An Unsupervised Learning Based Approach for Extracting Relationship and Attribute Based Access Control Policy

ABSTRACT

With the rapid advances in computing and information technologies, traditional access control models have become inadequate in terms of capturing fine-grained, expressive security requirements of newly emerging applications. A relationship and attribute-based access control (ReABAC) model provides a more flexible approach for addressing access control needs of complex and dynamic systems. While organizations are interested in employing such newer access control approaches, the challenge of migrating to such AC approaches pose as a significant obstacle. Many large-scale businesses need to grant authorization to their huge user populations that are distributed across disparate and heterogeneous computing environments including legacy systems and the latest/emerging ones. Each of these computing environments may have their own access control model. Manual development of a single policy framework for a whole organization is tedious, costly and error-prone.

In this paper, we present a methodology for automatically learning ReABAC policy rules from access logs of a system to simplify the policy development/refinement process. The proposed approach employs an unsupervised learning-based algorithm for detecting patterns in access logs and extracting ReABAC authorization rules from these patterns. We propose a policy quality metric to evaluate the extracted policy. In addition, we present two policy improvement algorithms including rule pruning and policy refinement algorithms to boost the quality of mined policy. Policy refinement algorithms are useful in policy maintenance as well. Finally, we implement a prototype of the proposed approach to demonstrate its feasibility, efficiency and effectiveness.

CCS Concepts

•**Security and privacy** → **Access control; Authorization;**

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SACMAT '19 June 4–6, 2019, Toronto, Canada

© 2019 ACM. ISBN 123-4567-24-567/08/06.

DOI: 10.475/123_4

Keywords

Access Control Policy; Policy Mining; Attribute Based Access Control; Relationship Based Access Control

1. INTRODUCTION

Access control systems are a critical component of an information system protecting information resources from unauthorized accesses. However, with the rapid advances in newer computing and information technologies (e.g. social networks, IoT, cloud computing, etc.), existing access control (AC) models such as Discretionary Access Control (DAC) [25, 9], Mandatory Access Control (MAC) [3, 23], and Role-Based Access Control (RBAC) [24] have become inadequate in providing flexible and expressive authorization services [7]. For example, a health care 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 [14, 15]. *Attribute Based Access Control* (ABAC) models present a promising approach that addresses newer challenges in emerging applications [10]. An ABAC approach grants access rights to users based on attributes of entities in the system (e.g. user attributes, object attributes, environmental conditions, etc.).

To capture flexible access requirements of social network systems, *Relationship Based Access Control* (ReBAC) model has been introduced to enforce authorization based on the relationships (e.g., friendship relation) between entities in a system [7]. Combining ReBAC with an ABAC approach can increase policy expressiveness and support a more flexible and diverse set of access policies. An access rule such as "*Only my doctor can see my lab results.*" can be supported by such an approach.

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 huge user populations that are highly distributed across disparate computing environments including legacy systems and the latest ones. Each of these computing environments may have their own AC model. 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 by helping organizations to cut the cost, time, and error of policy development. Policy mining algorithms ease the migration to more recent authorization frameworks 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 [17]. Following that, several researchers have investigated various role mining techniques [26, 19, 31]. Although the proposed approaches are beneficial in developing optimal sets of roles, they are not applicable in extracting ABAC or ReBAC policies.

Xu and Stoller were the first to study the problem of mining ABAC policies given different sets of information (e.g RBAC policy, logs, etc.) [32, 33, 34]. Their approach employs some heuristics to merge and simplify ABAC policies. Bui et al. propose an algorithm for extracting a ReBAC policy as an object-oriented ABAC policy with path expressions [5].

These studies suffer from several limitations: first, as their algorithms are based on heuristic approaches, their proposed techniques work very well for simple and small-sized AC policies; however, as the number of rules in the policy and the number of elements in each rule increases, the proposed algorithms become inefficient; especially as discussed in Section 5, their F-measure decreases dramatically when the complexity/size of the original policy increases. As a result, for a real environment with hundreds of policy rules along with thousands of entities with a huge number of attributes, these approaches may not result in a high quality mined policy.

Second, these algorithms only support positive attributes and relations (e.g. $\langle department, cs \rangle$) while in the modern and advance information systems, the need for employing negative attributes and relations (e.g. $\langle department, !ee \rangle$) is obvious. For example, in RBAC, *Separation of Duty* (SoD) constraint is enforceable using negative relations. The policy AC rule can use the relation condition $\langle purchaser, !approver \rangle$ to ensure that the same user cannot purchase an item and approve the purchase too.

Lastly, the similarity metrics that are used in these works for evaluating the correctness of the mined policy are only suitable for the synthesized cases when the original ABAC or ReBAC policy is available. In reality and when the original policy is based on a different AC model (e.g. RBAC), such similarity metrics are not applicable. For such a case, we need a separate similarity metric to evaluate the correctness of the mined policy.

