An Automatic Attribute Based Access Control Policy Extraction from Access Logs

Leila Karimi, Student Member, IEEE, Maryam Aldairi, Student Member, IEEE, James Joshi, Senior Member, IEEE, and Mai Abdelhakim, Member, IEEE

Abstract—With the rapid advances in computing and information technologies, traditional access control models have become inadequate in terms of capturing fine-grained, and expressive security requirements of newly emerging applications. An attribute-based access control (ABAC) model provides a more flexible approach for addressing the authorization needs of complex and dynamic systems. While organizations are interested in employing newer authorization models, migrating to such models pose as a significant challenge. Many large-scale businesses need to grant authorization to their user populations that are potentially distributed across disparate and heterogeneous computing environments. Each of these computing environments may have its own access control model. The manual development of a single policy framework for an entire organization is tedious, costly, and error-prone. In this paper, we present a methodology for automatically learning ABAC policy rules from access logs of a system to simplify the policy development process. The proposed approach employs an unsupervised learning-based algorithm for detecting patterns in access logs and extracting ABAC authorization rules from these patterns. In addition, we present two policy improvement algorithms, including rule pruning and policy refinement algorithms to generate a higher quality mined policy. Finally, we implement a prototype of the proposed approach to demonstrate its feasibility.

Index Terms—Access Control, Attribute Based Access Control, Policy Mining, Policy Engineering, Machine Learning, Clustering.

1 INTRODUCTION

Access control systems are a critical component of an information system that protect information resources from unauthorized accesses. Various access control models and approaches have been proposed in the literature including Discretionary Access Control (DAC) [1, 2], Mandatory Access Control (MAC) [3, 4], and Role-Based Access Control (RBAC) [5]. However, with the rapid advances in newer computing and information technologies (e.g., social networks, Internet of Things (IoT), cloud/edge computing, etc.), existing access control (AC) models have become inadequate in providing flexible and expressive authorization services [6]. For example, a healthcare environment requires a more expressive AC model that meets the needs of patients, health care providers as well as other stakeholders in the health care ecosystem [7, 8]. Attribute Based Access Control (ABAC) models present a promising approach that addresses newer challenges in emerging applications [9]. An ABAC approach grants access rights to users based on attributes of entities in the system (i.e., user attributes, object attributes, and environmental conditions) and a set of authorization rules.

Although organizations and developers are interested in employing the next generation of AC models, adopting such policy frameworks pose a significant challenge. Many large organizations need to grant authorization to their vast user populations that are highly distributed across disparate computing environments, including legacy systems. Each of these computing environments may have its own AC model. The manual development of a single policy for the entire organization is tedious and error-prone. Policy Mining techniques have been proposed in the literature to address such challenges to help organizations cut the cost, time, and error of policy development/management. Policy mining algorithms ease the migration to more recent authorization models by completely (or partially) automating the development of AC policies.

Policy mining techniques were first introduced for developing RBAC policies. Kuhlmann et al. coined the term “role mining” to refer to a data mining approach that constructs roles from a given permission assignment dataset [10]; this work was followed by various role mining techniques [11, 12, 13]. Although the proposed approaches are beneficial in developing optimal sets of roles, they are not applicable in extracting ABAC policies.

Xu and Stoller were the first to study the problem of mining ABAC policies from given access control matrices or logs [14, 15]. Following that, several researchers have investigated various ABAC policy mining techniques [16, 17, 18]. However, these studies suffer from several limitations:

First of all, the earlier works do not support mining authorization rules with negative filters. An ABAC policy rule can comprise of a set of positive and negative filters. Negative filters are useful in scenarios when an exception needs to be expressed. For example, a healthcare provider can express the following rule using a negative attribute filter: “A nurse can read a patient’s record except for payment purposes.” Using negative filters in rule expressions results
in a more concise authorization policy (Section 5).

Second, some proposed approaches such as in [15], [17] are unable to mine a high-quality policy when the given access log is not complete in the sense that every possible combination of attribute values is not included in the access log (Section 4).

Third, the proposed approaches are unable to mine a policy from noisy access logs containing over-assignments and under-assignments [16], [18]. Having noisy access records is a common problem in evolving domains such as IoT or social networks [19]. It is essential that an ABAC policy miner should be capable of handling a reasonable amount of noise to be applicable to real-world domains.

Last but not least, the existing approaches do not include techniques for improving the mined policy after the first round of extraction. In addition, in scenarios where the authorization policies may change over time (such as in social networks with addition and removal of various applications), these approaches do not provide any guidelines for adjusting the policy. This makes practical deployment of these approaches not useful.

Furthermore, none of the previous work addresses these issues in an integrated way. In this paper, we propose a machine learning based ABAC policy mining approach to address these challenges. To summarize, the primary contributions of this paper are as follows:

1) We propose an unsupervised learning approach to extract ABAC policy rules that contain both positive and negative attribute filters as well as positive and negative relation conditions.

2) The proposed policy mining approach is practical even with an incomplete set of access logs and in the presence of noise.

3) As part of the unsupervised learning based approach, we propose the rule pruning and policy refinement algorithms to enhance the quality of the mined policy and to ease its maintenance.

4) We propose a policy quality metric based on policy correctness and conciseness to be able to compare different sets of mined policy rules and to select the best one based on given criteria.

5) We implement a prototype of the proposed model and evaluate it using various ABAC policies to show its efficiency and effectiveness.

To the best of our knowledge, our proposed approach is the first unsupervised learning based ABAC policy mining method that can be used to extract both positive and negative attribute and relationship filters and its efficiency is independent of the overall complexity of access control policy including the number of rules in the policy and the number of filters/relations in each rule.

The rest of the paper is organized as follows. In Section 2, we overview the ABAC model and its policy language as well as an unsupervised learning algorithm. In Section 3, we define the ABAC policy extraction problem, discuss the related challenges, and introduce the metrics for evaluating the extracted policy. In Section 4, we present the proposed approach. In Section 5, we present the evaluation of the proposed approach on various sets of policies. We present the related work in Section 6 and the conclusions and future work in Section 8.

2 Preliminaries

In this section, we overview ABAC, the ABAC policy language, and the unsupervised learning algorithm.

2.1 ABAC Model

In 2013, NIST published a “Guide to ABAC Definition and Consideration” [9], according to which, “the ABAC engine can make an access control decision based on the assigned attributes of the requestor, the assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.” Throughout the paper, we use user attributes, object attributes, and session attributes to refer to the attributes of the requester, attributes of the object, and the environmental attributes/conditions, respectively.

Accordingly, U, O, S, OP are sets of users, objects, sessions, and operations in a system and user attributes (A_u), object attributes (A_o), and session attributes (A_s) are mappings of subject attributes, object attributes, and environmental attributes as defined in the NIST Guide [9]. E = U ∪ O ∪ S and A = A_u ∪ A_o ∪ A_s, are the sets of all entities and all attributes in the system, respectively.

Definition 1. (Attribute Range). Given an attribute a ∈ A, the attribute range V_a is the set of all valid values for a in the system.

Definition 2. (Attribute Function). Given an entity e ∈ E, an attribute function f_{a,e} is a function that maps an entity to a specific value from the attribute range. Specifically, f_{a,e}(e, a) returns the value of attribute a for entity e.

Example 1. f_{a,e}(John, position) = faculty indicates that the value of attribute position for user John is faculty.

Example 2. f_{a,e}(dep1, crs) = {cs101, cs601, cs602} represents that the value of attribute crs for object dep1 is a set {cs101, cs601, cs602}.

Each attribute in the system can be a single-valued (atomic) or multi-valued (set). In Example 1 position is a single-valued attribute while crs is a multi-valued attribute in Example 2. For simplicity, we consider only atomic attributes in this work.

