Mining Access Control Policies from Logs via Attribute-Based Categories

Anna Bamberger[0000−0002−7389−8988] and Maribel Fernández[0000−0001−8325−5815]

Department of Informatics, King’s College London
anna.bamberger@kcl.ac.uk
https://www.kcl.ac.uk/informatics

Abstract. We present an algorithm to mine access control policies from logs based on attribute-based definitions of categories. We derive category definitions from attributes associated with principal and resource entities in access control logs, to our best knowledge, the first such mechanism explored. Preliminary experimental results using a proof-of-concept implementation show that the miner provides an initial foundation for the construction of dynamic policies derived from categories. The objective of this system is to respond to changes impacting access control decisions on the fly. Our category-based model aims to represent a dynamic system whose goal is to strengthen security, facilitate ease of use and provide greater flexibility when dealing with complex organisational networks. Finally, our proposed policy miner aims to tie flexibility with improved explainability of its resulting policies since it seeks to find meaningful connections between features rather than providing a simple feature aggregation.

Keywords: Access control · Policy Mining · Policy Administration · Category-Based Access Control.

1 Introduction

Access control has been the subject of a wide range of studies. Many of these studies have focused on RBAC, a stable, well-defined model combining characteristics of expressive policies with an easily manageable structure [34, 35, 3, 18]. As software systems and their complexities grew the aforementioned models, while still providing a reliable and well constructed access control architecture, did not impart a sufficiently large scope for including the intricacies displayed by the now larger systems. In light of such developments extensions of the RBAC model have been proposed [12–14]. The attribute-based access control (ABAC) model [23, 37, 41, 38, 39] introduces the notion of attribute as a dynamic addition to role labels known from RBAC. Further research has been conducted merging the existing RBAC-based system into ABAC [44] as well as proposals for a more generalised ABAC model subsuming RBAC and its affiliates [25]. The ABAC model and its descendants, despite displaying features of dynamicity, do not provide any mechanism to create meaningful connections between attributes. This
can lead to an overly large selection of rules which are difficult to understand, explain and apply.

It becomes evident that there is a need for a model which is easy to understand and maintain and which does not restrict itself to the utilisation of just one or even multiple static features. The category-based access control (CBAC) model [4, 7, 10] inherently provides such architecture via a skeleton of building blocks, from which the dynamically modifiable authorisations arise. The key components in CBAC are the categorisation of the main entities (principals and resources) and the linkage between categories and actions, from which the set of authorisations at each given point in time can be deduced.

Following the definition of the CBAC model [7, 8, 10] we are proposing both a proof of concept and implementation for the generation of CBAC policies.

Multiple policy creation mechanisms have been proposed [28, 36, 32, 39, 27]. These algorithms translate the theoretical access control model to actual access control policies. A connector, which we call access control policy generator, links information from the system to the generation of actual policies and employs as its executive step the policy mining mechanism. In this work we make use of an NLP Doc2vec pre-filter [31] followed by an FP-growth-based check [21] to build CBAC policies.

Since the CBAC model subsumes most of the access control models in use (including RBAC and ABAC [5, 16]) a mining mechanism for CBAC policies has the potential to be applicable in other models defined as CBAC instances, thus avoiding the proliferation of miners for specific models. Another advantage of a CBAC miner is the increase of both flexibility and autonomy of the access control system thereby reducing the maintenance burden.

Our main contributions are as follows:

1. We present a new approach to mine attribute-based CBAC policies from access control logs via a proof of concept as well as its initial practical implementation technique for the generation of CBAC policies. We discuss a running example on a hypothetical data set and compare it with the ABAC miner from [40] and [39].

2. We provide an extensive quality check mechanism via a validation step to ensure that our generated policies are compliant with the access logs from which they have been derived.

3. The proposed miner seeks to display both dynamic as well as explainable features for policy generation compared with other dynamic models such as ABAC, due to combining the NLP method with FP-growth.

Overview The rest of the paper is organised as follows: Section 2 recalls preliminary notions about CBAC and policy mining. Section 3 gives an overview of the policy mining algorithm. Section 4 provides implementation details. Section 5 validates the correctness and quality of the proposed method. Section 6 analyses related work with conclusions and suggestions for future work in section 7.
2 Preliminaries

2.1 The Category-Based Metamodel

A key feature of the CBAC metamodel [4] is the classification into categories of entities, together with sets of relationships between entities, and a set of axioms that the relationships should satisfy.

The following main generic sets of entities are included in the metamodel and are to be employed alongside and in addition to any application-dependent and/or environment-specific entities: a countable set $C$ of categories $c_0, c_1, \ldots$; a countable set $P$ of principals $p_0, p_1, \ldots$ (we assume that principals that request access to resources are pre-authenticated and in possession of their pre-authenticated credentials); a countable set $A$ of named actions $a_0, a_1, \ldots$; a countable set $R$ of resources $r_0, r_1, \ldots$; a finite set $A_{\text{uth}}$ of possible answers to access requests (e.g., grant, deny) and a countable set $S$ of situational identifiers to denote contextual information (situational identifiers might not be needed in some instances of the metamodel).