In this paper, we propose an unsupervised learning based approach for mining ReBAC access control rules from an access log data. As we don't have an access to a real world ReBAC access log data, we build our model on synthetic data. Our unsupervised learning algorithm is trained on access logs and tries to detect patterns between those records. These patterns are used to extract attribute expressions as well as relation conditions of the corresponding AC policy. After the first phase of policy rule extraction, we achieve a policy which may not be as accurate and concise as we desire. We improve the quality of the mined policy through iterations of policy improvement steps including rule pruning and policy refinement algorithms. It's important to be able to evaluate the mined policy in each step. For this purpose, we propose the *policy quality metric* which incorporate the correctness of the policy (based on its relative F-measure to the original policy) and the policy complexity (based on its relative loss of complexity to the worst-case policy) to measure the policy quality. We have implemented a prototype of our proposed algorithms to check its feasibility and compare it with previous approaches.

To summarize, the primary contributions of this paper are as follows:

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

2. 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.

3. We propose the *policy quality metric* based on the policy correctness and its conciseness to be able to compare different sets of mined policy rules and to select the best one based on our criteria.

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

To the best of our knowledge, our proposed approach is the first unsupervised learning based ReBAC policy mining method that can be applied 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 if rules in the policy and the size of filters in each rule.

The rest of the paper is organized as follows. In Section 2, we overview the ReABAC model and its policy language as well as an unsupervised learning algorithm. Section 3 defines the ReABAC policy extraction problem, discuss the related challenges, and 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 7.

2. PRELIMINARIES

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

2.1 Relationship Based Access Control

Relationship-based access control (ReBAC) was first introduced for access control in online social networks (OSNs). In ReBAC, authorization policies are specified based on the relationships between entities that can be expressed by assertions. For example, an assertion "father(John, Peter)" specifies that Peter is John's father [7]. In this paper, for extracting relations in our unsupervised learning algorithm, we show each relation as an attribute of an entity in the system. For example "*u_father*" is a user attribute and in the previous example, its value for user "*John*" is "*Peter*".

2.2 ReABAC Model

According to NIST's "*Guide to ABAC Definition and Consideration*" [10], "*the ABAC engine can make an access control decision based on the assigned attributes of the requester, 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, respectively.

Accordingly, U, O, and S are sets of users, objects, and sessions in the 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 NIST Guide [10]. $E = U \cup O \cup S$ and $A = A_u \cup A_o \cup A_s$ are the sets of all entities and all attributes in the system, respectively.

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

Definition 2. ***(Attribute Function)****. [34] Given an entity $e \in E$, 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. Each attribute function maps elements in E to atomic or set values.

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

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

Example 3. *Tuple $\langle label, !top\text{-}secret\rangle$ 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 $e \in E$ satisfies an attribute filter $\mathcal{F}$, denoted as $e \models \mathcal{F}$, iff*

$$\forall \langle a_i, v_i\rangle \in \mathcal{F} : v_i \subseteq f_{a\_e}(e, a_i).$$

Example 4. *Suppose $A_u = \{dept, position, courses\}$. The set of tuples $\mathcal{F_U} = \{\langle dept, \{CS\}\rangle, \langle position, \{grad, ugrad\}\rangle\}$ denotes a user attribute filter. Here, the users in the CS department who are either graduate or undergraduate students satisfy $\mathcal{F_U}$.*

Definition 5. ***(Relation Condition)****. A* relation condition *is defined as a set of tuples $\mathcal{R} = \{\langle a, b\rangle | \, a, b \in A \wedge a \neq b \wedge V_a = V_b\}$.*

A relation is used in a rule to denote the equality condition between two attributes of users, objects, or sessions. We should note that the two attributes in the relation condition must have the same range.

Definition 6. ***(Relation Condition Satisfaction)****. An entity $e \in E$ satisfies a relation condition $\mathcal{R}$, denoted as $e \models \mathcal{R}$, iff*

$$\forall \langle a_i, b_i\rangle \in \mathcal{R} : f_{a\_e}(e, a_i) = f_{a\_e}(e, b_i).$$

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

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

The decision d of an access right tuple can be *permit* or *deny*. The access right tuple with *permit* decision means that user u can perform an operation op on an object o and can exercise it in session s. The access right tuple with *deny* decision means that user u cannot perform operation op on object o through session s.