Attribute filters are used to denote the sets of users, objects, and sessions to which an authorization rule applies.

Definition 3. (Attribute Filter). An attribute filter is defined as a set of tuples \( F = \{(a, v!|v) | a \in A \land v \in V_a\} \). Here \((a, v)\) is a positive attribute filter tuple that indicates a has value v, and \((a, !v)\) is a negative attribute filter tuple that indicates a has any value in its range except v.

Example 3. Tuple \( <\text{label}, \text{top-secret}> \) points to all entities in the system that do not have “top-secret” as their security label “label”.

Definition 4. (Attribute Filter Satisfaction). An entity e ∈ E satisfies an attribute filter \( F \), denoted as e \( \models F \), if

\[
\forall(a_i, v_i) \in F: f_{a_i,e}(e, a_i) = v_i \land \forall(a_i, !v_i) \in F: f_{a_i,e}(e, a_i) \neq v_i.
\]
Definition 6 (Relation Condition). A relation condition is defined as a set of tuples \( R = \{ (a, b|l) | a, b \in A \land a \neq b \}. \) Here \((a, b)\) is a positive relation condition tuple that indicates \( a \) and \( b \) have the same values, and \((a, b)\) is a negative relation condition tuple that indicates \( a \) and \( b \) do not have the same values.

A relation is used in a rule to denote the equality condition between two attributes of users, objects, or sessions.

Definition 6 (Relation Condition Satisfaction). An entity \( e \in E \) satisfies a relation condition \( R \), denoted as \( e \models R \), if
\[
\forall (a_i, b_i) \in R: f_a(e, a_i) = f_a(e, b_i)
\]
\[
\forall (a_i, b_i) \in R: f_a(e, a_i) \neq f_a(e, b_i).
\]

Definition 7 (Access Request). An access request is a tuple \( q = \langle u, o, s, op \rangle \) where user \( u \in U \) sends a request to the system to perform operation \( op \in OP \) on object \( o \in O \) in session \( s \in S \).

Definition 8 (Authorization Tuple/Access Log). An authorization tuple is a tuple \( t = \langle q, d \rangle \) containing decision \( d \) made by the access control system for request \( q \). An Access Log \( L \) is a set of such tuples.

The decision \( d \) of an authorization tuple can be permit or deny. The authorization tuple with permit decision means that user \( u \) can perform an operation \( op \) on an object \( o \) in session \( s \). The authorization tuple with deny decision means that user \( u \) cannot perform operation \( op \) on object \( o \) in session \( s \).

Access log is a union of Positive Access Log, \( L^+ \), and Negative Access Log, \( L^- \), where:
\[
L^+ = \{ \langle q, d \rangle | \langle q, d \rangle \in L \land d = \text{permit} \},
\]
and
\[
L^- = \{ \langle q, d \rangle | \langle q, d \rangle \in L \land d = \text{deny} \}.
\]

Definition 9 (ABAC Rule). An access rule \( \rho \) is a tuple \( \langle F, R, op \rangle \), where \( F \) is an attribute filter, \( R \) is a relation condition, and \( op \) is an operation.

Example 5. Consider rule \( \rho_1 = \{ ((\text{position}, \text{student}), (\text{location}, \text{campus}), (\text{type}, \text{article})), (\text{dept}_a, \text{dept}_b) \}, \) read. It can be interpreted as “A student can read an article if he/she is on campus and his/her department matches the department of the article”.

Definition 10 (Rule Satisfaction) An access request with \( q = \langle u, o, s, op \rangle \) is said to satisfy a rule \( \rho \), denoted as \( q \models \rho \), if
\[
 q = \langle u, o, s, op \rangle \models F \land \langle u, o, s \rangle \models R \land op \models op \rho.
\]

Definition 11 (ABAC Policy). An ABAC policy is a tuple \( \pi = \langle E, OP, \Lambda, f_a,e, \mathcal{P} \rangle \) where \( E, OP, \Lambda \), and \( \mathcal{P} \) are sets of entities, operations, attributes, and ABAC rules in the system and \( f_a,e \) is the attribute function.

Definition 12 (ABAC Policy Decision). The decision of an ABAC policy \( \pi \) for an access request \( q \) denoted as \( d_\pi(q) \) is permit if:
\[
\exists \rho \in \pi : q \models \rho
\]
otherwise, the decision is deny.

If an access request satisfies a rule of the access control policy, then the decision of the system for such access request is permit. If the access request does not satisfy any rule in the access control policy then the decision of the system for such access request is deny.

TABLE I summarizes the notations used in this paper.

### 2.2 Unsupervised Learning Algorithm

Unsupervised learning algorithms try to infer a function that describes the structure of unlabeled data. They are useful when no or very few labeled data is available. We leverage such methods for extracting ABAC policies from access logs.

In particular, given a set of authorization tuples, we employ an unsupervised learning approach to mine and extract an ABAC policy that has high quality. An unsupervised learning approach is suitable because there is no labeled data available for desired ABAC rules. ABAC policy extraction, in this case, can be considered as a mapping between authorization tuples to a set of clusters that are representative of the desired ABAC rules. Such a mapping can be expressed as a function, \( h : X \rightarrow \mathcal{Y} \), where:

1) \( X \) is a set of authorization tuples (i.e., access log).
2) \( \mathcal{Y} \) is a set of numbered labels (i.e., cluster labels, each cluster corresponding to a rule of the ABAC policy \( \pi \)).

The goal is then to learn the function \( h \) with low clustering error and mine the desired policy with high quality.

### 3 Problem Definition

#### 3.1 ABAC Policy Extraction Problem

Although organizations are interested in employing an ABAC model, adopting it is a big challenge for them. The manual development of such a policy is tedious and error-prone. Policy Mining techniques have been proposed to address such challenges in order to reduce the cost, time, and error of policy development/maintenance. ABAC policy mining algorithms ease the migration to the ABAC framework by completely (or partially) automating the development of ABAC policy rules.

The primary input to a policy mining algorithm is the log of authorization decisions in the system. The log indicates authorization decision (i.e., permit or deny) for any given access request by a user of the system. For ABAC policy mining, such a log is accompanied by attributes of entities involved in the log entries. The goal of a policy mining algorithm is to extract ABAC policy rules from access logs that have high quality with respect to some quality metrics (e.g., policy size and correctness).

We define the ABAC policy extraction problem formally as follows:
sets of users, objects, sessions, and operations

$A_u$, $A_o$, and $A_s$

sets of user attributes, object attributes, and session attributes

$E = U \cup O \cup S$

set of all entities

$A = A_u \cup A_o \cup A_s$

set of all attributes

$V_a$

Attribute Range: set of all valid values for $a \in A$

$f_a(x; e, a)$

Attribute Function: a function that maps an entity $e \in E$ to a value from $V_a$

$\mathcal{T} = \{(a, v) | \forall v \in V_a\}$

Attribute Filter

$\mathcal{R} = \{(a, b) | a, b \in A \land a \neq b \land V_a = V_b\}$

Relation Condition

$q = (u, o, s, op)$

Access Request

$L$

Access Log, set of authorization tuples

$L^+ = \{(q, d) | (q, d) \in L \land d = \text{permit}\}$

Positive Access Log

$L^- = \{(q, d) | (q, d) \in L \land d = \text{deny}\}$

Negative Access Log

$\rho = \langle T, R, op, d \rangle$

ABAC Rule

$\pi = \langle E, OP, A, f_a, e, \rho \rangle$

ABAC Policy

$d_a(q)$

the decision of an ABAC policy $\pi$ for an access request $q$

$TP_{n \epsilon \mathcal{L}}, FP_{n \epsilon \mathcal{L}}, TN_{n \epsilon \mathcal{L}}$, and $FN_{n \epsilon \mathcal{L}}$

Relative True Positive, False Positive, True Negative, and False Negative Rates

$\text{ACC}_{n \epsilon \mathcal{L}}$

Relative Accuracy Rate

$\text{F-score}_{n \epsilon \mathcal{L}}$

Relative F-score

$\text{WSC}(\pi)$

Weighted Structural Complexity of policy $\pi$

$Q_w$

Policy Quality Metric

### Definition 13. (ABAC Policy Extraction Problem)

Let $I = \langle E, OP, A, f_a, e, \mathcal{L} \rangle$, where the components are as defined earlier, then the **ABAC policy extraction problem** is to find a set of rules $\mathcal{R}$ such that the ABAC policy $\pi = \langle E, OP, A, f_a, e, \mathcal{R} \rangle$ has high quality with respect to $\mathcal{L}$.

