complexity in practice. We also note that $|L|$ and d will be relatively small in many real-world systems. Therefore, the overall complexity in practice would be log quadratic in terms of the number of system graph vertices.

Example 3.4. Continuing from Example 3.2, Figure 3 illustrates how the minimal set of PERMIT and DENY patterns are identified. Figure 3a demonstrates the selection of a minimal subset of PERMIT patterns from $P^+ \cup P^{\pm}_{\text{refined}}$ to cover all pairs in $\mathcal{U}_{\text{permit}}$. In this example, patterns ϕ_5 and ϕ_6 together cover every PERMIT pair with fewer rules than any other combination, yielding a final minimal set of PERMIT rules $\{\phi_5, \phi_6\}$. However, ϕ_5 also matches pairs $\langle u_4, r_4\rangle$ and $\langle u_8, r_8\rangle$ with observed DENY decisions. Without an explicit DENY rule, these pairs would be incorrectly permitted by ϕ_5. Accordingly,



(a) Selecting Minimal Set of PERMIT Patterns



(b) Selecting Minimal Set of DENY Patterns

Figure 3: Policy Optimization in Two Stages, Selecting Minimal Set of Final PERMIT and DENY Patterns.

in Figure 3b, we collect these newly exposed DENY pairs into $\mathcal{U}_{\text{deny}}$ and solve a second set cover problem over P^-. In this example, ϕ_3 covers all pairs in $\mathcal{U}_{\text{deny}}$; thus, our final minimal ReBAC policy becomes { $\langle \phi_5, \texttt{PERMIT} \rangle$, $\langle \phi_6, \texttt{PERMIT} \rangle$, $\langle \phi_3, \texttt{DENY} \rangle$ }. User-resource pairs not matched by these rules, i.e., $\langle u_2, r_2 \rangle$ and $\langle u_9, r_9 \rangle$, will be implicitly denied by default.

4 Experimental Evaluation

In this section, we present the experimental evaluation of the proposed mining algorithm, which extracts both PERMIT and DENY authorization rules for ReBAC policies. The evaluation is structured into two main parts. First, we assess the correctness, conciseness, and performance of our proposed algorithm on two datasets with distinct policy configurations: one containing only PERMIT authorization rules and the other containing both PERMIT and DENY rules. Second, we perform an empirical feasibility analysis to explore whether mixed policies requiring both PERMIT and DENY rules can be effectively represented using only PERMIT rules. This dual focus allows us to thoroughly validate the algorithm's effectiveness and investigate the necessity of explicit DENY rule mining in ReBAC policy generation. Finally, we conduct an additional experiment in which we vary the number of nodes in the system graph to measure how the total runtime scales with increasing graph size.

All experiments were conducted on Google Colab, which provided a runtime environment equipped with an Intel® Xeon® CPU @ 2.20 GHz, 12 GB of RAM, and Python 3.11.11 running in a containerized environment.

4.1 Datasets and Mining Evaluation Cases

Dataset Construction. We utilize two distinct system graphs to evaluate our policy mining approach. The first dataset models a medical records system taken from [17]. We also synthesize online social network graphs containing relationships such as friend, colleague, and family. Since such relationship are inherently symmetric, we treat edges as bidirectional. We synthesize networks using a controlled random process. Specifically, each pair of nodes has a 1% chance of forming an edge. If an edge is created, one of relationship labels L is chosen according to the specified probabilities of them. The key statistics for both datasets, including the number of users U, resources R, unique relationship labels L, and edges E are summarized in table 1.

Table 1: Summary of Policy Configurations

Policy	$\lvert U \rvert$	$\lvert R \rvert$	$\lvert L \rvert$	$\lvert E \rvert$
Medical Records System	30	6	8	52
Online Social Network	100	100	3	55

Generating Mining Evaluation Cases. Each experiment is framed as a mining evaluation case (MEC) [18], defined as $\langle G, \rho_T \rangle$. For a given G and ground truth policy ρ_T, we produce the set of authorizations $A = \lambda(G, \rho_T)$, which represents the computed PERMIT/DENY decisions for all relevant (u, r) pairs. We then provide (G, A) as input to our proposed miner algorithm. The miner outputs a mined policy ρ_m, which we evaluate compared to ρ_T.