The assignment of entities to categories is specified via relationships between those entities and their categories, which can be defined extensionally (as a table) or intentionally (e.g., the tuples in a relationship may be obtained by executing a program). The metamodel includes the following generic relationships:

- Principal-Category Assignment, $\mathcal{PCA} \subseteq P \times C$, s.t. $(p, c) \in \mathcal{PCA}$ iff the principal $p \in P$ is in the category $c \in C$;

- Resource-Category Assignment, $\mathcal{RCA} \subseteq R \times C$, s.t. $(r, c) \in \mathcal{RCA}$ iff the resource $r \in R$ is in the category $c \in C$;

- Permissions, $\mathcal{ARCA} \subseteq A \times C \times C$, s.t. $(a, c_r, c_p) \in \mathcal{ARCA}$ iff the action $a \in A$ on resource category $c_r \in C$ can be performed by principals assigned to the category $c_p \in C$.

- Authorisations, $\mathcal{PAR} \subseteq P \times A \times R$, s.t. $(p, a, r) \in \mathcal{PAR}$ iff the principal $p \in P$ is allowed to perform the action $a \in A$ on the resource $r \in R$.

A reflexive-transitive relation $\subseteq$ between categories is also included (this can simply be equality). The flexible nature of aforementioned relationships permits but does not require their mapping to tables and/or the provisioning of functional table implementations. The list of relationships is not exclusive and can be revised to incorporate application-dependent relationships.

Authorisations are derived using the core axiom (a1):

$$(a1) \forall p \in P, \forall a \in A, \forall r \in R, ((\exists c_p \in C, \exists c'_p \in C, \exists c_r \in C, \exists c'_r \in C, (p, c_p) \in \mathcal{PCA} \land (r, c_r) \in \mathcal{RCA} \land c_p \subseteq c'_p \land c_r \subseteq c'_r \land (a, c'_r, c'_p) \in \mathcal{ARCA}) \Rightarrow (p, a, r) \in \mathcal{PAR})$$

Based on this axiom, we can deduce that if a principal $p$ is in the category $c_p$ (i.e., $(p, c_p) \in \mathcal{PCA}$), a resource $r$ is in the category $c_r$ (i.e., $(r, c_r) \in \mathcal{RCA}$), and the category $c_p$ is permitted to perform the action $a$ on resource category $c_r$ (i.e., $(a, c_r, c_p) \in \mathcal{ARCA}$) then $p$ is authorised to perform $a$ on $r$ (that is, $(p, a, r) \in \mathcal{PAR}$).
Inherited authorisations can be derived if hierarchies of categories are incorporated. Any derived request that is not authorised by an interaction of the underlying categories is assumed to be prohibited; alternatively, additional relations and axioms can be included to define prohibitions and to deal with more general kinds of answers [6].

Using the basic sets of entities and relationships of the metamodel, we can define a generic CBAC model: A **CBAC policy** is a tuple \( \langle E, Rel \rangle \) of entities and relationships that satisfy the axiom \((a1)\).

The classification of entities into categories can be statically defined by labels (e.g., using roles, which can only be updated by the administrator) or dynamic (e.g., a minor may change into the adult category once they passed their 18th birthday). Most of the existing access control models can be defined as instances of CBAC by selecting appropriate sets of entities and relationships and specifying adequate notions of categories [4, 5].

**Category-Based Specification of ABAC.** We now consider the attribute-based access control model ABAC [19] and its category-based specification C-ABAC [16], upon which our practical implementation is based. In C-ABAC, we include the generic sets of entities: \( \mathcal{P}, \mathcal{A}, \mathcal{R}, \mathcal{C} \), as well as a countable set \( \mathcal{E}_{\mathcal{N}} \) of environment entities and a countable set \( \mathcal{A}_{\mathcal{V}} \) of attributes ranging over values \( \mathcal{V} \). Attributes can assume a non-exhaustive list of values from strings, numbers and Boolean conditions with possible validity constraints (e.g., an age value cannot be negative) as well as a pre-defined range which the values must fit into. Attributes are further split into principal, resource and environmental attributes. The relations linking principals, resources and environment entities to their attributes are referred to as \( \mathcal{P}\mathcal{A}\mathcal{T}\mathcal{A}, \mathcal{R}\mathcal{A}\mathcal{T}\mathcal{A}, \mathcal{E}\mathcal{A}\mathcal{T}\mathcal{A} \) in [16]. In addition the following relationship is included in \( \text{Rel} \) in C-ABAC in to link categories with attributes:

- **Category-Attribute Assignment**, \( \mathcal{C}\mathcal{A}\mathcal{T}\mathcal{A} \subseteq \mathcal{C} \times \mathcal{C} \), which defines the attribute values required in each category:
  
  \[(c, \text{cond}) \in \mathcal{C}\mathcal{A}\mathcal{T}\mathcal{A} \iff \text{category } c \in \mathcal{C} \text{ requires attribute values satisfying the Boolean condition } \text{cond} \in \mathcal{C} \text{; we say that } c \text{ is defined by } \text{cond}. \text{ We write } \Gamma \vdash \text{cond} \text{ if } \Gamma \text{ specifies attributes and values that satisfy } \text{cond}. \text{ A reflexive-transitive relation } \subseteq \text{ between categories is also included. We assume } \subseteq \text{ is compatible with the conditions defining categories, i.e., } c \subseteq c' \text{ means that the defining condition for } c \text{ implies that of } c'.\]

  \[
  (c1) \forall c, c' \in \mathcal{C}, \forall \Gamma, ((\exists (c, d)(c', d') \in \mathcal{C}\mathcal{A}\mathcal{T}\mathcal{A}, \Gamma \vdash d) \iff (c \subseteq c'))
  \]

The following axioms state that an entity belongs to a category \( c \) if its attribute values satisfy \( c \)'s definition.