### 3.2 Challenges and Requirements

For an ABAC policy extraction approach to be applicable to a wide range of real-world scenarios, we identify the following challenges and requirements:

1. **Correctness of Mined Policy**: The mined policy must be consistent with input authorization log in a way that the access decision of the mined policy must result in the same access decision of the log entry. An inconsistent extracted policy may result in situations in which an originally authorized access is denied (more restrictive) or originally unauthorized access is permitted (less restrictive) by the system.

2. **Complexity of Mined Policy**: The mining algorithm should endeavor to extracting a policy that is as concise as possible. Since the policy rules need to be manipulated by human administrators, the more concise they are, the more manageable and easier to interpret they would be. In addition, succinct rules are desirable as they are easier to audit and manage.

3. **Negative Attribute Filters**: The ABAC policy mining solution should support both positive and negative attribute filters which will result in more concise and comprehensible mined policy.

4. **Relation Conditions**: The solution should support the extraction of relation conditions for policy mining in order to generate more concise and comprehensible mined policy.

5. **Sparse Logs**: In real-world, the access log input to the policy mining algorithm may be sparse, representing only a small fraction of all possible access requests. The policy mining algorithm must be able to extract useful rules even from a sparse log.

6. **Mining Negative Authorization Rules**: An ABAC policy can contain both positive and negative rules which permit or deny access requests, respectively. The use of negative rules is helpful in situations where specifying exceptions to more general rules is important. Being able to define negative policy rules would result in more concise ABAC policy. Thus, the policy mining algorithm should be able to extract both positive and negative authorization rules.

7. **Noisy Authorization Log**: In the real world and with complex and dynamic information systems, it is possible to have a noisy authorization log consisting of over-assignments and under-assignments. These issues happen either due to a wrong configuration of the original authorization system or improper policy updates by administrators. The policy mining algorithm should be capable of extracting meaningful rules even with the presence of an acceptable amount of noise in the input access log.

8. **Dynamic and Evolving Policies**: The proposed method should employ a mechanism to support the dynam- icty of the information systems and their authorization policies and ease the maintenance of evolving systems.

Our proposed approach addresses all the requirements except the sixth one. Table 2 shows the challenges that are addressed by our proposed approach and how it improves upon the state-of-the-art policy mining techniques.

### 3.3 Evaluation Metrics

One of the main metrics for evaluating the quality of an extracted policy is how accurately it matches the original policy. That means the decisions made by the extracted policy for a set of access requests should be similar to the
decisions made by the original policy for that set of requests. As an example, if the decision of the original policy for an access request \( q \) is \( \text{permit} \), then the decision of the mined policy for the same access request must be \( \text{permit} \) as well. If the mined policy denies the same access request, then we record this authorization tuple as a False Negative. We define \( \text{Relative True Positive} \), \( \text{Relative False Positive} \), \( \text{Relative True Negative} \), and \( \text{Relative False Negative} \) rates, respectively, as follows:

### Definition 14. (Relative True Positive Rate)
Given an access log \( L \) and an ABAC policy \( \pi \), the relative true positive rate of \( \pi \) regarding \( L \) denoted as \( TP_{\pi|L} \) is the portion of positive access logs that the decision of \( \pi \) for them is \( \text{permit} \):

\[
TP_{\pi|L} = \frac{\left| \{ (q,d) \in L^+ \mid d_\pi(q) = \text{permit} \} \right|}{\left| L^+ \right|}
\]

Here, \( |s| \) is the cardinality of set \( s \).

### Definition 15. (Relative False Positive Rate)
The relative false positive rate of \( \pi \) regarding \( L \) denoted as \( FP_{\pi|L} \) is the portion of negative access logs that the decision of \( \pi \) for them is \( \text{permit} \):

\[
FP_{\pi|L} = \frac{\left| \{ (q,d) \in L^- \mid d_\pi(q) = \text{permit} \} \right|}{\left| L^- \right|}
\]

Similarly, we calculate the relative true negative rate and false negative rate of \( \pi \) regarding \( L \) denoted as \( TN_{\pi|L} \) and \( FN_{\pi|L} \) as follows:

\[
TN_{\pi|L} = \frac{\left| \{ (q,d) \in L^- \mid d_\pi(q) = \text{deny} \} \right|}{\left| L^- \right|}
\]

\[
FN_{\pi|L} = \frac{\left| \{ (q,d) \in L^+ \mid d_\pi(q) = \text{deny} \} \right|}{\left| L^+ \right|}
\]

The relative precision and relative recall are calculated as follows:

\[
\text{Precision}_{\pi|L} = \frac{TP_{\pi|L}}{TP_{\pi|L} + FP_{\pi|L}}
\]

\[
\text{Recall}_{\pi|L} = \frac{TP_{\pi|L}}{TP_{\pi|L} + FN_{\pi|L}}
\]

The relative accuracy metric, \( ACC_{\pi|L} \), measures the accuracy of mined policy \( \pi \) with regards to the decisions made by the original policy indicated by \( L \) and is defined formally as follows:

\[
\text{Definition 16. (Relative Accuracy)} \quad \text{Given the relative true positive and negative rates, the relative accuracy of } \pi \text{ regarding } L \text{ denoted as } ACC_{\pi|L} \text{ is calculated as follows:}
\]

\[
ACC_{\pi|L} = \frac{TP_{\pi|L} + TN_{\pi|L}}{TP_{\pi|L} + TN_{\pi|L} + FP_{\pi|L} + FN_{\pi|L}}
\]

As accuracy may be misleading in unbalanced data sets [20] (which is very probable in case of access logs), we use relative F-score to better evaluate the mined policy:

\[
F\text{-score}_{\pi|L} = 2 \cdot \frac{\text{Precision}_{\pi|L} \cdot \text{Recall}_{\pi|L}}{\text{Precision}_{\pi|L} + \text{Recall}_{\pi|L}}
\]

On the other hand, as the number of filters in each rule and the number of rules in an access control policy increases, policy intelligibility would decrease and maintenance of the policy would become harder. Hence, complexity is another key metric for evaluating the quality of a policy. Weighted Structural Complexity (WSC) is a generalization of policy size and was first introduced for RBAC policies [21] and later extended for ABAC policies [15]. WSC is consistent with usability studies of access control rules, which indicates that the more concise the policies are the more manageable they become [22]. Informally, for a given ABAC policy, its WSC is a weighted sum of the number of its elements. Formally, for an ABAC policy \( \pi \) with rules \( \mathcal{P} \), its WSC is defined as follows:

\[
WSC(\pi) = WSC(\mathcal{P})
\]

\[
WSC(\mathcal{P}) = \sum_{\rho \in \mathcal{P}} WSC(\rho)
\]

\[
WSC(\rho) = w_1 WSC(F_U) + w_2 WSC(F_O) + w_3 WSC(F_S) + w_4 WSC(R)
\]

\[
WSS_{\mathcal{S}}(\mathcal{S}) = \sum_{s \in \mathcal{S}} |s|
\]

where \( |s| \) is the cardinality of set \( s \) and the \( w_i \)'s are user-specified weights.

