Towards Automatic Attribute Based Access Control Policy Design and Management for Highly Dynamic Environments

by

Leila Karimi

M.S. in Information Technology Engineering, Sharif University of Technology, 2013

B.S. in Information Technology Engineering, Sharif University of Technology, 2011

Submitted to the Graduate Faculty of

the School of Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2022

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Leila Karimi

It was defended on

May 27th, 2022

and approved by

Dr. James Joshi, School of Computing and Information, University of Pittsburgh

Dr. Hassan Karimi, School of Computing and Information, University of Pittsburgh

Dr. Balaji Palanisamy, School of Computing and Information, University of Pittsburgh

Dr. Mai Abdelhakim, Department of Electrical and Computer Engineering, Swanson School of Engineering, University of Pittsburgh

Advisor: Dr. James Joshi, School of Computing and Information, University of Pittsburgh

Co-Advisor: Dr. Mai Abdelhakim, Department of Electrical and Computer Engineering, Swanson School of Engineering, University of Pittsburgh

Towards Automatic Attribute Based Access Control Policy Design and Management for Highly Dynamic Environments

Leila Karimi, PhD

University of Pittsburgh, 2022

With the rapid advances in computing and information technologies, traditional access control models have become inadequate in terms of capturing fine-grained, and expressive security requirements of newly emerging applications. An attribute-based access control (ABAC) model provides a more flexible approach to addressing the authorization needs of complex and dynamic systems. An ABAC model grants access to a requester based on attributes of entities in a system and an authorization policy; however, its generality and flexibility come with higher costs: the costs of policy development, enforcement, and maintenance. Hence, while organizations are interested in employing newer authorization models, migrating to such models poses a significant challenge. Many large-scale businesses need to grant authorizations to their user populations that are potentially distributed across disparate and heterogeneous computing environments. Each of these computing environments may have its own access control (AC) model. The manual development of a single policy framework for an entire organization is tedious, costly, and error-prone. Further, the increasing complexities of organizational systems and the need for federated access to their resources make the task of AC enforcement and management much more challenging. In addition, policy misconfigurations that hinder the effectiveness of AC systems expose an organization to various security threats.

In this dissertation, we propose approaches and methods that facilitate ABAC policy Design and management. In particular, (i) we propose a methodology for automatically learning ABAC policy rules from access logs of a system to simplify the policy development process. The proposed approach employs a clustering-based algorithm for detecting patterns in access logs and extracting ABAC authorization rules from these patterns. In addition, we propose two policy improvement algorithms, including rule pruning and policy refinement algorithms to generate a higher quality mined policy. Further, (ii) we propose an adaptive

ABAC policy learning approach to automate the authorization management task. We model ABAC policy learning as a reinforcement learning problem. In particular, we propose a contextual bandit system, in which an authorization engine adapts an ABAC model through a feedback control loop; it relies on interaction with users/administrators of the system to receive their feedback that assists the model in making authorization decisions. We propose four methods for initializing the learning model and a planning approach based on attribute value hierarchy to accelerate the learning process. In addition, (iii) we propose a machine learning based approach for detecting ABAC policy misconfiguration and refining ABAC policy rules in order to enhance the quality of policy and prevent system exploitation. We then evaluate our proposed methods and approaches by implementing a prototype of the ABAC policy extraction method, the adaptive ABAC policy learning framework, and the ABAC policy misconfiguration detection and tuning approach.

Table of Contents

List of Tables

List of Figures

Preface

My Ph.D. journey has been the most challenging and enriching period of my life. First, I would like to express my sincere gratitude to my advisor Prof. James Joshi for his support and supervision during the journey. I also would like to thank my committee members, Dr. Hassan Karimi, Dr. Balaji Palanisamy, and Dr. Mai Abdelhakim for their valuable feedback and insightful suggestions on my dissertation work.