Access log is a union of *Positive Access Log*, $\mathcal{L}^+$, and *Negative Access Log*, $\mathcal{L}^-$, where:

$$\mathcal{L}^+ = \{\langle q, d\rangle | \langle q, d\rangle \in \mathcal{L} \wedge d = permit\},$$

and

$$\mathcal{L}^- = \{\langle q, d\rangle | \langle q, d\rangle \in \mathcal{L} \wedge d = deny\}.$$

Definition 9. ***(ReABAC Rule)****. An* access rule *ρ is a tuple $\langle \mathcal{F_U}, \mathcal{F_O}, \mathcal{F_S}, \mathcal{R}, op, d\rangle$, where $\mathcal{F_U}$ is a user attribute filter, $\mathcal{F_O}$ is an object attribute filter, $\mathcal{F_S}$ is a session attribute filter, $\mathcal{R}$ is a relation condition, op is an operation, and d is the decision of the rule ($d \in \{permit, deny\}$).*

Example 5. *The rule $\rho_1 = \langle\{\langle position, \{student\}\rangle\}, \{\langle location, \{campus\}\rangle\}, \{\langle type, \{article\}\rangle\}, \{\langle dept_u, dept_o\rangle\}, read, permit\rangle$ 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 right tuple $t = \langle q, d\rangle$ with $q = \langle u, o, s, op\rangle$ is said to satisfy a rule ρ, denoted as $t \models \rho$, iff*

$$u \models \mathcal{F_U} \wedge o \models \mathcal{F_O} \wedge s \models \mathcal{F_S} \wedge op = op_\rho \wedge d = d_\rho.$$

Definition 11. ***(ReABAC Policy)****. An ABAC policy is a tuple $\pi = \langle E, OP, A, f_{a\_e}, \mathcal{P}\rangle$ where E, OP, A, 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. ***(ReABAC Policy Decision)****. The decision of an ABAC policy π for an access request q denoted as $d_\pi(q)$ is* permit *iff:*

$$\exists \rho \in \pi \langle q, permit\rangle \models \rho$$

otherwise, the decision is deny*.*

Definition 13. ***(Relative False Positive Rate)****. Given an access log $\mathcal{L}$ and an ABAC policy π, the relative false positive rate of π regarding $\mathcal{L}$ denoted as $FP_{\pi|\mathcal{L}}$ is the portion of negative access logs that the decision of π for them is* permit*:*

$$FP_{\pi|\mathcal{L}} = \frac{|\{\langle q, d\rangle \in \mathcal{L}^- | d_\pi(q) = permit\}|}{|\mathcal{L}^-|}$$

Here, $|s|$ is the cardinality of set s.

Definition 14. ***(Relative False Negative Rate)****. The relative false negative rate of π regarding $\mathcal{L}$ denoted as $FN_{\pi|\mathcal{L}}$ is the portion of positive access logs that the decision of π for them is* deny*:*

$$FN_{\pi|\mathcal{L}} = \frac{|\{\langle q,d\rangle \in \mathcal{L}^+ | d_\pi(q) = deny\}|}{|\mathcal{L}^+|}$$

Similarly, we calculate the relative true positive rate and true negative rate of π regarding $\mathcal{L}$ denoted as $TP_{\pi|\mathcal{L}}$ and $TN_{\pi|\mathcal{L}}$ as follows:

$$TP_{\pi|\mathcal{L}} = \frac{|\{\langle q,d\rangle \in \mathcal{L}^+ | d_\pi(q) = permit\}|}{|\mathcal{L}^+|}$$

$$TN_{\pi|\mathcal{L}} = \frac{|\{\langle q,d\rangle \in \mathcal{L}^- | d_\pi(q) = deny\}|}{|\mathcal{L}^-|}$$

Definition 15. ***(Relative Accuracy)****. Given the relative true positive and negative rates, the relative accuracy of π regarding $\mathcal{L}$ denoted as $ACC_{\pi|\mathcal{L}}$ is calculated as follows:*

$$ACC_{\pi|\mathcal{L}} = \frac{TP_{\pi|\mathcal{L}} + TN_{\pi|\mathcal{L}}}{TP_{\pi|\mathcal{L}} + TN_{\pi|\mathcal{L}} + FP_{\pi|\mathcal{L}} + FN_{\pi|\mathcal{L}}}$$

2.3 Unsupervised Learning Algorithm

Unsupervised learning algorithms try to infer a function that describes the structure of unlabeled data. Unsupervised learning is 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 access right 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 access right tuples to a set of clusters that are representative of the desired ABAC rules. Such a mapping can be expressed as a function, $h : \mathcal{X} \rightarrow \mathcal{Y}$, where:

1. $\mathcal{X}$ is a set of access right 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 π).

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 ReABAC Policy Extraction Problem

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

3.2 Challenges

One of the main challenges in mining an access control policy is determining the number of rules in the policy. In an unsupervised learning approach, this challenge corresponds to determining the number of clusters, k. In Section 4.3, we explain the method we use in our approach to overcome such a challenge.

Another key challenge in policy engineering is how to ensure that the mined policy represents the original policy. In other words, we need a set of metrics to evaluate the quality of the mined policy. In Section 3.3, we propose a policy quality metric that can be used to evaluate the correctness and conciseness of the mined policy.