Van Rijsbergen’s proposed an effectiveness measure for combining two different metrics \( P \) and \( R \) as follows [23]:

\[
E = 1 - \frac{1}{\frac{\alpha}{P} + \frac{1 - \alpha}{R}}
\]

To be able to compare the quality of different mined ABAC policies, we combine the two metrics based on Van

| Table 2: State-of-the-art ABAC Rule Mining Techniques |
|-----------------------------------------------------|
| Policy Correctness | Xu et al. [13] | Medvet et al. [16] | Iyer et al. [17] | Cotrini et al. [18] | Our Proposed Approach |
|---------------------|----------------|-------------------|-----------------|-------------------|---------------------|
| Policy Complexity   | ✓              | ✓                 | ✓               | ✓                 | ✓                   |
| Negative Attribute Filters | ✓          | ✓                 | ✓               | ✓                 | ✓                   |
| Relation Conditions | ✓              | ✓                 | ✓               | ✓                 | ✓                   |
| Sparse Logs         | ✓              | ✓                 | ✓               | ✓                 | ✓                   |
| Negative Authorization Rules | ✓         | ✓                 | ✓               | ✓                 | ✓                   |
| Noisy Authorization Log | ✓           | ✓                 | ✓               | ✓                 | ✓                   |
| System Dynamicity   | ✓              | ✓                 | ✓               | ✓                 | ✓                   |
Rijsbergen’s effectiveness measure [23] and define the Policy Quality Metric as follows:

\[
Q_\pi = \left( \frac{\alpha}{F_{\text{score}}(\pi) |L|} + \frac{1 - \alpha}{\Delta WSC_{\pi}} \right)^{-1}
\]

Here \( \alpha = \frac{1}{1 + \beta^2} \) where \( \beta \) determines the importance of relative F-score over policy complexity and \( \Delta WSC_{\pi} \) shows the relative loss of the complexity with regards to the complexity of worst-case mined policy. \( \Delta WSC_{\pi} \) is calculated as follows:

\[
\Delta WSC_{\pi} = \frac{WSC_{\text{max}} - WSC(\pi) + 1}{WSC_{\text{max}}}
\]

\.  

\( WSC_{\text{max}} \) is the weighted structural complexity of the worst-case mined policy.

### Definition 17. (Worst-case Mined Policy)

Worst-case mined policy \( \pi_w \) is the policy that is extracted by iterating through access log \( L \) and adding an access control rule for each authorization tuple if it’s not already included in the mined policy. The corresponding rule for each authorization tuple includes all attributes of user, object, and subject of that authorization tuple.

The worst-case mined policy has F-score equal to one; this occurs if the access log is complete, which means that every possible combination of attribute values are represented in the access log. On the other hand, it has a high complexity resulting in very low quality policy. Considering the equal importance of relative F-score and relative loss of complexity of the policy, we calculate the quality measure as follows:

\[
Q_\pi = \frac{2 \cdot F_{\text{score}}(\pi) |L| \cdot \Delta WSC_{\pi}}{F_{\text{score}}(\pi) |L| + \Delta WSC_{\pi}}
\]

A mined policy with a higher F-score would have a higher policy quality. On the other hand, as the complexity of a policy increases, its quality will decrease. The intuition here is that once an extracted policy reaches a high F-score, adding additional rules will lead to a decrease in \( Q_\pi \).

For the worst-case mined policy \( \pi_w \), \( \Delta WSC_{\pi_w} \approx 0 \), so its policy quality \( Q_{\pi_w} \) is very close to zero. For an empty mined policy \( \pi_e \) (a policy without any rule), while \( \Delta WSC_{\pi_e} \approx 1 \), as it denies all the access requests, its false negative rate is one and its true positive rate is zero. So its precision is zero and as a result, its F-score is zero as well. So the quality of the empty policy \( Q_{\pi_e} \) is zero, too.

The worst-case mined policy and the empty mined policy are the two extreme cases with policy quality equal to zero. Other mined policies between these two cases have higher policy quality than zero.

### 4 Proposed Approach

Our proposed learning-based ABAC policy extraction procedure consists of the steps summarized in Figure 1.

#### 4.1 Data Pre-processing

As features of our learning algorithm are categorical variables, the first step in pre-processing the access log is to convert all numerical variables to their corresponding categorical values.

We also need to handle missing values in this step. As the frequency of each attribute value is an important factor in our rule extraction algorithm (Section 4.4) for deciding if an attribute is effective or not, it’s important to replace missing values in a way that it doesn’t mess up with the original frequency of each attribute value. For this purpose, we replace each missing value by \( \text{UNK} \) (i.e., unknown).

#### 4.2 Selection of Learning Algorithm

We use the \textit{K-modes algorithm} [24], which is a well known unsupervised learning algorithm used for clustering categorical data. \textit{K-modes} has been proved effective in mining ABAC policies [25]; this algorithm uses an initialization method based on both the distance between data points and the density of data points. Using both density and distance when initializing clusters help avoid two problems: (i) clustering outliers as new clusters are based only on the distances; and (ii) creating new clusters in the surrounding of one center based only on the density. Compared to a random initialization method, this method provides more robustness and better accuracy in the clustering process [24].

#### 4.3 Parameter Tuning

In the next step, we \textit{tune the learning parameters}. There are several challenges that need to be addressed in this step, which include the following:

##### 4.3.1 Number of Clusters (\(k\))

One of the main challenges in an unsupervised learning is determining the number of clusters, \(k\). In our sample policies, as we know the number of rules in each policy, we can set the number of clusters beforehand but in a real situation as we do not know the size of the rules in
advance, making the correct choice of \( k \) is difficult. One of the popular methods for determining the number of clusters in an unsupervised learning model is the Elbow Method \([26, 27]\). This method is based on total within group sum of squares. \( k \) will be chosen as the number of clusters if adding another cluster doesn’t give much better modeling of the data (i.e., the elbow point of the graph).

As a second approach, we choose a number of clusters \( (k) \) which gives the best modeling of the data in terms of the policy quality metric. For this purpose, we run our clustering algorithm for different values of \( k \) and calculate the accuracy of the corresponding model using 10-fold cross-validation. The value of \( k \) that maximizes the accuracy of the model is selected as the final number of clusters.

Note that increasing \( k \) will ultimately reduce the amount of clustering error or it will increase the accuracy of the model, but by increasing the number of clusters, the number of extracted rules will also increase resulting in more complexity (i.e., higher WSC). So it is important to find an optimal \( k \) that balances between policy accuracy and WSC.

### 4.3.2 Cluster Initialization & Local Optima

Different cluster initializations can lead to a different set of clusters as \( k \)-means/\( k \)-modes may converge to local optima. To overcome this issue, for a given number of clusters, \( k \), we train multiple models with different cluster initializations and then select the partition with the smallest clustering error.