Ground Truth Policies. We generate two variants of ground truth policies ρ_T for each dataset: one with only PERMIT rules (*PERMIT-Only*), and another with both PERMIT and DENY rules (*PERMIT-and-DENY*). Corresponding to each variant, we produce two types of MECs: a strong MEC and a weak MEC [18]. Strong MECs are constructed by ensuring that ρ_T is both minimal and maximal with respect to G; there are no alternative policies with fewer or additional rules that are still semantically equivalent on G. In contrast, weak MECs arise in cases where multiple semantically equivalent but syntactically different policies can represent the same authorizations.

4.2 Evaluation Metrics

Correctness. We consider two particular correctness metrics: *semantic similarity* and *syntactic equivalence*. These metrics evaluate the quality of a mined policy ρ_m in comparison to the target policy ρ_T, as follows:

Definition 4.1 (Semantic Similarity). Let $P(\rho_T)$ denote the set of permitted $\langle u, r \rangle$ pairs under the target policy ρ_T, and $P(\rho_m)$ the set of permitted pairs under the mined policy ρ_m. The *semantic similarity* between ρ_m and ρ_T is defined as:

$$\text{SemanticSimilarity}(\rho_m, \rho_T) = \frac{|P(\rho_m) \cap P(\rho_T)|}{|P(\rho_T)|}$$

A SemSim score of 1.0 indicates perfect semantic equivalence between the two policies.

Definition 4.2 (Syntactic Equivalence). Let ρ_T be the target policy and ρ_m the mined policy. The *syntactic equivalence* between ρ_m and ρ_T is satisfied if:

$$\rho_m = \rho_T$$

This condition requires that ρ_m and ρ_T are not only semantically identical but also the specific rules and their patterns match exactly.

For weak MECs, we require only perfect semantic similarity (SemSim = 1.0) to ensure correctness. In contrast, for strong MECs, correctness requires both perfect semantic similarity and syntactic equivalence ($\rho_m = \rho_T$).

Conciseness. We measure policy complexity and conciseness using *weighted structural complexity (WSC)* [7, 22]. Since we have a single action, WSC depends solely on the lengths of relationship patterns in a rule. The WSC of a policy is computed as:

$$\text{WSC}(\rho) = \sum_{\phi \in \rho} |\phi|$$

Lower WSC values indicate simpler, more concise policies. We compare WSC_{orig} (for ρ_T) and WSC_{mined} (for ρ_m) to assess whether the mining process preserves or improves policy simplicity. Additionally, we report the number of PERMIT and DENY rules in both the original and mined policies. Specifically, $|\rho^+_{orig}|$ and $|\rho^-_{orig}|$ represent the number of PERMIT and DENY rules in the original ReBAC policies, while $|\rho^+_{mined}|$ and $|\rho^-_{mined}|$ denote the corresponding counts in the mined ReBAC policies.

Performance. We assess the runtime performance of our prototype implementation of the mining algorithm to evaluate its scalability and efficiency.

These metrics collectively offer a comprehensive view of correctness, conciseness, and performance, ensuring the mined policies are both accurate and practical for real-world applications.

4.3 Mining Performance

We begin our evaluation with *PERMIT-Only* policies, which consist exclusively of PERMIT rules. Such policies are both conceptually simpler and more commonly encountered, making them an ideal starting point. While *PERMIT-Only* policies serve as a valuable initial benchmark, they cannot fully represent real-world access control scenarios, many of which require explicit denials to handle exceptions. Consequently, we extend our analysis to *PERMIT-and-DENY* policies that integrate both PERMIT and DENY rules.

In Table 2, we present the results of our proposed algorithm under both the *PERMIT-Only* and *PERMIT-and-DENY* policy variants within the Medical Records System setting. Likewise, Table 3 reports the corresponding outcomes for the *PERMIT-Only* and *PERMIT-and-DENY* variants in the online social network scenario, providing a comparative perspective of our proposed algorithm across various metrics.

The experimental results demonstrate the effectiveness of the proposed algorithm across both *PERMIT-Only* and *PERMIT-and-DENY* policies.

Considering correctness, in all weak MEC scenarios, the mined policies achieve perfect semantic similarity, 1.0, accurately reproducing the target authorizations. For strong MECs, syntactic equivalence is also satisfied, meaning the mined policy not only matches the authorization decisions but also the exact rule patterns of the ground truth. This demonstrates the our proposed algorithm's effectiveness in both weak and strong variants.