\[
(c1) \forall p \in \mathcal{P}, \forall c \in \mathcal{C}, ((p, c) \in \mathcal{P}\mathcal{C}\mathcal{A} \iff (\exists (p, l_p) \in \mathcal{P}\mathcal{T}\mathcal{A}\mathcal{T}\mathcal{A}, (l_p, \mathcal{R}\mathcal{A}\mathcal{T}\mathcal{A}, \mathcal{E}\mathcal{A}\mathcal{T}\mathcal{A} \vdash d)))
\]

\[
(c2) \forall r \in \mathcal{R}, \forall c \in \mathcal{C}, ((r, c) \in \mathcal{R}\mathcal{C}\mathcal{A} \iff (\exists (r, l_r) \in \mathcal{R}\mathcal{A}\mathcal{T}\mathcal{A}, (l_r, \mathcal{R}\mathcal{A}\mathcal{T}\mathcal{A}, \mathcal{E}\mathcal{A}\mathcal{T}\mathcal{A} \vdash d)))
\]
2.2 Policy mining

There is no previous study of policy mining for the CBAC model. Since CBAC relies on the concept of category to structure authorisations (principals and resources are categorised, and permissions are assigned to specific categories, from which the authorisation relation is derived), it is clear that a crucial step in the generation of CBAC policies is the definition of appropriate categories of principals and resources. In order to define a policy miner for CBAC, the first step is to define a language to express category definitions. ABAC policy mining algorithms have potential to significantly reduce the cost of migration to ABAC, by partially automating the development of an ABAC policy [39]. In this work, we will consider categories defined on the basis of attributes, i.e., we consider the C-ABAC model.

We define the C-ABAC policy mining problem as the problem of creating a C-ABAC policy, given a list of logs with accompanying attribute data, such that the \( PAR \) relation includes the authorisations in the input logs. The challenge is to create the \( CAT_A \) and \( ARC_A \) relations, i.e., the category definitions and the association between actions, resource categories and principal categories.

We propose to mine C-ABAC policies using an algorithm consisting of two stages. Initially we propose natural language processing to identify groups of sufficiently similar entities as per Doc2vec ([31], [29]). Subsequently we use the FP-growth mining algorithm ([21]) to identify frequently occurring patterns in access control logs based on the similarity groupings identified by the initial NLP stage. This second step is executed to ensure that member principals from similarly grouped categories do indeed have the same permissions with relation to the same resource categories. Should this not be the case diverging categories are split into separate groups.

For the initial NLP-based similarity assessment we chose the Doc2vec algorithm as introduced in [31], [29]. Doc2vec seeks to produce a numerical representation of the underlying document by producing vectors of words, similar to its foundation word2vec ([31]), the difference being that Doc2vec does not maintain a logical structure in the document but rather introduces another vector named document/paragraph ID to existing word vectors. This is the vector capturing the fundamental concept of the document. We chose Doc2vec specifically because of its ability to incorporate the additional document vector on top of the word vectors, so that with each iteration not only words but the entire document concept is updated which is an appropriate procedure to handle one consolidated set of logs.

The FP-growth algorithm [21, 22] has been chosen as a basis for the validation process completing the NLP-based category mining process due to a logical association of frequently occurring items with a common super class. The FP-growth method uses a bottom-up methodology: it chooses the least occurring prefix, i.e., the lowest leaf of the tree corresponding to the prefix, and builds a frequent pattern for this prefix adding items one-by-one and moving up towards the root. Items not meeting a minimum support are discarded (recall that the
support of a pattern \( A \), which is a set of items, is the number of transactions containing \( A \).

## 3 Policy Mining Algorithm Overview

**Challenges and proposed solution:** There are two key challenges we would like to tackle with our proposed algorithm for mining CBAC policies: Firstly, the algorithm must provide a foundation for flexible utilisation of its policy-defining features, i.e., categories, without forcing the application of any specific type of pre-defined elements in the construction of such categories. Secondly, while preserving this dynamic nature, the algorithm must still satisfy the requirements of decidability from which a measurable quality check for conformance with organisational governance can be inferred.

To handle these two challenges we present a hierarchy based on a preliminary NLP grouping which provides an initial clustering of entities without any pre-defined feature restrictions, thereby countering both the flexibility and decidability challenge. We subsequently build an elaborate quality check mechanism via FP-growth to confirm true authorisation matches amongst the newly generated categories.

For the initial construction of categories we use the Doc2vec algorithm [31], [29] to establish a category grouping based on the similarity of attributes of principal and resource entities in log transactions. We chose similarity scores as the initial determining factor of category formation because logically, shared association of certain features with certain access requests suggests that such features must be relevant in the evaluation of the request.

The second stage of the algorithm creation is the modelling of ARCA based on the confirmation of categories for principals and resources, respectively via FP-growth. The key idea here is that whilst the category label appears as a foundational designation of the policy framework any authorisation assignments derived from such categories must be validated to ensure conformance of all category members with authorisations and the opposite entities there authorisations have been linked to via the logs. It is this dynamic interaction connecting specific resources, principals and their associated actions which gives rise to the authoritative abstraction we refer to as \( \mathcal{PAR} \) which derives current permission assignments for a principal over a resource, those permissions having been defined by \( \mathcal{ARCA} \) over principal categories, \( \mathcal{PCA} \), and resource categories identified in \( \mathcal{RCA} \).