I am thankful to my colleagues at the LERSAIS lab and my school friends who made my Ph.D. experience more enjoyable. I would like to thank Nathalie Baracaldo, Chao Li, Nuray Baltaci Akhuseyinoglu, Runhua Xu, Jinlai Xu, Kuheli Sai, Maryam Aldairi, and Maryam Karimi. I appreciate the support of the great staff at school, especially, Patricia Markham and Kelly Shaffer whose support was very helpful to me on numerous occasions. I also acknowledge that part of the research presented in this dissertation has been supported by the US National Science Foundation (NSF) under the grant DGE-1438809.

I am especially so grateful to my husband, Mohsen Kamali. Thanks for encouraging and supporting me during the toughest days of my Ph.D. Without you, this wouldn't be possible. Last but not least, I would like to thank my parents, my sister, and my brother. Thank you for giving me the love and support that I needed to follow my dreams.

1.0 Introduction

Access control systems are critical components of information systems that help protect information resources from unauthorized access. Various access control models and approaches have been proposed in the literature including Discretionary Access Control (DAC) [53] [25], Mandatory Access Control (MAC) [9] [51], and Role-Based Access Control (RBAC) [52]. However, with the rapid advances in newer computing and information technologies (e.g., social networks, Internet of Things (IoT), cloud/edge computing, etc.), existing access control (AC) approaches have become inadequate in providing flexible and expressive authorization services [20]. For example, a healthcare environment requires a more expressive AC model that meets the needs of patients, healthcare providers as well as other stakeholders in the healthcare ecosystem [32, 34]. *Attribute Based Access Control* (ABAC) models present a promising approach that addresses newer challenges in emerging applications [28].

An ABAC approach grants access rights to users based on attributes of entities in the system (i.e., user attributes, object attributes, and environmental conditions) and a set of authorization rules. An ABAC model provides an increased level of flexibility that promotes information system security [28]. It has been considered as a recommended AC model in the federal identity credential and access management roadmap and implementation guidance [15]. However, an ABAC approach comes with some costs: the costs of policy development, enforcement, and maintenance. In this dissertation, we aim to provide solutions to such complex issues.

In this dissertation, we propose an ABAC policy design and management framework shown in Figure 1 that can be employed by organizations by integrating it with their information systems. We assume that, when migrating to an ABAC model, there is an access log available from the former AC model of the system which will be used for extracting ABAC policy rules. In such access logs, each access tuple corresponds to an access request and the authorization decision of the system for this request based on the former policy. Using the available access log, we will extract the corresponding policy rules. The extracted policy rules will be pruned/refined in multiple rounds to reach the desired policy quality.