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 of the original policy for that set of requests. We use the relative accuracy metric, $ACC_{\pi|\mathcal{L}}$, to measure the accuracy of mined policy π with regards to the decisions made by the original policy indicated by $\mathcal{L}$. As accuracy may be misleading in unbalanced data sets [1] (which is very probable in case of access logs), we use **relative F-score** to better evaluate the mined policy:

$$F\text{-}score_{\pi|\mathcal{L}} = 2 \cdot \frac{Precision_{\pi|\mathcal{L}} \cdot Recall_{\pi|\mathcal{L}}}{Precision_{\pi|\mathcal{L}} + Recall_{\pi|\mathcal{L}}}$$

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 [20] and later extended for ABAC policies [34]. WSC is consistent with usability studies of access control rules, which indicates that the more concise the policies are the more manageable they become [2]. Informally, for a given ABAC policy, its WSC is a weighted sum of the number of its elements. Formally, for an ABAC policy π with rules $\mathcal{P}$, its WSC is defined as follows:

$$WSC(\pi) = WSC(\mathcal{P})$$

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

$$WSC(\rho = \langle \mathcal{F_U}, \mathcal{F_O}, \mathcal{F_S}, \mathcal{R}, op, d\rangle) = w_1 WSC(\mathcal{F_U}) + w_2 WSC(\mathcal{F_O}) + w_3 WSC(\mathcal{F_S}) + w_4 WSC(\mathcal{R})$$

$$\forall s \in \{\mathcal{F_U}, \mathcal{F_O}, \mathcal{F_S}, \mathcal{R}\} : WSC(s) = \sum |s|$$

where $|s|$ is the cardinality of set s and the w_i are user-specified weights.

Van Rijsbergen's proposed an effectiveness measure for combining two different metrics P and R as follows [22]:

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

To be able to compare the quality of different mined ReABAC policies, we combine the two metrics based on Van Rijsbergen's effectiveness measure [22] and define the **Policy Quality Metric** as follows:

$$\mathcal{Q}_\pi = (\frac{\alpha}{F\text{-}score_{\pi|\mathcal{L}}} + \frac{1-\alpha}{\Delta WSC_\pi})^{-1}$$

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

$$\Delta WSC_\pi = \frac{WSC_{max} - WSC(\pi) + 1}{WSC_{max}}$$

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

Definition 16. ***(Worst-case Mined Policy)****. Worst-case mined policy π_w is the policy that is extracted by iterating through access log $\mathcal{L}$ and adding an access control rule for each access right tuple if it's not already included in the mined policy. The corresponding rule for each access right tuple includes all attributes of user, object, and subject of that access right tuple.*

The worst-case mined policy has F-score equal to one (if the access log is complete, which means that every possible combination of attribute values are presented in the access log) while on the other hand it has a huge complexity value and as a result, its policy quality is very low. Considering the equal importance of relative F-score and relative loss of complexity the policy, we calculate the quality measure as follows:

$$\mathcal{Q}_\pi = \frac{2 \cdot F\text{-}score_{\pi|\mathcal{L}} \cdot \Delta WSC_\pi}{F\text{-}score_{\pi|\mathcal{L}} + \Delta WSC_\pi}$$

A mined policy with 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 $\mathcal{Q}_\pi$.

For worst-case mined policy π_w, $\Delta WSC_{\pi_w} \approx 0$, so its policy quality $\mathcal{Q}_{\pi_w}$ is very close to zero.

For an empty mined policy π_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 $\mathcal{Q}_{\pi_e}$ 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. OUR PROPOSED APPROACH

In this section, we present the proposed methodology to extract ReBAC policies from a set of the access log. First, we show how to prepare data for our algorithm. Next, we explain the policy rule extraction method. Finally, we show how to improve the extracted policy to get a policy with higher quality.

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 need to handle *missing values* in this step, as well. 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 *UNK* (i.e. unknown).

4.2 Selection of Learning Algorithm

K-modes algorithm [6] is a well known unsupervised learning algorithm used for clustering categorical data. *K-modes* has been proved effective in mining ABAC[16]. The algorithm uses an initialization method based on the distance between data points and the density of data points as well. Using both density and distance when initializing clusters help avoid two problems: clustering outliers as new clusters based on the distance only, and creating new clusters in the surrounding of one center based on the density only. Comparing with random initialization method, this method provides more robustness and better accuracy in the clustering process[6].

4.3 Parameter Tuning

In *K-modes algorithm*, the value for k is important for the coverage of local optima. To find the optimal k, we train the model with different k values and select the k value when there is no cluster reassignment or when the cost is minimum. The cost is defined as follows:

$$E = \sum_{l=1}^{k} \sum_{i=1}^{n} y_{il} d(X_i, Q_l)$$

where d is a similarity measure defined as the square Euclidean distance, X_i denotes a set of n data points, Q_l is a vector of a cluster, and y_{il} is an element of the partition matrix Y [11].