In terms of conciseness, the mined policies consistently exhibit $WSC_{mined} \leq WSC_{orig}$ across all experiments. In particular, in weak MECs where multiple semantically equivalent representations exist, our proposed algorithm is able to select a minimal set of rules that cover the same access decisions as the original policy.

In summary, the experimental findings indicate that the proposed approach produces accurate, concise, and minimal ReBAC policies.

4.4 Necessity of DENY Rules

While the previous experiments validated our algorithm's ability to mine accurate and concise ReBAC policies under both *PERMIT-Only* and *PERMIT-and-DENY* scenarios, we now turn our attention to a more fundamental question: Can policies that inherently require explicit DENY rules be effectively represented using only PERMIT rules? In other words, if a policy's constraints depend on deliberate denials, rather than default non-permissions, can PERMIT rules alone suffice to capture its complete authorization semantics?

To investigate this, we conducted a large-scale feasibility study simulating policies that mix PERMIT and DENY rules, and then tested whether their resulting authorization configurations could be reproduced by a policy composed solely of PERMIT rules.

In our analysis, we systematically varied the complexity of the generated ReBAC policies by adjusting the number of PERMIT and DENY rules. Let P represent the number of PERMIT rules, and D represent the number of DENY rules. We considered every combination (P, D) where P and D each range from 1 to 10. This setup yielded 100 distinct policy configurations, starting from the simplest $(1, 1)$ configuration and extending up to the more complex $(10, 10)$ scenario.

For each generated scenario, we attempted to model the required permissions strictly using PERMIT rules alone. Formally, this was approached using our proposed methodology: the universe of authorized $\langle u, r \rangle$ pairs came from the original mixture of PERMIT and DENY rules, and we sought to cover this universe solely with PERMIT rules. If no collection of positive subsets could achieve an exact match of the required authorizations, we classified that scenario as *infeasible* under a *PERMIT-Only* policy representation.

To achieve statistically robust results, we did not rely on a single scenario per configuration. Instead, for each of the 100 (P, N) configurations, we generated 15,000 independent *trials*, each trial comprising 100 different random scenarios. In total, this amounted to 1,500,000 scenarios per configuration. For each trial, we recorded the number of infeasible scenarios, leveraging the Central Limit Theorem, these statistics provide stable population-level estimates of infeasibility rates. Figure 4 presents the histogram of infeasibility counts across 15,000 trials to provide a visual understanding of the infeasibility distribution.

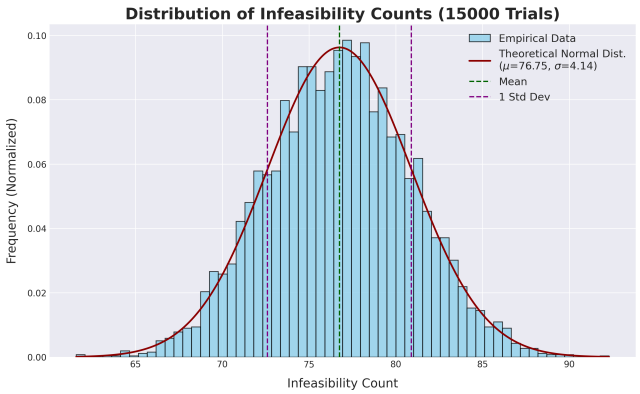


Figure 4: Histogram of Infeasible Scenarios Per Trial Across 15,000 Trials.

The results indicate that, on average, 77 out of 100 scenarios per trial were *infeasible*. In other words, when DENY rules were originally necessary to encode certain denials, attempts to model these same constraints with PERMIT rules alone were unsuccessful about 77% of the time. The standard deviation of 4.14 demonstrates low variability around the mean suggesting the results are robust and statistically stable.

These findings quantitatively confirm that DENY rules are essential mechanisms for capturing policies that rely on explicit denials. Without them, significant portions of the authorization space cannot be accurately reproduced.