The algorithm is therefore subdivided into 2 sections. The input is a list of access control transactions as demonstrated in Table 1.

1. Generation of category definitions:
   The initial category definition is executed via the Doc2vec-based TypeDefinition and mergeCommon functions. TypeDefinition is aimed at identifying candidates with high similarity scores referring to the similar function, those
initial candidates are then clustered under an initial assumed common category based on the similarity score. mergeCommon then subsequently merges category duplicates based on commonly found neighbors.

The next step, the createTree function, constitutes the formation of principal and resource attribute frequency trees in accordance with FP-growth requirements, respectively.

Based on the frequency trees, the algorithm confirms candidate definitions of categories for principals and resources, as well as the assignment of permissions to categories (the latter is the essence of the ARCA relation) by ensuring that all candidates assumed to belong to one category are associated with the same permissions and opposite category candidates. According to the C-ABAC model [16], the category definitions are used to define the PCA and RCA relations.

2. Authorisations:

The appropriate, dynamic PAR relation can subsequently be derived from the ARCA relation following a final quality audit to ensure completeness and equivalence of derived authorisations, whether originating from the principal or the resource side, as well as their compliance with organisational requirements at the given time.

Section 4 provides a detailed explanation of the individual steps involved in each of the two stages.

4 Category mining proof of concept - Use case

To demonstrate the functionality of the proposed C-ABAC mining method we describe a small example followed by an application of the algorithm described in section 3 using the NLP method and FP-growth procedure.

We are assuming that the present case involves the initial mining of a CBAC policy from logs where no such policy previously exists. Those initial mining initiatives are referred to as uncovered categories in our algorithm as per [39] and [38]. The use of the algorithm to improve existing policies will be considered in future work.

4.1 Category definition

Doc2vec initially groups all given log transactions into principal/resource groups according to a similarity score assessment based on the attributes associated with principals and resources, respectively, via the typeDefinition and mergeCommon functions. These groups represent an initial assumption regarding future category definitions. The next step - FP-growth - needs to confirm that this assumption is correct and where that is not the case this next step needs to provide remediative action. FP-growth therefore acts as an auditor of the initial group assumption. Each group resulting from the Doc2vec-based similarity assessment is isolated from the complete logs and subdivided into a sublog. One
FP-tree is created for each member of each sublog group using as prefix, or lowest leaf, the prefix of those group members only. An FP-tree is therefore built by running each similarity-based sublog group member against the entire logs to identify frequent patterns for each sublog group member. This is to ensure that all sublog group members have the same authorisations on the same resource groups, splitting members from each other into separate groups where this is not the case, and also to identify members from separate groups which nonetheless have equivalent authorisations and should therefore be merged into one group. The main FP-growth functions are as follows: Pursuant to the creation of the FP-tree per group member createTree subsequently stores the conditional FP-bases in a header table which is then passed as an argument with the newly created FP-tree to createCategories. The function createCategories then calls the frequent patterns recursion of FP-growth to return the frequent patterns which are the confirmed newly generated category specifications. Those confirmed categories are the output which represents the logs enhanced with categories associated with each principal and resource as defined by the previous category specifications. Category permissions (i.e., ARCA tuples) are then derived and audited via QualityCheck to ensure log compliance and equivalence of category authorisations between the issued authorisations and the previously obtained category specifications. All correctly returned authorisations represent the final relation ARCA from which the dynamic, specific authorisation PAR can be computed.

Example 1 (Simple University System - Data set). As a simple example to represent the concept of category mining we consider a scenario as shown in Table 1. It displays hypothetical transaction logs for a hypothetical university access control system which has been designed to grant read or write access to certain resources. The resources (denoted by abbreviations for the sake of brevity) have two attributes, level and type, with values in the range 5-7 and Teaching/Research/Admin, respectively. The resources can be accessed by principals who themselves have two attributes, program and contract, which have the values BSc/MSc/PhD and TA/Staff, respectively.

The following is a summary of attribute function metadata for principals and resources as well as the actions in this hypothetical scenario:

**Principals Attributes**: ID, Name, Program, Contract **Attribute values**: string(k, e.g., kxxx), string (25, e.g., Alice), BSc/MSc/PhD, TA/Staff.

**Resources Attributes**: ID, Name, Level, Type **Attribute values**: url, string(8, e.g., PLD), 4-7, Learning/Teaching/Research/Admin.

**Actions** Read/Write

The principals in this example have been denoted as A-J for simplicity along with their individual identifiers. The principal attributes values of this system represent the attributes Program where BSc/MSc/PhD is the value of the attribute named Program and where TA/Staff is the value of the attribute named
Table 1. Example logs (Access transactions)