While finding an optimal k is important for clustering the access logs, our policy improvement algorithms discussed in Section 4.5 are designed to help in achieving the best number of policy rules that are as concise as possible while providing an accurate representation of the original policy.

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 access right 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 rule. We adopt the rule extraction algorithm in [16] and extend it to extract both positive and negative attribute filters. Based on the idea of *effective attribute* in [16], we define *effective positive attribute* and *effective negative attribute* as follows:

Definition 17. ***(Effective Positive (Negative) Attribute)****. Let $S = \{\langle a, v \rangle\}$ be the set containing 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 ρ_i corresponding to cluster C_i, where the frequency of occurrence of*

v_j in the set of all the records of cluster C_i is much higher (lower) than its frequency of occurrence in the original data (This is determined based on a threshold $\mathcal{T}_P$ ($\mathcal{T}_N$)). 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 ρ_i corresponding to C_i .

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

DEFINITION 18. ***(Effective Positive (Negative) Relation).*** *Let $R = \{\langle a, b\rangle\}$ 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 *pairs of ρ_i corresponding to cluster C_i, where the correlations between a_j and b_j in all records of cluster C_i is much higher (lower) than their correlation in the original data (This is determined based onbased on a threshold θ_P (θ_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 ρ_i corresponding to this cluster.*

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

Algorithm 1 Effective attribute extraction algorithm

```
 1: procedure EXTRACTATTRIBUTEFILTERS
Input: $C_i$, $A$, $V$, $\mathcal{L}$, $\mathcal{T}_P$, $\mathcal{T}_N$
Output: $\mathcal{F}$
 2:     $\mathcal{F} \leftarrow \emptyset$
 3:     for all $a_j \in A$ do
 4:         for all $v_j \in V_{a_j}$ do
 5:             if $Freq(v_j, C_i) - Freq(v_j, \mathcal{L}) > \mathcal{T}_P$ then
 6:                 $\mathcal{F}i \leftarrow \mathcal{F} \cup \langle a_j, v_j\rangle$
 7:             end if
 8:             if $Freq(v_j, \mathcal{L}) - Freq(v_j, C_i) > \mathcal{T}_N$ then
 9:                 $\mathcal{F}i \leftarrow \mathcal{F} \cup \langle a_j, !v_j\rangle$
10:             end if
11:         end for
12:     end for
        return $\rho_i$
13: end procedure
```

Algorithm 2 Effective relation extraction algorithm

```
 1: procedure EXTRACTRELATIONS
Input: $C_i$, $A$, $\mathcal{L}$, $\theta_P$, $\theta_N$
Output: $\mathcal{R}$
 2:     $\mathcal{R} \leftarrow \emptyset$
 3:     for all $a_j \in A$ do
 4:         for all $b_j \in A$ and $b_j \neq a_j$ do
 5:             if $Corr(a_j, b_j, C_i)$ - $Corr(a_j, b_j, \mathcal{L})$>$\theta_P$ then
 6:                 $\mathcal{R} \leftarrow \mathcal{R} \cup \langle a_j, b_j\rangle$
 7:             end if
 8:             if $Corr(a_j, b_j, \mathcal{L})$ - $Corr(a_j, b_j, C_i)$>$\theta_N$ then
 9:                 $\mathcal{R} \leftarrow \mathcal{R} \cup \langle a_j, !b_j\rangle$
10:             end if
11:         end for
12:     end for
        return $\mathcal{R}$
13: end procedure
```

4.5 Policy Improvements

After the first phase of policy rule extraction, we get a policy which may not be as accurate and concise as we desire. We improve 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 corresponding 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 [13] as follows:

$$J(S_1, S_2) = |S_1 \cap S_2|/|S_1 \cup S_2|$$

So the similarity between two rules ρ_1 and ρ_2 is calculated as follows:

$$J(\rho_1, \rho_2) = \frac{[\sum_{\mathcal{F}\in\{\mathcal{F}_\mathcal{U}, \mathcal{F}_\mathcal{O}, \mathcal{F}_\mathcal{S}\}} |\mathcal{F}_{\rho_1} \cap \mathcal{F}_{\rho_2}| + |\mathcal{R}_{\rho_1} \cap \mathcal{R}_{\rho_2}| + |op_{\rho_1} \cap op_{\rho_2}|]}{[\sum_{\mathcal{F}\in\{\mathcal{F}_\mathcal{U}, \mathcal{F}_\mathcal{O}, \mathcal{F}_\mathcal{S}\}} |\mathcal{F}_{\rho_1} \cup \mathcal{F}_{\rho_2}| + |\mathcal{R}_{\rho_1} \cup \mathcal{R}_{\rho_2}| + |op_{\rho_1} \cup op_{\rho_2}|]}$$

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.