### 4.4 Policy Rules Extraction

The main phase in our proposed approach is the extraction of ABAC policy rules. In the first step, we need to collect all the authorization tuples related to each rule of the policy. We use data clustering for this purpose. We divide the access log into clusters where the records in each cluster correspond to one AC rule in the system. This is done based on finding similar patterns between features (i.e., attribute values) of the records (i.e., access control tuples). In the second step, we extract the attribute filters of such a rule. We adapt the rule extraction algorithm in \([25]\) and extend it to extract both positive and negative attribute filters. We define effective positive attribute and effective negative attribute as follows:

**Definition 18. (Effective Positive (Negative) Attribute).** Let \( S = \{(a, v)\} \) be the set of all possible attribute-value pairs in a system; we define \( \langle a_j, v_j \rangle \in S \) (\( \langle a_j, !v_j \rangle \in S \) as an effective positive (negative) attribute pair of \( \rho_i \) corresponding to cluster \( C_j \), where the frequency of occurrence of \( v_j \) in the set of all the records of cluster \( C_j \) is much higher (lower) than its frequency of occurrence in the original data; this is determined based on a threshold \( T_P \) (\( T_N \)). The attribute expression \( \langle a_j, v_j \rangle \) (\( \langle a_j, !v_j \rangle \)) is added to the attribute filters of the extracted rule \( \rho_i \) for \( C_j \).

In the final step, we extract the relation conditions for AC rules for each cluster. This will be done based on the frequency of equality between pairs of attributes in the records of each cluster. We define effective positive relation and effective negative relation as follows:

**Definition 19. (Effective Positive (Negative) Relation).** Let \( R = \{(a, b)\} \) be the set of all possible relations between pairs of attributes in the system; we define \( \langle a_j, b_j \rangle \) as an effective positive (negative) relation pair of \( \rho_i \) corresponding to cluster \( C_j \), where the frequency of \( a_j \) equals \( b_j \) in all the records of cluster \( C_j \) is much higher (lower) than their frequency in the original data; this is determined based on a threshold \( T_P \) (\( T_N \)). The relation \( \langle a_j, b_j \rangle \) \( \langle a_j, !b_j \rangle \) is added to the relation conditions of the extracted rule \( \rho_i \) for this cluster.

Algorithms 1 and 2 show effective attribute and effective relation extraction procedures, respectively.

| Algorithm 1 Effective attribute extraction algorithm |
|-----------------------------------------------|
| **Input:** \( C_i, A, V, L, T_P, T_N \) |
| **Output:** \( F \) |
| **procedure** EXTRACTATTRIBUTEFILTERS |
| 1: \( F \leftarrow \emptyset \) |
| 2: for all \( a \in A \) do |
| 3: for all \( v_j \in V_a \) do |
| 4: if \( \text{Freq}(v_j, C_i) - \text{Freq}(v_j, L) > T_P \) then |
| 5: \( F_i \leftarrow F \cup \langle a, v_j \rangle \) |
| 6: end if |
| 7: if \( \text{Freq}(v_j, L) - \text{Freq}(v_j, C_i) > T_N \) then |
| 8: \( F_i \leftarrow F \cup \langle a, !v_j \rangle \) |
| 9: end if |
| 10: end for |
| 11: end for |
| 12: return \( \rho_i \) |
| 13: end procedure |

| Algorithm 2 Effective relation extraction algorithm |
|-----------------------------------------------|
| **Input:** \( C_i, A, L, \theta_P, \theta_N \) |
| **Output:** \( R \) |
| **procedure** EXTRACTRELATIONS |
| 1: \( R \leftarrow \emptyset \) |
| 2: for all \( a \in A \) and \( b \neq a \) do |
| 3: for all \( b \in A \) and \( \theta_P \) do |
| 4: if \( \text{Freq}(a = b, C_i) - \text{Freq}(a = b, L) > \theta_P \) then |
| 5: \( R \leftarrow R \cup \langle a, b \rangle \) |
| 6: end if |
| 7: if \( \text{Freq}(a = b, L) - \text{Freq}(a = b, C_i) > \theta_N \) then |
| 8: \( R \leftarrow R \cup \langle a, !b \rangle \) |
| 9: end if |
| 10: end for |
| 11: end for |
| 12: return \( R \) |
| 13: end procedure |

### 4.5 Policy Enhancement

After the first phase of policy rule extraction, we get a policy which may not be as accurate and concise as we desire. We enhance the quality of the mined policy through iterations of policy improvement steps that include: rule pruning and policy refinement.

#### 4.5.1 Rule Pruning

During the rule extraction phase, it’s possible to have two clusters that correspond to the same rule. As a result, the
extracted rules of these clusters are very similar to each other. Having two similar rules in the final policy increases the complexity of the mined policy while it may not help the accuracy of the policy and as a result, it hurts the policy quality. To address such an issue, in the rule pruning step, we identify similar rules and eliminate the ones whose removal improves the policy quality more. If eliminating neither of the two rules improves the policy quality, we keep both the rules. This may happen when we have two very similar AC rules in the original policy. We measure the similarity between two rules using Jaccard similarity $J$ as follows:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

So we calculate the similarity between two rules $\rho_1$ and $\rho_2$ as follows:

$$J(\rho_1, \rho_2) = \frac{\sum_{F \in \{F_π, F_\text{FN}, F_\text{FP}\}} |F_\rho_1 \cap F_\rho_2| + |R_\rho_1 \cap R_\rho_2| + |op_\rho_1 \cap op_\rho_2|}{\sum_{F \in \{F_π, F_\text{FN}, F_\text{FP}\}} |F_\rho_1 \cup F_\rho_2| + |R_\rho_1 \cup R_\rho_2| + |op_\rho_1 \cup op_\rho_2|}$$

We consider two rules to be similar if their Jaccard similarity score is more than 0.5, which means that the size of their common elements is more than half of the size of the union of their elements. Algorithm 3 shows the rule pruning procedure.

**Algorithm 3 Rule Pruning algorithm**

1: procedure RULEPRUNING  
Input: $\pi$  
Output: $\pi$  
2: $P \leftarrow \pi$  
3: $q \leftarrow \text{CALCQUALITY}(P)$  
4: for all $\rho_i \in P$ do  
5:     for all $\rho_j \in P$ and $\rho_i \neq \rho_j$ do  
6:         if $\text{SIMILARITY}(\rho_i, \rho_j) > 0.5$ then  
7:             $P_i \leftarrow P_i / \rho_i$  
8:             $P_j \leftarrow P_j / \rho_j$  
9:             $q_i \leftarrow \text{CALCQUALITY}(P_i)$  
10:            $q_j \leftarrow \text{CALCQUALITY}(P_j)$  
11:                if $q_i > q$ and $q_j > q_i$ then  
12:                    $P \leftarrow P_i$  
13:                end if  
14:            end if  
15:        end for  
16:    end if  
17: end for  
18: return $P$  
19: end procedure

4.5.2 Policy Refinement

During the rule extraction phase, it is possible to extract rules that are either too restricted or too relaxed compared to the original policy rules. A rule is restricted if it employs more filters than the original rule.

**Example 6.** Consider the following two rules:

$$\rho_1 = \{(\text{position}, \text{faculty}), \{(\text{type}, \text{gradebook}), (\text{setScore}, \text{permit})\}

\rho_2 = \{(\text{position}, \text{faculty}, \text{uDept}, \text{EE}), (\text{type}, \text{gradebook}), (\text{setScore}, \text{permit})\}$$

Here $\rho_2$ is more restricted than $\rho_1$ as it imposes more conditions on the user attributes.

Having such a restricted rule in the mined policy would result in a larger number of FN records as an access request that would be permitted by the original rule will be denied by the restricted rule.

On the other hand, an extracted rule is more relaxed compared to the original rule if it misses some of the filters. In Example 6, $\rho_1$ is more relaxed than $\rho_2$. Such a relaxed rule would result in more FP records as it permits access requests that should be denied as per the original policies.

To address these issues, we propose a policy refinement procedure which is shown in Algorithm 4. Here, we try to refine the mined policy ($\pi_m$) based on the patterns discovered in the FN or FP records. These patterns are used to eliminate extra filters from restricted rules or append missing filters to relax the rules.