<table>
<thead>
<tr>
<th>Principal ID</th>
<th>Action</th>
<th>Resource ID</th>
<th>Attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url1</td>
<td>A, BSc, Read, PLD, 5, Teaching</td>
</tr>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url1.1</td>
<td>A, BSc, Read, OSC, 5, Teaching</td>
</tr>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url2</td>
<td>B, BSc, Read, PLD, 5, Teaching</td>
</tr>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url3</td>
<td>J, Staff, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kxxx</td>
<td>Write</td>
<td>url1.1</td>
<td>J, Staff, Write, Handbook, Admin</td>
</tr>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url1.1</td>
<td>J, Staff, Read, OSC, 5, Teaching</td>
</tr>
<tr>
<td>kxxx</td>
<td>Read</td>
<td>url1.1</td>
<td>J, Staff, Write, OSC, 5, Teaching</td>
</tr>
<tr>
<td>kxxxy</td>
<td>Read</td>
<td>url2</td>
<td>B, BSc, Read, COM, 6, Teaching</td>
</tr>
<tr>
<td>kxxxy</td>
<td>Read</td>
<td>url1</td>
<td>B, BSc, Read, PLD, 5, Teaching</td>
</tr>
<tr>
<td>kzzz</td>
<td>Read</td>
<td>url3</td>
<td>J, Staff, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kzzz</td>
<td>Write</td>
<td>url3</td>
<td>J, Staff, Write, Handbook, Admin</td>
</tr>
<tr>
<td>kzzz</td>
<td>Read</td>
<td>url1.1</td>
<td>J, Staff, Read, OSC, 5, Teaching</td>
</tr>
<tr>
<td>kzzz</td>
<td>Write</td>
<td>url1.1</td>
<td>J, Staff, Write, OSC, 5, Teaching</td>
</tr>
<tr>
<td>kyxx</td>
<td>Read</td>
<td>url1</td>
<td>I, Staff, Read, PLD, 5, Teaching</td>
</tr>
<tr>
<td>kyxx</td>
<td>Write</td>
<td>url1</td>
<td>I, Staff, Write, PLD, 5, Teaching</td>
</tr>
<tr>
<td>kxxz</td>
<td>Read</td>
<td>url2</td>
<td>C, MSc, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kxxz</td>
<td>Read</td>
<td>url3</td>
<td>J, Staff, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kxxz</td>
<td>Read</td>
<td>url4</td>
<td>C, MSc, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kxxw</td>
<td>Read</td>
<td>url3</td>
<td>D, MSc, Read, Handbook, Admin</td>
</tr>
<tr>
<td>kxxz</td>
<td>Read</td>
<td>url5</td>
<td>G, PhD, TA, Read, PGR, Research</td>
</tr>
<tr>
<td>kxxz</td>
<td>Write</td>
<td>url5</td>
<td>G, PhD, TA, Write, PGR, Research</td>
</tr>
<tr>
<td>kyyz</td>
<td>Read</td>
<td>url5</td>
<td>F, PhD, Read, PGR, Research</td>
</tr>
<tr>
<td>kyyx</td>
<td>Read</td>
<td>url5</td>
<td>H, PhD, TA, Read, PGR, Research</td>
</tr>
<tr>
<td>kyyx</td>
<td>Write</td>
<td>url5</td>
<td>H, PhD, Write, PGR, Research</td>
</tr>
</tbody>
</table>

Contract. The same approach is applied for resources where the values 5-7 represent the attribute function defined as Level while the values Teaching/Admin/Research, are representative of the attribute function defined as type.

Table 1 is therefore a hypothetical example of 20 access request transactions as recorded for this fictitious university system. The Transactions column lists, for each transaction, the ID of the principal with the requested permission (i.e., action to access the specific resource ID). The Attribute values column lists the names of principals and resources as well as their attribute values at the time the transaction happened (as stated in the $\text{PAT}_A$ and $\text{RAT}_A$ relations).

Example 2 (Simple University System - Process and Results). The following is a summary of our process displayed with our example, from category definition to the issuance and audit of $\text{ARCA}$:

As the first step, principal and resource groups must be defined according to the similarity clustering of Doc2vec. This is followed by the FP-growth-based confirmation of the correctness of similarity. An example of a Doc2vec-generated group is (A(BSc), C(MSc)) for principals and (PLD(5, Teaching), OSC(5, Teaching)) for resources. Once FP-growth has confirmed all valid categories they are associated with the label of its members, or multiple labels where the results so require, to obtain category definitions (the $\text{CAT}_A$ relation described in Section 2.1). The labels correspond to the main chosen attribute associated with the principal/resource member and therefore acts as the identifier of the now-confirmed category (previously NLP-derived group). In this case labels have been chosen as Program for principals and Type for resources, respectively.

Example frequent patterns generated following Doc2vec with minimum support $= 2$ The following are example frequent patterns as generated by FP-growth for the sublog principal/resource group members noted above (principal group members A, C and resource group members PLD, OSC) by using userID/resourceID as the FP prefix for principals and resources, respectively and by running each group member against the complete logs: A(Read, url1, url1.1), C(Read, url3, url2, url4), PLD(kxxx, kxyx, Read), OSC(kxxx, kzzz, Read)
We can see in this example frequent pattern display that principal A has Read permissions on resources of the Type Teaching and Level 5 whereas C has Read permissions on resource groups with associated Type Teaching and Level 5 as well as Type Admin and Level Handbook, two separate resource groups (defined as separate by Doc2vec). We therefore need to split this group into two categories, one for principal A and one for principal C. Subsequent members will be allocated to each category A or C if their frequent patterns match those associated with category A or C. We repeat this process for resources, here, we determine that both resources PLD and OSC have Read permissions associated with two groups of principals: 1.Program BSc, no Contract, and 2.Program PhD, Contract Staff. We therefore keep PLD and OSC as one category, subsequent members will be added to this category provided their frequent patterns match those of the members already inside the category.