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:*

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

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

Here ρ_2 is more restricted than ρ_1 as it imposes more conditions on the user attributes.

Having such a restricted rule in the mined policy would result in larger number of *FNs* 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. Back to Example 6, ρ_1 is more relaxed compared to ρ_2. Such a relaxed rule would result in more *FPs* as it permits access requests that should be denied based on the original policies.

Algorithm 3 Rule Pruning algorithm

```
 1: procedure ruleP RUNING
Input: $\pi$
Output: $\pi$
 2:     $\mathcal{P} \leftarrow \pi.\mathcal{P}$
 3:     $q \leftarrow$ calcQuality($\mathcal{P}$)
 4:     for all $\rho_i \in \mathcal{P}$ do
 5:         for all $\rho_j \in \mathcal{P}$ and $\rho_i \neq \rho_j$ do
 6:             if Similarity($\rho_i, \rho_j$) $> 0.5$ then
 7:                 $\mathcal{P}_i \leftarrow \mathcal{P}/\rho_i$
 8:                 $\mathcal{P}_j \leftarrow \mathcal{P}/\rho_j$
 9:                 $q_i \leftarrow$ calcQuality($\mathcal{P}_i$)
10:                 $q_j \leftarrow$ calcQuality($\mathcal{P}_j$)
11:                 if $q_i >= q$ and $q_i >= q_j$ then
12:                     $\mathcal{P} \leftarrow \mathcal{P}_i$
13:                 end if
14:                 if $q_j >= q$ and $q_j >= q_i$ then
15:                     $\mathcal{P} \leftarrow \mathcal{P}_j$
16:                 end if
17:             end if
18:         end for
19:     end for
        return $\mathcal{P}$
20: end procedure
```

To address these issues, we propose a *policy refinement* procedure which is shown in Algorithm 4. Here, we try to refine the mined policy (π_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 π_{FN} and π_{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 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 π_m that is similar to a rule from π_{FN} or π_{FP} based on the Jaccard similarity (Section 4.5.1) is selected and will be refined as follows:

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

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

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

  where $\mathcal{F} = \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$. So, the extra filters are removed from the restricted rule (ρ_j).

- if there is no such rule, then ρ_i is the missing rule and we add it to π_m.

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

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

where $\mathcal{F} = \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$ includes all filters in the rule. So, the missing filters are added to the relaxed rule (ρ_j).

These refinements can be done in multiple iterations until further refinement does not give a better model in terms of policy quality $\mathcal{Q}_\pi$.

Algorithm 4 Policy refinement algorithm

```
 1: procedure refinePolicy
Input: $A$, $\mathcal{L}$
Output: $\pi_m$
 2:     $\mathcal{FN} \leftarrow$ getFNs($\pi_m, \mathcal{L}$)
 3:     $\pi_{FN} \leftarrow$ extractPolicy($\mathcal{FN}$)
 4:     for all $\rho_i \in \pi_{FN}.\mathcal{P}$ do
 5:         $R_s \leftarrow$ getSimilarRules($\pi_{FN}.\mathcal{P}, \pi_m.\mathcal{P}$)
 6:         if $|R_s| = 0$ then
 7:             $\pi_m.\mathcal{P} \leftarrow \pi_m.\mathcal{P} \cup \rho_i$
 8:         else
 9:             for all $\rho_j \in R_s$ do
10:                 for all $\mathcal{F} \in \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$ do
11:                     $\mathcal{F}_{\rho_j} \leftarrow \mathcal{F}_{\rho_j} \backslash (\mathcal{F}_{\rho_j} \backslash \mathcal{F}_{\rho_i})$
12:                 end for
13:             end for
14:         end if
15:     end for
16:     $\mathcal{FP} \leftarrow$ getFPs($\pi_m, \mathcal{L}$)
17:     $\pi_{FP} \leftarrow$ extractPolicy($\mathcal{FP}$)
18:     for all $\rho_i \in \pi_{FP}.\mathcal{P}$ do
19:         $R_s \leftarrow$ getSimilarRules($\pi_{FP}.\mathcal{P}, \pi_m.\mathcal{P}$)
20:         if $|R_s|! = 0$ then
21:             for all $\rho_j \in R_s$ do
22:                 for all $\mathcal{F} \in \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$ do
23:                     $\mathcal{F}_{\rho_j} \leftarrow \mathcal{F}_{\rho_j} \cup (\mathcal{F}_{\rho_i} \backslash \mathcal{F}_{\rho_j})$
24:                 end for
25:             end for
26:         end if
27:     end for
        return $\pi_m$
28: end procedure
```

5. EXPERIMENTAL EVALUATION

To evaluate the effectiveness of the proposed approach, we conduct our experiments on several different sets of access control logs. We have generated and used synthetic data for this purpose as real access control data sets are not available. The synthesized data is generated based on the systematic access control policies proposed in [34].