To extract patterns from the FN or FP records, we apply our rule extraction procedure on these records to get the corresponding policies $\pi_{\text{FN}}$ and $\pi_{\text{FP}}$. Here our training data are FN and FP records, respectively. We compare the extracted FN or FP rules with the mined policy and remove the extra filters or append the missed ones to the corresponding rules. As an example, consider the FP records. Here, our goal is to extract the patterns that are common between access requests that were permitted based on the mined policy while they should have been denied based on the original policy.

In each step of refinement, a rule from $\pi_m$ that is similar to a rule from $\pi_{\text{FN}}$ or $\pi_{\text{FP}}$ based on the Jaccard similarity (Section 4.5.1) is selected and then refined in two ways as discussed below.

**Policy refinement based on $\pi_{\text{FN}}$:** In the case of FN records, two situations are possible: a rule is missing from the mined policy ($\pi_m$) or one of the rules in $\pi_m$ is restricted. To resolve this issue, for each rule $\rho_i \in \pi_{\text{FN}}$:

- if there is a similar rule $\rho_j \in \pi_m$ then we refine $\rho_j$ as follows:

$$\forall f \in F : F_{\rho_j} = F_{\rho_j} / (F_{\rho_j} / F_{\rho_i})$$

where $F = F_{\text{FIL}} \cup F_{\text{FIL}} \cup F_{\text{SR}} \cup R$. So, the extra filters are removed from the restricted rule ($\rho_j$).

- if there is no such rule, then $\rho_i$ is the missing rule and we add it to $\pi_m$.

**Policy refinement based on $\pi_{\text{FP}}$:** In the case of FP records, some filters might be missing in an extracted rule in the mined policy ($\pi_m$); so for each rule $\rho_i \in \pi_{\text{FP}}$, we refine the mined policy as follows:

$$\forall f \in F : F_{\rho_j} = F_{\rho_j} \cup (F_{\rho_j} / F_{\rho_i})$$

where $F = F_{\text{FIL}} \cup F_{\text{FIL}} \cup F_{\text{SR}} \cup R$ includes all the filters in the rule. So, the missing filters are added to the relaxed rule ($\rho_j$).
These refinements can be done in multiple iterations until further refinement does not give a better model in terms of policy quality $Q_\pi$.

Algorithm 4 Policy refinement algorithm

1: procedure \texttt{REFINEPOLICY}

Input: $A, \mathcal{L}$

Output: $\pi_m$

2: $\mathcal{F}N \leftarrow \text{GETFNs}(\pi_m, \mathcal{L})$

3: $\pi_{FN} \leftarrow \text{EXTRACTPOLICY}(\mathcal{F}N)$

4: for all $\rho_i \in \mathcal{F}N \cdot \mathcal{P}$ do

5: $R_i \leftarrow \text{GETSIMILARRULES}(\pi_{FN} \cdot \mathcal{P}, \pi_m, \mathcal{P})$

6: if $|R_i| = 0$ then

7: $\pi_m \cdot \mathcal{P} \leftarrow \pi_m \cdot \mathcal{P} \cup \rho_i$

8: else

9: for all $\rho_j \in R_i$ do

10: for all $\mathcal{F} \in \mathcal{F}_U \cup \mathcal{F}_O \cup \mathcal{F}_S \cup \mathcal{R}$ do

11: $\mathcal{F}_{\rho_j} \leftarrow (\mathcal{F}_{\rho_j} \setminus (\mathcal{F}_{\rho_i} \setminus \mathcal{F}_{\rho_j}))$

12: end for

13: end for

14: end if

15: end for

16: $\mathcal{F}P \leftarrow \text{GETFPs}(\pi_m, \mathcal{L})$

17: $\pi_{FP} \leftarrow \text{EXTRACTPOLICY}(\mathcal{F}P)$

18: for all $\rho_i \in \mathcal{F}_{FP} \cdot \mathcal{P}$ do

19: $R_i \leftarrow \text{GETSIMILARRULES}(\pi_{FP} \cdot \mathcal{P}, \pi_m, \mathcal{P})$

20: if $|R_i| = 0$ then

21: for all $\rho_j \in R_i$ do

22: for all $\mathcal{F} \in \mathcal{F}_U \cup \mathcal{F}_O \cup \mathcal{F}_S \cup \mathcal{R}$ do

23: $\mathcal{F}_{\rho_j} \leftarrow (\mathcal{F}_{\rho_j} \cup (\mathcal{F}_{\rho_i} \setminus \mathcal{F}_{\rho_j}))$

24: end for

25: end for

26: end if

27: end for

28: return $\pi_m$

end procedure

5 Experimental Evaluation

We have implemented a prototype of our proposed model presented in Section 4. Here, we present our experimental evaluation.

5.1 Datasets

We perform our experiments on multiple datasets including synthesized and real ones. The synthesized access logs are generated from two sets of ABAC policies: a manually written set of policies adapted from [15] and a completely randomly generated set of policies. To synthesize our input data, for each ABAC policy (i.e., University Policy, Healthcare Policy, etc.), a set of authorization tuples is generated and the outcome of the ABAC policy for each access right is evaluated. The authorization tuples with permit as their outcomes are the inputs to our unsupervised learning model.

Our real datasets are built from access logs provided by Amazon in Kaggle competition [29] and available in the UCI machine learning repository [30].

Manual Policy - University: This policy is adapted from [15] and it controls access of different users including students, instructors, teaching assistants, etc., to various objects (applications, gradebooks, etc.).

Manual Policy - Healthcare: This policy is adapted from [15] and is used to control access by different users (e.g., nurses, doctors, etc.) to electronic health records (EHRs) and EHR items.

Manual Policy - Project Management: This policy is adapted from [15] and it controls access by different users (e.g., department managers, project leaders, employees, etc.) to various objects (e.g., budgets, schedules and tasks).

Random Policies: The authorization rules for this policy is generated completely randomly from random sets of attributes and attribute values.

Real Dataset - Amazon Kaggle: The Kaggle competition dataset [29] includes access requests made by Amazon’s employees over two years. Each record in this dataset describes an employee’s request to a resource and whether the request was authorized or not. A record consists of the employee’s attributes and values and the resource identifier. The dataset includes more than 12,000 users and 7,000 resources. In this experiment we focused on the top five resources in the dataset.

Real Dataset - Amazon UCI: This dataset is provided by Amazon in the UCI machine learning repository [30]. It includes more than 36,000 users and 27,000 permissions. Since the dataset contains over 33,000 attributes, our focus in this experiment is narrowed only to the top requested 18 attributes in the dataset.

Table 3 shows the details about the manual and random access log datasets. In this table, $|\mathcal{P}|$ shows the number of rules in the original policy, $|A|$ and $|V|$ show the number of attributes and attribute values, respectively, $WSC(\pi)$ shows the complexity of the mined policy and $|\mathcal{L}|$ shows the number of access control tuples in the given access log.

5.2 Experimental Setup

To evaluate our proposed method, we use a computer with 2.6 GHz Intel Core i7 and 16 GB of RAM. We used Python 3 in the mining and the evaluation process. The algorithms were highly time-efficient (e.g., maximum time consumption is less than half an hour).

To generate the synthesized access log $\mathcal{L}$ we brute force through all attributes $A$ and their values $V_a$ to produce all possible combinations for the tuples. This method was used to generate a complete access log for the random and manual policies dataset. We also produced a sparse dataset by randomly selecting authorization tuples from the complete dataset. For example, a 20% sparse dataset is generated by randomly removing 10% of the permitted and 10% of the denied tuples from the complete access logs. For the noisy dataset, we combine a portion of the data by setting denied tuples as permitted and vise versa. For instance, 20% noisy data contains 10% of the denied tuples in the permitted log as permitted, and another 10% of the permitted tuples in the denied log as denied. The average threshold for selecting effective attributes and relations is between 0.3 and 0.20 in all experiments.
### 5.3 Results

Figure 2 shows the final $F$-score $\pi_{L}$ (after multiple rounds of refinement) for all the policies. As we can see, all the policies have F-score higher than 88% with five out of nine policies reaching 100% F-score after multiple rounds of refinement. These numbers are much higher than the Cotrini et al. [18] and Xu et al. [15] F-score for the same policies which are reported in [18]. The highest F-score Cotrini et al. [18] could achieve for the real dataset is around 35% while the reported Xu et al. F-score for all real datasets is 0% as their proposed approach is incapable of extracting complex policies.