The function further seeks to merge separately returned categories which should, however, be classified under the same category definition because they are associated with the same opposite entity and the same action. Running Doc2vec on the example above returned as separate the initial groups for principals F (PhD) and H (PhD, TA). However, running FP-growth confirmed that although there are the two separate groups (PhD, TA) and (PhD) they both are linked to the same action (Read) for the same resource group (PGR, Research). Therefore, by logic, both (PhD, TA) and (PhD) should be classified under the more general (PhD), consisting of members with attributes (PhD) and (PhD, TA) which will be the final category definition returned for this scenario. Note that G will not be included in this category since G has a Write permission associated with resource group (PGR, Research). G’s category will therefore be a separate one. Applying the same logic to resources the final category definitions for the example above therefore will be as detailed below. The arrow indicates the reason they were placed into this category where appropriate, i.e., the ARCA relation. To be precise, the brackets following the arrow represent the action and opposite entity of the ARCA tuple of the labelled entity:

**Final CAT A** Principals assuming exact frequent pattern ARCA (BSc) → (Read, 5, Teaching), (BSc) → (Read, 5/6, Teaching), (Staff) → (Read/Write, 5, Teaching/Handbook Admin), (Staff) → (Read/Write, 5, Teaching/PGR/Research), Read, Handbook Admin, (MSc) → (Read, 6/7, Teaching/Handbook Admin), (MSc) → (Read, Handbook Admin), (PhD/TA) → (Read/Write, PGR/Research), (PhD) → (Read, PGR/Research)

**Final CAT A** Resources assuming exact frequent pattern ARCA (Teaching, 5) → (BSc/Staff), (Teaching, 5) → (BSc, Read/Staff, Write), (Teaching, 6, Read) → (BSc/MSc), (Teaching, 7) → (MSc, Read), (Admin/Handbook) → (Staff/MSc, Read), (Admin/Handbook) → (Staff, Write), (Research/PGR) → ((PhD/TA), Read), (Research) → ((PhD/TA), Write), (Research/PGR) → (PhD, Read), (Research/PGR) → (Staff, Write)

Following our category definition grouping, the category assignment function establishes which categories are assigned to principal and resource in each trans-
action (PCA and RCA relations in CBAC). For example, in the first transaction in Table 1 principal A will be assigned to BSc and PLD to Teaching.

ARCA then connects entities from both category definitions with their corresponding actions as per logs via our definition of FP-growth. So, for example, category BSc will have Read access to category Teaching. Once we have all ARCA tuples, we go back to the logs with QualityCheck to ensure that the authorisations we produced match up with the records we used for generation.

As noted above, CBAC does not require those records to be limited to attributed-based access logs, but, in the ABAC instance of CBAC, we can also use a record of existing policies for our mining approach. As long as there are no hidden attributes and that the records represent a complete account of the specific access policy we can show that our CBAC miner produces the same results as the ABAC miner but requiring far less manual intervention and resulting in a much more succinct, clearer structure of policy assignment. Because the output from QualityCheck returns data logs enriched with the corresponding principal/resource category assignments we can derive PAR from ARCA immediately, in our example this would represent each of the twenty transactions connecting principals with the appropriate resources via their connected actions.

Rather than specifying individual authorisations to each principal for each resource, we aim to specify categories such that the authorisations exhibited in the logs can be derived according to CBAC axioms. In example 2 one category for principals could be, e.g., MSc or TA, while resources could potentially be categorised as Teaching or Level 5. The subsequent ARCA in this example then includes a tuple (Read, Teaching, BSc) indicating that the action Read on teaching resources is authorised for principals in the category BSc, and a tuple such as e.g., (Write, Teaching, Staff) indicating that the action Write on Teaching resources is authorised for principals in the Staff category, etc. PAR is then the dynamic authorisation relation which is derived on the fly from our policy specifications, i.e., derived from the category definitions and ARCA. So, here, PAR would e.g., include a tuple (A, Read, PLD) which indicates that BSc student A has read access to the teaching resource named PLD.

Following the two stages of our algorithm we have observed a reduction from originally 20 transactions with 10 principals and 6 resources into 14 ARCA relations with 8 principal and 10 resource categories. The relatively high number of categories is currently due to FP-growth looking for an exact match between frequent patterns of each NLP-produced group member which is the most securely accurate but also the most restrictive method. The algorithm permits adding a weight to FP-growth to allow for a match above a certain threshold, e.g., 0.87 which would make further reductions possible.

The key note here is the distinction between categories, which serve as an easily identifiable, dynamic label, and ARCA, the justification behind those category groupings as defined by the FP-growth check. Because FP-growth uses either the resource or principal as prefix leaf for the determination of its frequent patterns those patterns are split into either resource or principal sequences which then are assigned a category label. ARCA is the underlying FP-growth pattern
associated with the corresponding category and can be seen as the equivalent of an ABAC rule. This comparison will be examined in detail in section 5.

5 Validation

Following the description of our process for category definition we are now turning our focus on validating our practical implementation results. To do so a two-step process is required. In the first step the objective is to define the correctness of our method. Here we seek to establish that both the NLP as well as the FP-growth algorithms are suitably correct methods for the definition of categories, i.e., that termination and a resulting definite output can be confirmed. The second step seeks to further achieve a confirmation of a satisfactory standard of quality of policies obtained, which in the case of CBAC means a minimal number of correctly defined categories. We propose a number of extended additional validation checks and apply a direct comparison with the ABAC miner from [39] and [38] to provide a clear documentation of how our method compares to and differs from the aforementioned ABAC miners.

Proposition 1 (Correctness). Given a finite list of transactions, each consisting of principal, action, resource, attributes (name and value) for principal, resource and environment, and answer (grant, deny), the CBAC miner terminates and outputs:

- a list of attribute-based category definitions and
- either an ARCA relation such that the PAR authorisation relation (derived from ARCA and the category definitions according to axiom (a1)) includes the granted transactions in the input, or a list of input transactions where the answers exhibit inconsistencies with respect to the known attributes of principals, resources and environment.