We compare our work with the previously proposed approach by Xu and Stoller [34] which we refer to as *XS work*. It's important to note that the proposed method by Xu et al. is only capable of extracting positive attributes and relations and their proposed technique works very well for simple and small scale AC policies, however, as the number of rules in the policy, the number of elements in each rule, the number of attributes, and the size of attribute ranges increase, the proposed algorithm does not perform well. To be able to compare our work with their approach, we have considered different sets of policies, a simple policy with positive expressions, a more complex policy with positive expressions and a policy with both positive and negative expressions. All of these policies are adapted from the *University* and *Healthcare* policies in [34].

5.1 Datasets

We have carried our experiments on four sample policy access logs that were adapted based on *University* and *Healthcare* policies from [34]. The *University* policy controls access of different users (e.g. students, instructors, teaching assistants,etc.) to various objects (applications, gradebooks, etc.). The *Healthcare* policy controls access by different users (e.g. nurses, doctors, etc.) to electronic health records (EHRs) and EHR items. The first two policies, which we refer to as `UniversityP` and `HealthcareP` are simple policies with only positive attribute expressions and fewer rules, attributes, and attribute ranges. The second set of policies which are referred to as `ComplexUniversityPN` and `ComplexHealthcarePN` are complex policies with both positive and negative expressions.

5.2 Implementation

In the rule extraction step, first we find initial rules by using k-modes algorithm on access control logs. After tuning k, we find the effective positive and negative attributes in the rules. To find effective relations we need to find the actual correlation between the attributes in data. Since our data sets are categorical, we use an approach from [4] to calculate correlation. The correlation function uses Nominal Association in calculating the correlation between attributes. In calculating correlation, only attributes that have same ranges are considered (e.g., $crs == crstaken$, $dept_u! == dept_o$). An effective positive or negative relation has to pass the defined threshold. A threshold is set based on the difference between the attribute correlation scores in the original data and the cluster. The policy extraction algorithm is written in Python (Python 3.5).

Table 1 shows the details related to the sizes of the sample policies. $|\mathcal{P}|$ is the number of rules in the policy, $|A|$ is the size of all attributes, $|V|$ is the size of all attribute values, $WSC(\pi)$ is the complexity of the policy, and $|\mathcal{L}|$ is the size of the records in the access log.

5.3 Results

As shown in Table 2, our proposed method is very effective in extracting all sample policies. For sample policies `UniversityP` and `HealthcareP` which include only positive filters, our method was able to extract sets of rules similar to those of the original policy. For sample policies `ComplexUniversityPN` and `ComplexHealthcarePN` with both positive and negative expressions, our method achieved the f-score of 0.91 and 0.75 with $WSC = 59$ and $WSC = 80$, respectively. While XS work was effective on simple policies `UniversityP` and `HealthcareP`, with more complex policies including both positive and negative expressions, their work didn't provide any output.

Figure 1 shows the FN and FP rates as well as the f-score of the model after each round of refinement for mining `ComplexHealthcarePN` policy. As shown in the figure, after multiple rounds of policy refinement, the F-score of the model was improved from 0.58 to 0.75. We should note that each round of refinement would not reduce both FN and FP

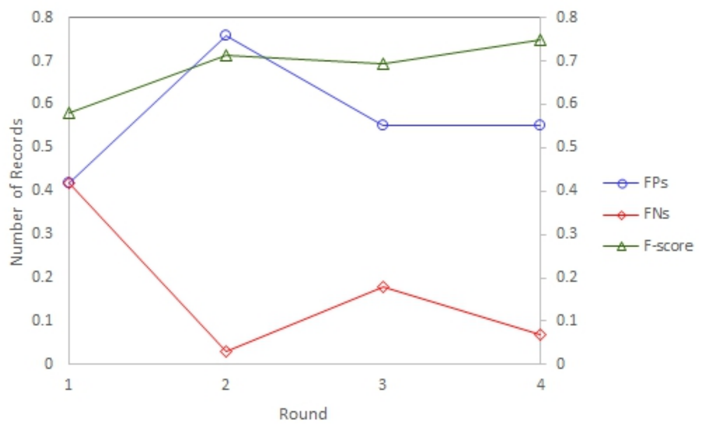

Figure 1: The Improvement of Healthcare Policy Mining Model After Multiple Rounds of Refinement

rates, but ultimately after few rounds of refinement, both rates would be decreased and the f-score would be increased.