Table 2: Comparison of the Proposed Algorithm on the Medical Records System Policy

MEC Variant	Number of Original Rules		Number of Mined Rules		WSC		Sem. Sim.
	$\lvert\rho^+_{orig}\rvert$	$\lvert\rho^-_{orig}\rvert$	$\lvert\rho^+_{mined}\rvert$	$\lvert\rho^-_{mined}\rvert$	WSC_{orig}	WSC_{mined}	
PERMIT-Only							
Weak	9	-	8	-	24	22	1
Strong	8	-	8	-	22	22	1
PERMIT-and-DENY							
Weak	9	4	8	3	33	30	1
Strong	8	3	8	3	30	30	1

Table 3: Comparison of the Proposed Algorithm on the Online Social Network Policy

MEC Variant	Number of Original Rules		Number of Mined Rules		WSC		Sem. Sim.
	$\lvert\rho^+_{orig}\rvert$	$\lvert\rho^-_{orig}\rvert$	$\lvert\rho^+_{mined}\rvert$	$\lvert\rho^-_{mined}\rvert$	WSC_{orig}	WSC_{mined}	
PERMIT-Only							
Weak	40	-	37	-	148	135	1
Strong	40	-	40	-	162	162	1
PERMIT-and-DENY							
Weak	40	10	35	8	186	158	1
Strong	40	10	40	10	197	197	1

4.5 Running Time Performance

In this subsection, we present two sets of experimental results on the runtime performance of our prototype implementation. First, we analyze how varying the number of `PERMIT` (P) and `DENY` (D) rules affects the total runtime. Second, we assess the scalability of our approach by measuring the runtime as the system graph grows in size (in terms of both nodes and edges). Initially, we recorded the total runtime for each (P, D) configuration over all conducted trials in the experiment discussed in Section 4.4. Figure 5 presents a 2D heatmap of the total runtime recorded across various (P, D) configurations. Notice how configurations with fewer `PERMIT` rules and more `DENY` rules (bottom-right corner) lead to lower overall runtimes, indicated by lighter shades.

From Figure 5, we can identify two interrelated patterns. First, as P increases (while D remains fixed), the total runtime tends to rise. This is likely due to a larger universe of user-resource pairs generated by the additional `PERMIT` rules, which in turn makes the coverage problem more complex and time-consuming to solve.

Second, when P is held constant and D increases, the total runtime generally decreases or stabilizes. In this case, the growing number of `DENY` rules prunes the candidate universe by introducing early conflicts, effectively reducing the size of the set that must be covered.

Together, these patterns underscore the importance of both P and D in shaping the complexity of the ReBAC policy inference process. A higher P expands the universe, increasing runtime, while a higher D within a fixed P can counterbalance that growth by trimming the universe and thus streamlining the search.

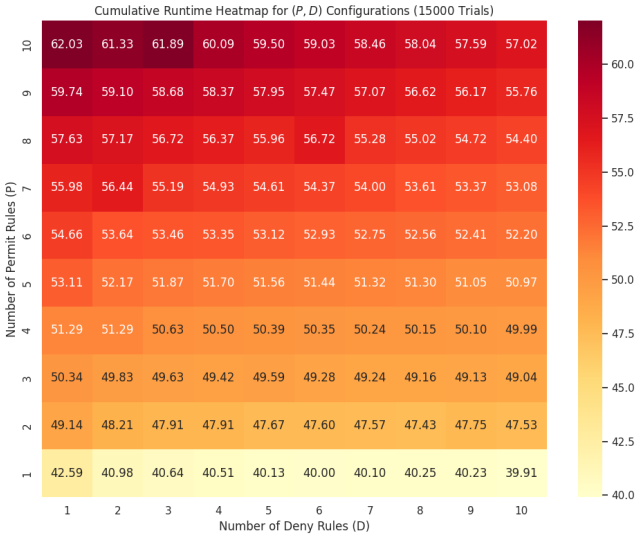


Figure 5: Cumulative Runtime Heatmap for (P, D) Configurations. The Horizontal Axis Represents D (Number of `DENY` Rules), and the Vertical Axis Represents P (Number of `PERMIT` Rules). Each Cell Indicates the Cumulative Runtime (in Seconds) for 15,000 Trials of the Corresponding (P, D) Configuration. Darker Shades Correspond to Higher Runtimes.

To further assess the scalability of our approach, we conducted an additional experiment varying the size of the system graph

of online social network dataset from 100 to 600 nodes. In each configuration, we recorded the total runtime (in seconds) for our mining algorithm. Figure 6 visualizes these results, where the x-axis represents the node size, the y-axis indicates the total runtime.