Proof (Proof Sketch). Termination follows from the fact that the list of input transactions is finite, both the NLP and the FP-tree growth algorithms terminate, and all the functions used in the miner are bounded iterations (for-loops over finite lists). The correctness of FP-growth ensures that the function createCategories produces a list of attribute-based category definitions. The last part of the proposition follows from the correctness of the QualityCheck function, which aims to confirm log compliance as well as to identify any transactions where the principal and resource categories and the action coincide, but the answers are different (grant/deny), which may suggest a policy error requiring an alert trigger.

To confirm the correctness of any hypothetical category definitions as in line with successful log requests we must execute 2 steps: 1. we need to ensure that our miner produces category definitions as appropriate for the given successful requests, i.e., that the miner returns a policy that generates the same authorisations as in the input logs. This is confirmed with the QualityCheck function based on FP-trees, therefore the miner is correct. 2. We need to confirm the
quality of policies obtained. In the case of CBAC policies, good quality means a minimal number of categories. Our example - which has been deliberately chosen to include many different authorisations for the same labels of principals - shows that we can reduce from 20 transactions with 10 principals and 6 resources to 14 ARCA relations with 8 principal categories and 10 resource categories - this is with an exact frequent pattern match which can be relaxed - which indicates that policies obtained by the proposed method are indeed in line with CBAC axioms. A number of options are available for exploration to confirm the validity of our proposed method. Traditional policy property checks as defined in e.g., [20, 24, 42] seek to determine whether the policy satisfies the given property. In our case we are interested in a reduction of categories. Following the property verification approach would therefore in our case entail the manual creation of policies, a generation of synthetic logs and then running the miner to assess whether there is a difference in size between the manually created and the mined policy. Suggestions for more elaborate verification methods can be found in e.g., [30] where the authors propose a mutation verification introducing faults, or mutations, into the policy and checking whether the miner finds and rejects those faults - which would indicate a good quality of policies - or whether the miner simply uses those faults and returns authorisations notwithstanding which would indicate a low quality. In future work we seek to analyse and apply the most appropriate of these and similar verification approaches to confirm that our initial hypothesis regarding the good quality of our mined policies can is indeed correct.

Example 3: Validation University System - Data Set, Process and Results Since we are mining policies based on attributes, a direct comparison between the outputs of our proposed and the ABAC miner such as the found in [39] is a natural initial step for our validation work. To achieve a direct comparison we are using the university data set described as part of the implementation of policy mining algorithms in [39] and in the extended version thereof [40] and compare the published results as obtained from the ABAC miner having processed that data set with the corresponding results having applied our proposed CBAC miner to the same data. As described in section 4.1 the equivalent of what is generally defined as an ABAC rule is our definition of ARCA which is represented by the frequent patterns produced following the NLP pre-filter. The key distinguisher with the ABAC rule is that in our system these patterns provide the foundation from which a final category definition can be derived. Therefore, rather than producing a set of unclassified rules, we use our frequent patterns, i.e., what we define as ARCA to justify dynamic principal/resource classification. Currently our FP-growth method is using an exact match between frequent patterns of group members. This can be relaxed by adding a weight to FP-growth to allow grouping above a specified threshold, e.g., 0.87, which would incorporate a relaxation of accuracy similar to the Jaccard similarity results used by [39]. Whether such relaxation is appropriate depends on the organisation, we chose to describe the most restrictive yet most accurate approach.

Our subsequent application of the proposed CBAC miner to the same data set begins with the NLP similarity-based grouping, where resources and principals
are grouped into similar clusters according to their respective attributes. The following is an example of a such principal category assumption based on the initial NLP pre-filter:

**NLP-based category pre-definition:** Note that only the logIDs would be included as members identifiers, however the full principal/attribute association is displayed for clarity.

1. Name(csStu5), Position(student), Department(cs), courseTaken(cs601, cs602), courseTaught(no course taught), isChair(false)
2. Name(csStu1), Position(student), Department(cs), courseTaken(cs101), courseTaught(cs101, cs602), isChair(false).

We subsequently move to the second FP-growth based check of the NLP pre-defined category definitions against the complete logs as found in the log section of the data set. An example log consists of a log/user/resource ID as well as the associated action, so e.g., (3, csChair, csStu1trans, read). We choose a sublog which corresponds to the pre-defined category definitions and run FP-growth with prefixes from those definitions only against the entire logs, so in the example above we would run the position prefix of csStu5 and csStu1 (userID for both, i.e., csStu1 and csStu5) against the logs to check if the returned frequent patterns are equivalent.

An example of such a frequent pattern in this case is (cs101gradebook, readMyScores, read). Where it is found that frequent patterns of two assumed members of one category are not the same those members are split into two categories. On the other hand, where it is found that members from other categories share the same frequent patterns as members from categories initially defined as separate all members producing equivalent frequent patterns are merged into one category. Utilising position (e.g., student, staff, student/faculty) as the final label for principal and type (e.g., gradebook, application, transcript/roster) as the final label for resource categories, respectively, our algorithm ends with 15 ARCA tuples, 10 principal and 11 resource categories as extracted from 100 transactions, 21 principals and 33 resources which confirms the category reduction hypothesis as obtained with the CBAC miner. It should be noted that, in addition to the category reduction, our algorithm does not stop at the rule production stage as the ABAC miner does. Rather, we seek to interpret those rules as obtained by NLP/FP-growth to end up with a label which is dynamic in nature - as described by the CBAC axioms from section 2.1 - and can therefore support dynamically changing input while at the same time reducing the complexity displayed by the ABAC mining approach.