![Fig. 2. Final F-Score of the policies mined by our proposed algorithm.](image)

Figure 3 shows the size of the mined policy for all policy sets. The policy sizes are all reasonable and manageable and much lower than the reported policy sizes in [18]. As an example, the overall mined policy size for Amazon Kaggle dataset in our approach is 71 while the size of the mined policy for the same dataset in [18] is more than 1000.

![Fig. 3. Size of the policies mined by our proposed algorithm.](image)

Figure 4 shows the improvement of F-score of the mined policy after each round of refinement for various policy sets. Some policies (e.g. Amazon Kaggle and Amazon UCI) reach a 100% F-score after few rounds of refinement while some policies (e.g. HealthcareP) need more rounds of refinement to achieve a good F-score.

![Fig. 4. Improvement of the F-score of the mined policies after multiple rounds of refinement.](image)

Figure 5 represents the effect of noisy input access log on the accuracy of the mined policy for each of manual policies. Every experiment includes four rounds of refinement. The 0% noise indicates a complete access log without any over-assignment or under-assignment, while 10% noise shows that 10% of records are misclassified. As we can see, a noisy dataset reduces the correctness of the final mined policy. However, the drop in correctness for HealthcareP and UniversityP is negligible and even with the presence of 30% noise the accuracy of the mined policies are over 93%.

![Fig. 5. Effect of noise on the accuracy of the mined policies.](image)

Finally, Figure 6 shows the effect of the sparsity of the input access log on the accuracy of the mined policy for each of manual policies. Every experiment includes four rounds of refinement. Here, 0% means that the given access log is complete while 10% sparsity mean that the given access log...
misses 10% of the records of the complete access log. The trends of the graphs show that as the sparsity increases, the quality of the mined policy in terms of correctness decreases. However, in the case of UniversityP policy, even with the 30% sparsity in the authorization logs, our proposed approach reached a 100% accuracy after few rounds of refinement. This indicates that even though the input log is sparse but the clusters are well formed and the records corresponding to each rule is well separated. In addition, as we omitted the records of the complete log randomly, this selection may also effect the quality of the final policy. As we can see, the accuracy of the ProjectManagementP policy with 30% sparsity is better than its accuracy when access log is 20% sparse.

![Fig. 6. Effect of sparsity on the accuracy of the mined policies.](image)

### 6 Related Work

As RBAC approach became popular, many organization decided to equip their information systems with more recent access control model, however migrating to RBAC from legacy access control systems was a huge obstacle for such environments. As a result, several researchers addressed such a challenge by introducing automated role extraction algorithms [10], [11], [12], [13], [21], [31], [32], [33], [34], [35], [36]. Role engineering or role mining are the terms that have been used to refer to procedures to extract an optimal set of roles given user-permission assignments.

In [10], Kuhlmann and Schimpf try to discover a set of roles from user-permission assignments using clustering techniques, however, they do not show the feasibility of their proposed approach through experiments. In addition, their proposed approach lacks a metric to choose the best model based on their clustering method.

The ORCA role mining tool is proposed by Schlegelmilch and Steffens and tries to perform a hierarchical clustering on user-permission assignments [11]. Their proposed method limits the hierarchical structure to a tree so that each permission/user is assigned to one role in the hierarchy. This feature limits the feasibility of their proposed approach as, in real environments, roles do not necessarily form a tree.

Ni et al. propose a supervised learning approach for role mining which maps each user-permission assignment to a role using a supervised classifier (i.e., a support vector machine (SVM)) [36]. The main limitation of their proposed approach is that the roles and some part of the role-permission assignments are needed beforehand; and hence, it is not applicable in many organizations.

Vaidya et al. are the first to define the Role Mining Problem (RMP) formally and analyze its theoretical bounds [37]. They also propose a heuristic approach for finding a minimal set of roles for a given set of user-permission assignments.

Xu and Stoller are the first to propose an algorithm for mining ABAC policies from RBAC [38], logs [14], and access control list [15] plus attribute information. Their policy mining algorithms iterate over access control tuples (generated from available information, e.g., user permission relations and attributes) and construct candidates rules. They then generalize the candidate rules by replacing conjuncts in attribute expressions with constraints. The main limitation of these algorithms is that as they are based on heuristic approaches, the proposed techniques work very well for simple and small scale AC policies, however, as the number of rules in the policy and the number of elements in each rule increases, they do not perform well.

Following Xu and Stoller’s proposed method, Medvet et al. [16] propose a multi-objective evolutionary algorithm for extracting ABAC policies. The proposed approach is a separate and conquer algorithm, in each iteration of which, a new rule is learned and the set of access log tuples decreases to a smaller size. Their algorithm employs several search-optimizing features to improve the quality of the mined rules. Although their approach is a multi-objective optimization framework which incorporates requirements on both correctness and expressiveness, it suffers from the same issue as [15].

Iyer and Masoumzadeh [17] propose a more systematic, yet heuristic ABAC policy mining approach which is based on the rule mining algorithm called PRISM. It inherits shortcomings associated with PRISM that includes dealing with a large dimensionality of the search space of attribute values and generation of a huge number of rules.

Cotrini et al. propose an algorithm called Rhapsody for mining ABAC rules from sparse logs [18]. Their proposed approach is built upon subgroup discovery algorithms. They define a novel metric, reliability which measures how overly permissive an extracted rule is. In addition, they proposed a universal cross-validation metric for evaluating the mined policy when the input log is sparse. However, their algorithm is not capable of mining policies from logs with many attributes as the number of extracted rules grows exponentially in the number of attributes of the system.

### 7 Discussion and Limitations

As mentioned in section 5.3 our proposed approach is able to maintain a practical performance when applying to both synthesized and real datasets. In the case of synthesized datasets, we demonstrate the approach capability in mining policies containing both positive and negative attribute filters from complete datasets. On the other hand, real datasets show the approach potentials in mining such policies in the case of sparse datasets. In addition, the real datasets contain large number of attributes and attribute values as shown in Table 3. The ability of our proposed approach in mining high-quality policies for these datasets verifies that the size of attributes and attribute values has minimal impact on the effectiveness of our approach.
Our approach provides consistent results when mining policies for simple and small access logs as well as complex and large ones. Our experiments on various datasets with different levels of sparseness and noise show that the proposed approach maintains a good performance with a slight drop in the evaluation scores.

The proposed approach is based on an unsupervised clustering algorithm. Since finding the proper number of clusters is a challenge in the clustering algorithm, our approach is subjective to this issue as well. The same issue appears in finding the best thresholds to extract effective attributes and relations.

In our evaluation, we used random selection to create noisy and sparse datasets from complete datasets. Although we ensured the percentage of randomly selected tuples from permitted and denied logs, guaranteeing the quality of the sample is difficult.

8 Conclusion
In this paper, we have proposed an unsupervised learning based approach to automate ABAC policy extraction. The proposed approach is capable of discovering both positive and negative attribute expressions as well as positive and negative relation conditions while previous approaches in access control policy extraction had only focused on positive expressions. Furthermore, our work is capable of improving the extracted policy through iterations of our proposed rule pruning and policy refinement algorithms. Such refinement algorithms are based on the false positive and false negative records and they help in increasing the quality of the mined policy.