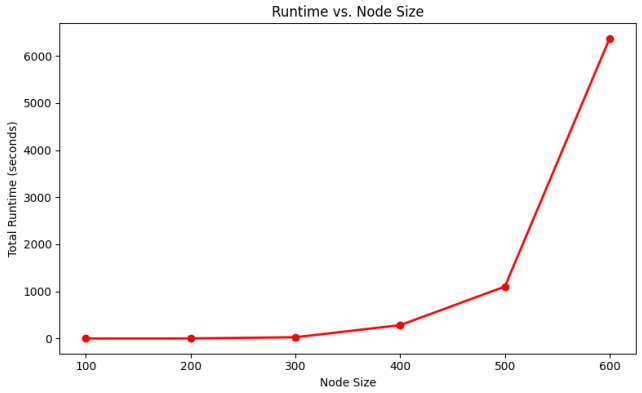


Figure 6: Runtime Performance as System Graph Size Increase.

As we mentioned, the overall time complexity of our proposed mining algorithm is $O(|L|^k \cdot |V|^2 \log |V| + |V| \cdot d^k)$ which, under the assumption of constant k and low graph density (with d remaining relatively small due to the 0.01 edge probability), simplifies to an effective scaling of $(|V| \cdot L^2 \log |V|)$. This analysis justifies our empirical observation: as the system graph grows, both in terms of nodes and edges, the NP-hard complexity of the policy-mining problem becomes more pronounced, leading to longer runtimes.

5 Related Work

Relationship-based access control (ReBAC) has received attention for its ability to incorporate social and organizational relationships into access control decisions [15]. Recent studies have aimed to automate the derivation of ReBAC policies from given lower-level authorizations and entity relationships. Bui and Stoller proposed a greedy heuristic approach to mining ReBAC policies, emphasizing efficiency and practicality over optimality [7]. They later extended this work with a grammar-based evolutionary algorithm, introducing genetic programming to explore policies while optimizing for rule minimality and conciseness [9]. They further addressed scenarios involving incomplete and noisy permission data [8]. The authors have also proposed a work [6], where they proposed FSSEA*, a feature selection-enhanced evolutionary algorithm. The researchers have also proposed decision tree-based algorithms for ReBAC policy mining, which support richer constructs such as negation [4]. They further extended their approach to handle scenarios involving unknown attribute values [5]. Iyer and Masoumzadeh proposed an optimal algorithm for ReBAC mining that utilizes rule mining and frequent graph-based pattern mining concepts within evolving systems, effectively addressing the challenges posed by dynamic environments [17]. In a complementary approach, they have extended ReBAC policy mining to address scenarios where authorization information is incomplete or fully exploring it is infeasible. This method leverages active learning techniques to infer ReBAC policies directly from authorization decisions observed in black-box systems [19]. Chakraborty and Sandhu investigated whether a ReBAC policy can be formulated from relationship graphs and existing authorization sets. Their work introduced a formal feasibility framework and proposed an algorithm to address this challenge [10].

Negative authorization has been supported in various access control policy models including role-based access control [3], attribute-based access control [1], and ReBAC [23]. Despite the recognized importance of negative authorizations, systematic methods for mining such rules have been relatively limited. In the attribute-based access control domain, Iyer and Masoumzadeh introduced a pioneering algorithm to mine both positive and negative rules using an extended version of the PRISM rule-mining framework [16]. Their work demonstrated that incorporating explicit negative authorizations leads to more concise and semantically accurate policies. However, no comparable systematic approach has been proposed for ReBAC. Existing ReBAC mining techniques focus primarily on discovering positive authorization patterns and rely on implicit deny-all semantics, leaving a critical gap in representing exceptions or special restrictions. Our work addresses this gap by introducing the first systematic method for mining both positive and negative authorization rules in ReBAC, enabling the derivation of policies that more accurately capture the full spectrum of observed access control decisions.

6 Conclusions

In this paper, we introduced a novel method for mining relationship-based access control (ReBAC) policies that incorporate both positive (`PERMIT`) and negative (`DENY`) authorization rules. Unlike prior approaches that rely solely on deny-by-default semantics, our technique systematically discovers explicit `DENY` rules. This extension offers greater expressive power for capturing exceptions and prohibitions, ultimately enabling ReBAC policies to better align with real-world access control needs.

We formulated the ReBAC policy mining problem with `PERMIT` and `DENY` rules, proved its NP-hardness, and devised a two-stage solution modeled by the set cover problem. Our experimental evaluations demonstrate that our prototype implementation produces minimal, correct, and concise policies across various scenarios. Further, our feasibility analysis showed that a significant portion of policies inherently requiring explicit `DENY` rules cannot be accurately represented by `PERMIT` rules alone, underscoring the importance of negative authorizations in complex environments.