## 6 Related Work

Access control models have been extensively studied in the past with the two most prominent models being RBAC ([34], [35]) and ABAC ([38],[39], [44], [15], [37]) as well as combinations and extensions thereof ([43], [11], [44]). Whilst all of these models have greatly expanded the understanding of access control features and how to manipulate them to achieve greater security as well as flexibility the
key characteristic of all of these models remains the reliance on a pre-defined label assignment as the foundation from which authorisations are derived. ABAC takes a step away from that due to its more dynamic nature, however, it can produce an excessively large pool of rules. The CBAC model aims to provide a greater scope for the application of a label-free and easy to maintain approach to enable a future system which is dynamically modifiable and sensitive to the evolution of entity features as well as the relationships between those entities and which does not require any prior knowledge about what characteristics elements must possess to be considered relevant for the definition of rules. The CBAC model can lay the foundation for such explorations since it has been shown to subsume previously existing models given its non-exclusive nature with regards to what constitutes relevance for a feature to be considered rule-defining. [17] and [16] show one example of such subsumption in relation to the ABAC model, this work provides the framework for the exploration of attribute-based category mining explored here. Multiple studies have also been proposed for mining as the practical implementation of RBAC policy generation, with [36] proposing a hierarchical clustering of permissions and [32] exploring the benefits of weighted structural complexity for situations where RBAC policies may envisage multiple objectives. ABAC mining has also made some significant progress with [39] focusing on an extended conciseness in the process of policy mining aiming to result in greater ease of use and [27] further exploring the use case for weighted structural complexity in an unsupervised ABAC mining approach via the formation of similarity clusters. None of the studies, however, allow for the allocation of multi-valued, dynamic attributes whilst at the same time promoting explainability and ease of maintenance. This section seeks to explore in detail how our CBAC mining approach differs from both ABAC and RBAC mining due to its inherently dynamic capabilities paired with an easy to use and maintain architecture.

Similar to [39] the methodology used for policy mining in this work is derived from attribute-based policies extracted from access logs. We have decided to use attributes as the foundation for the construction of categories via FP-growth due to its very wide interpretation of attributes which is closest to that of an all-encompassing feature definition. Despite the initial similar mining approach there are however some key distinguishers between ABAC and CBAC mining as we propose it here. A key distinguisher of CBAC from ABAC is the fact that ABAC produces an aggregation of attributes which are then translated to rules from which policies are derived. There are two problems with this: One, despite the initial dynamic attribute collection the resulting policies are inextricably linked to those attributes and therefore require predefinition of features known in advance. Two, since ABAC does not provide any mechanism to connect the semantic meaning of attributes this results in an overabundance of rules which are difficult to manage and whose interconnections cannot be well understood. This lack of meaningful relations poses problems related to redundancy, relevance as well as governance compliance and thereby compromises the initial flexibility which ABAC attempted to offer. We tackle both of these concerns with the
initial NLP-based stage of our algorithm which serves as a pre-filter to provide a set of initial category definitions based on semantic relationships. This pre-filter makes no preconceptions with regards to what features should be relevant. The following validation step via FP-growth aims to ensure all category members deemed similar actually carry the same level of authorisations with regards to the same resources. Further, CBAC, by its very nature, is not an aggregator of features. Rather, its goal is to find meaningful paths between features as well as relationships between those paths and subsequently group features according to the appropriate paths and relationships found. Since these paths may change at any point in time and environment CBAC has at its core the objective to track any such change and respond in real-time via a reevaluation of feature paths and relationships. [1] provides a heuristic search-based mining, by generating n rules with a random combination of attribute values. If a rule is not correct it is swapped by another one which differs just in one attribute value. This is a computationally expensive method and does not help interpret the rules which is what we are aiming to do. A work bearing similarity to ours is [26] where the authors propose an automated log pattern finding approach as well as a policy refinement and pruning algorithm to achieve higher quality of policies. The key differentiators with our work are 1. that input for our method is not limited to attributes as explained in section 2.1 and 2. that the authors, while proposing a policy quality metric, utilise a manually-driven quality check method as opposed to our automated FP-growth check. In a similar fashion to the authors in future work we would like to explore the application of our methods to negative authorisations, a wider scope of non-verbal data and conflict resolution.

7 Conclusions and Future Work

We have presented an initial foundation for a category-based access control miner based on access transaction logs augmented with attribute data, the first such study to the best of our knowledge. The next stage of our work shall include both further evaluation of the quality or our proposed mining approach - which the initial hypothesis set out in section 5 assumes to be satisfactory - as well as measures for the assessment of an incremental adjustment of new incoming policies. We aim to explore further mining as well as deep learning techniques such as the ones proposed in [33] and evaluate results from such methods to find the most suitable ones in our continued study of what constitutes continued relevance, without falling back on static definitions. Seeking to alleviate the explainability concern we aim to further explore rule-based techniques as a potential alternative to the current NLP pre-filter. Optimisation of our methodologies also remains a key aspect for future work. A suggestion to facilitate such optimisation could be the study of negative authorisations, or prohibitions, as proposed in [9], [6] and [2].
References