Most importantly, we have proposed the policy quality metric which considers both the conciseness and correctness of the mined policy and is important for comparing the extracted policy with the original one and improving it if needed.

We have evaluated our policy extraction algorithm on access logs generated for various sample policies and demonstrated its feasibility. Furthermore, we have shown that our approach outperforms previous works in terms of policy quality.

As future work, we plan to extend our method to support numerical data and extract negative authorization rules as well while studying the effect of various conflict resolution strategies on the quality of the mined policy.

References
[1] R. S. Sandhu and P. Samarati, “Access control: principle and practice,” IEEE communications magazine, vol. 32, no. 9, pp. 40–48, 1994.
[2] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, “Protection in operating systems,” Communications of the ACM, vol. 19, no. 8, pp. 461–471, 1976.
[3] D. E. Bell and L. J. LaPadula, “Secure computer systems: Mathematical foundations,” tech. rep., MITRE CORP BEDFORD MA, 1973.
[4] R. S. Sandhu, “Lattice-based access control models,” Computer, vol. 26, no. 11, pp. 9–19, 1993.
[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-based access control models,” Computer, vol. 29, no. 2, pp. 38–47, 1996.
[6] P. W. Fong and I. Siahahan, “Relationship-based access control policies and their policy languages,” in Proceedings of the 16th ACM symposium on Access control models and technologies, pp. 51–60, ACM, 2011.
[7] J. Jin, G.-J. Ahn, H. Hu, M. J. Covington, and X. Zhang, “Patient-centric authorization framework for sharing electronic health records,” in Proceedings of the 14th ACM symposium on Access control models and technologies, pp. 125–134, ACM, 2009.
[8] L. Karimi and J. Joshi, “Multi-owner multi-stakeholder access control model for a healthcare environment,” in Collaboration and Internet Computing (CIC), 2017 IEEE 3rd International Conference on, pp. 359–368, IEEE, 2017.
[9] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, et al., “Guide to attribute based access control (abac) definition and considerations (draft),” NIST special publication, vol. 800, no. 162, 2013.
[10] M. Kuhlmann, D. Shohat, and G. Schimpf, “Role mining-revealing business roles for security administration using data mining technology,” in Proceedings of the eighth ACM symposium on Access control models and technologies, pp. 179–186, ACM, 2003.
[11] J. Schlegelmilch and U. Steffens, “Role mining with orca,” in Proceedings of the tenth ACM symposium on Access control models and technologies, pp. 168–176, ACM, 2005.
[12] V. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, “Mining roles with semantic meanings,” in Proceedings of the 13th ACM symposium on Access control models and technologies, pp. 21–30, ACM, 2008.
[13] Z. Xu and S. D. Stoller, “Algorithms for mining meaningful roles,” in Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 57–66, ACM, 2012.
[14] Z. Xu and S. D. Stoller, “Mining attribute-based access control policies from logs,” in IFIP Annual Conference on Data and Applications Security and Privacy, pp. 276–291, Springer, 2014.
[15] Z. Xu and S. D. Stoller, “Mining attribute-based access control policies,” IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 5, pp. 533–545, 2015.
[16] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari, “Evolutionary inference of attribute-based access control policies,” in EMO (1), pp. 351–365, 2015.
[17] P. Iyer and A. Masoumzadeh, “Mining positive and negative attribute-based access control policy rules,” in Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies, pp. 161–172, ACM, 2018.
[18] C. Cotrini, T. Weghorn, and D. Basin, “Mining abac rules from sparse logs,” in 2018 IEEE European Symposium on Security and Privacy (EuroS&F), pp. 31–46, IEEE, 2018.
[19] P. Marinescu, C. Parry, M. Pomarole, Y. Tian, P. Tague, and I. Papagiannis, “Ivd: Automatic learning and enforcement of authorization rules in online social networks,” in 2017 IEEE Symposium on Security and Privacy (SP), pp. 1094–1109, IEEE, 2017.
[20] Wikipedia contributors, “Accuracy paradox-wikipedia, the free encyclopedia,” 2018. [Online; accessed 30-September-2019].
[21] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, “Mining roles with multiple objectives,” ACM Transactions on Information and System Security (TISSEC), vol. 13, no. 4, p. 36, 2010.
[22] M. Beckerle and L. A. Martucci, “Formal definitions for usable business roles for security administration using data mining technologies,” in Proceedings of the Ninth Symposium on Usable Privacy and Security, p. 2, ACM, 2013.
[23] C. J. v. Rijsbergen, Information retrieval. 2.ed, Butterworths, 1979.
[24] F. Cao, J. Liang, and L. Bai, “A new initialization method for categorical data clustering,” Expert Systems with Applications, vol. 36, no. 7, pp. 10223–10228, 2009.
[25] L. Karimi and J. Joshi, “An unsupervised learning based approach for mining attribute based access control policies,” in Big Data (Big Data), 2018 IEEE International Conference on, IEEE, 2018.
[26] R. L. Thorndike, “Who belongs in the family?,” Psychometrika, vol. 18, no. 4, pp. 267–276, 1953.
[27] C. Goutte, P. Toft, E. Rostrup, F. Å. Nielsen, and L. K. Hansen, “On clustering fmrI time series,” NeuroImage, vol. 9, no. 3, pp. 298–310, 1999.
[28] P. Jaccard, “The distribution of the flora in the alpine zone. 1,” New phytologist, vol. 11, no. 2, pp. 37–50, 1912.
[29] Amazon.com, “Amazon employee access challenge.” Kaggle.
[30] Montanez, Ken, “Amazon access samples.” UCI Machine Learning Repository: Amazon Access Samples Data Set.
Leila Karimi received an undergraduate degree and the MS degree in information technology engineering from the Sharif University of Technology, Tehran, Iran. She is a Ph.D. candidate at the School of Computing and Information (SCI), at the University of Pittsburgh. Her research interests lie at the intersection of information security, data privacy, and machine learning. Currently, she is working on applying machine learning techniques to solve challenging problems in the security domain.

Maryam Aldairi received an undergraduate degree management information systems from King Faisal University, Alhasa, KSA., and the MS degree in information science from the University of Pittsburgh. She is a Ph.D. student at the School of Computing and Information (SCI), at the University of Pittsburgh. Her research interests lie at the intersection of information security, adversarial learning, and machine learning. Currently, her focus is on applying machine learning techniques to solve challenging problems in the security domain.

James Joshi received the MS degree in computer science and the Ph.D. degree in computer engineering from Purdue University, in 1998 and 2003, respectively. He is a professor of School of Computing and Information (SCI), at the University of Pittsburgh. His research interests include Access Control Models, Security and Privacy of Distributed Systems, Trust Management and Information Survivability. He is the director of LERSAIS at the University of Pittsburgh. He is an elected fellow of the Society of Information Reuse and Integration (SIRI) and is a senior member of the IEEE and the ACM.

Mai Abdelhakim is an assistant professor in the department of electrical and computer engineering at the University of Pittsburgh’s Swanson school of engineering. She received her Ph.D. degree in Electrical Engineering from Michigan State University in 2014, and Bachelor’s and Master’s degrees in Electronics and Communications Engineering from Cairo University in 2006 and 2009, respectively. Prior to joining Swanson school of engineering, she was a visiting assistant professor in the School of Computing and Information at the University of Pittsburgh. Prior to that, she was a postdoctoral research scientist at OSRAM Research Center (2015-2016) and a postdoctoral research associate at Michigan State University (2014-2015). Her research interests include cyber-physical systems, cybersecurity, machine learning, stochastic systems modeling, and information theory.