Our framework supports system administrators in capturing intricate exceptions that deny-by-default alone fails to handle, thus enhancing the policy's precision, and manageability. While we focused on *consistency* and *minimality*, additional objectives such as *rule simplicity* (e.g., limiting pattern lengths) could be integrated into the optimization via a weighted set cover approach. In that extension, each rule would be assigned a cost proportional to its complexity, allowing the algorithm to optimize for simpler rule structures as well as minimality. We leave this potential enhancement for future work.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and helpful suggestions. This material is based upon work supported by the National Science Foundation under Grant No. 2047623.

References

[1] 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[2] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. 2017. Classifying and comparing attribute-based and relationship-based access control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 59–70.

[3] Mohammad A Al-Kahtani and Ravi Sandhu. 2004. Rule-based RBAC with negative authorization. In *20th Annual Computer Security Applications Conference*. IEEE, 405–415.

[4] Thang Bui and Scott D Stoller. 2020. A decision tree learning approach for mining relationship-based access control policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*. 167–178.

[5] Thang Bui and Scott D Stoller. 2020. Learning attribute-based and relationship-based access control policies with unknown values. In *International Conference on Information Systems Security*. Springer, 23–44.

[6] Thang Bui, Scott D Stoller, and Hieu Le. 2019. Efficient and extensible policy mining for relationship-based access control. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. 161–172.

[7] Thang Bui, Scott D Stoller, and Jiajie Li. 2017. Mining relationship-based access control policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. 239–246.

[8] Thang Bui, Scott D Stoller, and Jiajie Li. 2018. Mining relationship-based access control policies from incomplete and noisy data. In *International Symposium on Foundations and Practice of Security*. Springer, 267–284.

[9] Thang Bui, Scott D Stoller, and Jiajie Li. 2019. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security* 80 (2019), 317–333.

[10] Shuvra Chakraborty and Ravi Sandhu. 2021. Formal analysis of rebac policy mining feasibility. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*. 197–207.

[11] Yuan Cheng, Jaehong Park, and Ravi Sandhu. 2012. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. IEEE, 646–655.

[12] Vasek Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of operations research* 4, 3 (1979), 233–235.

[13] Jason Crampton and James Sellwood. 2014. Path conditions and principal matching: a new approach to access control. In *Proceedings of the 19th ACM symposium on Access control models and technologies*. 187–198.

[14] Irit Dinur and David Steurer. 2014. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 624–633.

[15] Philip WL Fong. 2011. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*. 191–202.

[16] Padmavathi Iyer and Amirreza Masoumzadeh. 2018. Mining positive and negative attribute-based access control policy rules. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. 161–172.

[17] Padmavathi Iyer and Amirreza Masoumzadeh. 2019. Generalized mining of relationship-based access control policies in evolving systems. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. 135–140.

[18] Padmavathi Iyer and Amirreza Masoumzadeh. 2022. Effective evaluation of relationship-based access control policy mining. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*. 127–138.

[19] Padmavathi Iyer and Amirreza Masoumzadeh. 2022. Learning relationship-based access control policies from black-box systems. *ACM Transactions on Privacy and Security* 25, 3 (2022), 1–36.

[20] Padmavathi Iyer, Amirreza Masoumzadeh, and Paliath Narendran. 2022. On the Expressive Power of Negated Conditions and Negative Authorizations in Access Control Models. *Computers & Security* 116 (2022), 102586.

[21] Srdjan Marinovic, Naranker Dulay, and Morris Sloman. 2014. Rumpole: An introspective break-glass access control language. *ACM Transactions on Information and System Security (TISSEC)* 17, 1 (2014), 1–32.

[22] Ian Molloy, Ninghui Li, Yuan Qi, Jorge Lobo, and Luke Dickens. 2010. Mining roles with noisy data. In *Proceedings of the 15th ACM symposium on Access control models and technologies*. 45–54.

[23] Edelmira Pasarella and Jorge Lobo. 2017. A datalog framework for modeling relationship-based access control policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. 91–102.

[24] Syed Zain R Rizvi, Philip WL Fong, Jason Crampton, and James Sellwood. 2015. Relationship-based access control for an open-source medical records system. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*. 113–124.