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 [20, 31, 17, 26, 28, 30, 35, 8, 19, 27, 21]. 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 [17], 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 [26]. 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)) [21]. 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 [29]. 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 [32], logs [33], and access control list [34] plus attribute information. Their policy mining algorithms iterate over access control tuples (generated

Table 1: Sizes of the Sample Policies

π	$\|\mathcal{P}\|$	$\|A\|$	$\|V\|$	$WSC(\pi)$	$\|\mathcal{L}\|$
UniversityP	8	9	45	26	8000
HealthcareP	9	13	51	34	9000
ComplexUniversityPN	13	9	46	48	13000
ComplexHealthcarePN	9	13	81	75	9000

Table 2: Comparison of our proposed algorithm with XS work [34]

π	$WSC(\pi_m)$		$F\text{-}score_{\pi_m}$		$\mathcal{Q}_m$	
	XS work	Proposed Work	XS work	Proposed Work	XS work	Proposed Work
UniversityP	26	26	1	1	≈ 1	≈ 1
HealthcareP	34	34	1	1	≈ 1	≈ 1
ComplexUniversityPN	-*	59	-	0.9	-	0.95
ComplexHealthcarePN	-*	80	-	0.75	-	0.86

* XS work [34] did not generate any output for complex policies.

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 Stroller's proposed method, Medvet et al. [18] and Bui. et al. [5] propose evolutionary approaches for mining ABAC/ReBAC policies with similar issues as mentioned earlier.

Iyer and Masoumzadeh [12] 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.

7. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an unsupervised learning based approach to automate ReABAC 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 consider 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 [34] 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.

8. REFERENCES

[1] Accuracy paradox, Nov 2018.

[2] M. Beckerle and L. A. Martucci. Formal definitions for usable access control rule sets from goals to metrics. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, page 2. ACM, 2013.

[3] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, MITRE CORP BEDFORD MA, 1973.

[4] W. Bergsma. A bias-correction for cramérâĂŹs v and tschuprowâĂŹs t. *Journal of the Korean Statistical Society*, 42(3):323–328, 2013.

[5] T. Bui, S. D. Stoller, and J. Li. Mining relationship-based access control policies. pages 239–246, 2017.

[6] F. Cao, J. Liang, and L. Bai. A new initialization method for categorical data clustering. *Expert Systems with Applications*, 36(7):10223–10228, 2009.

[7] P. W. Fong and I. Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 51–60. ACM, 2011.

[8] Q. Guo, J. Vaidya, and V. Atluri. The role hierarchy mining problem: Discovery of optimal role hierarchies. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 237–246. IEEE, 2008.

[9] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[10] 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*, 800(162), 2013.

[11] Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the*

1st pacific-asia conference on knowledge discovery and data mining,(PAKDD), pages 21–34. Singapore, 1997.

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

[13] P. Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[14] 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*, pages 125–134. ACM, 2009.

[15] 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*, pages 359–368. IEEE, 2017.

[16] L. Karimi and J. Joshi. An unsupervised learning based approach for mining attribute basedaccess control policies. In *Big Data (Big Data), 2018 IEEE International Conference on*. IEEE, 2018.

[17] 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*, pages 179–186. ACM, 2003.

[18] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari. Evolutionary inference of attribute-based access control policies. In *EMO (1)*, pages 351–365, 2015.

[19] I. 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*, pages 21–30. ACM, 2008.

[20] 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)*, 13(4):36, 2010.

[21] Q. Ni, J. Lobo, S. Calo, P. Rohatgi, and E. Bertino. Automating role-based provisioning by learning from examples. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 75–84. ACM, 2009.

[22] C. J. v. Rijsbergen. *Information retrieval. 2.ed.* Butterworths, 1979.

[23] R. S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, 1993.

[24] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[25] R. S. Sandhu and P. Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[26] J. Schlegelmilch and U. Steffens. Role mining with orca. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 168–176. ACM, 2005.

[27] H. Takabi and J. B. Joshi. Stateminer: an efficient similarity-based approach for optimal mining of role hierarchy. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, pages 55–64. ACM, 2010.

[28] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 175–184. ACM, 2007.

[29] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: A formal perspective. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):27, 2010.

[30] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 144–153. ACM, 2006.

[31] Z. Xu and S. D. Stoller. Algorithms for mining meaningful roles. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 57–66. ACM, 2012.

[32] Z. Xu and S. D. Stoller. Mining attribute-based access control policies from rbac policies. In *Emerging Technologies for a Smarter World (CEWIT), 2013 10th International Conference and Expo on*, pages 1–6. IEEE, 2013.

[33] 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*, pages 276–291. Springer, 2014.

[34] Z. Xu and S. D. Stoller. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 12(5):533–545, 2015.

[35] D. Zhang, K. Ramamohanarao, and T. Ebringer. Role engineering using graph optimisation. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 139–144. ACM, 2007.