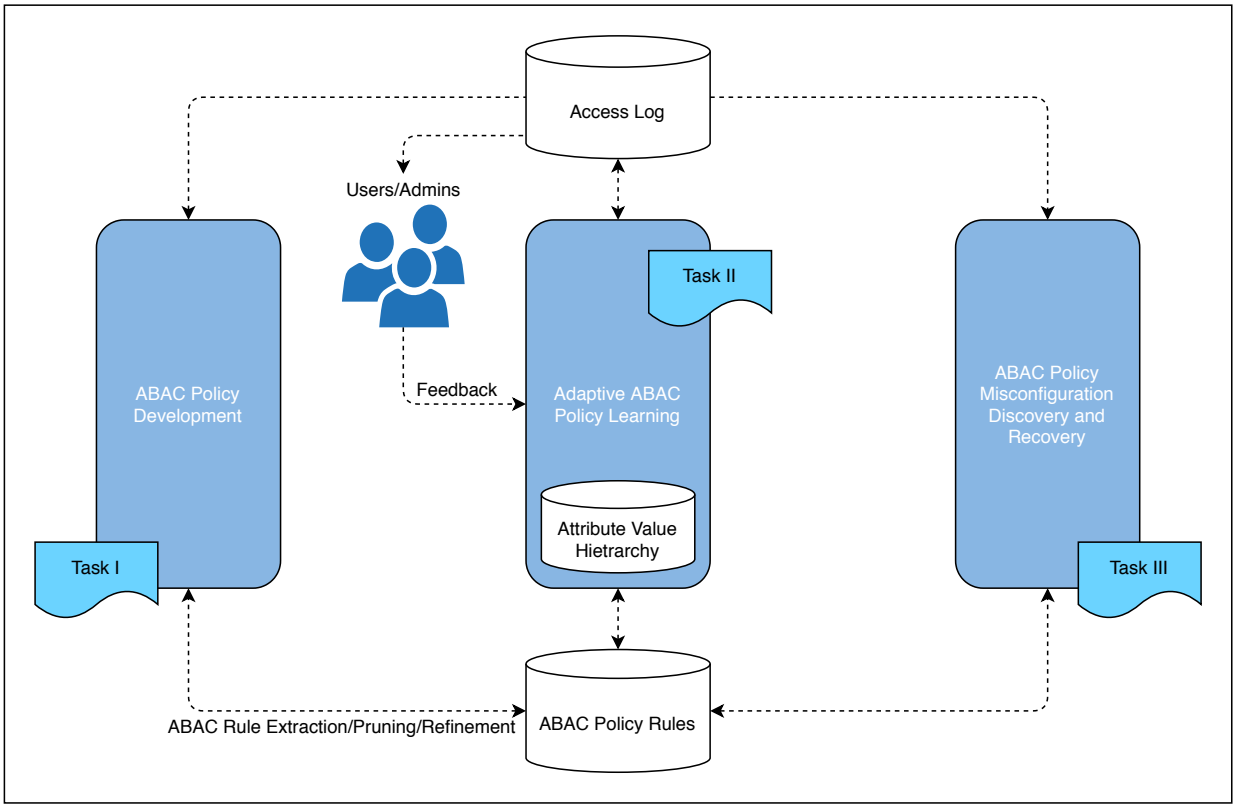


Figure 1: Overview of ABAC Policy Design and Management Framework

Emerging and modern computing and information systems are dynamic and highly interconnected. The authorization needs of the users and the attributes of the entities in the environment evolve rapidly. An established ABAC authorization system can become outdated quickly in such a dynamic environment requiring continuous maintenance. Manual revision and modification of policy rules is a cumbersome and error-prone task. To address such challenges, a system needs to employ an adaptive access control approach that learns to revise/modify authorization rules based on the feedback provided by the users. Such an authorization framework shows promise in providing a usable and effective access control solution for complex environments. To accelerate the learning process, the system may utilize a knowledge base including hierarchies over attribute values and apply a planning method exercising such hierarchies.

A dynamic and complex system can not completely rely on users' feedback for revising and modifying the outdated/defective ABAC policy authorization rules. It is also important to have a comprehensive mechanism for automatically detecting and correcting misconfigurations in an AC policy. Such a system can employ AI/ML based approaches to discover anomalies in an access log and amend ABAC policy rules accordingly.

1.1 Challenges and Motivations

In this section, we discuss the motivation for this dissertation and present the challenges that need to be addressed and the problem statement. Although organizations and developers are interested in employing the next-generation AC models, adopting such policy frameworks poses a significant challenge. Many large organizations need to grant authorizations to their huge user populations distributed across different policy domains, including legacy systems. Each of these policy domains may have its own AC model. The manual development of a single policy for the entire organization is a tedious and error-prone process. *Policy Mining* techniques have been proposed in the literature to address such challenges to help organizations cut the cost, time, and errors during policy development and management. Policy mining algorithms ease the migration to more recent/appropriate

authorization models by completely (or partially) automating the process of constructing AC policies. However, the existing research suffers from several limitations, as follows:

- First, the existing approaches do not support mining authorization rules with negative filters. An ABAC policy rule can be comprised of a set of positive and negative filters. Negative filters are useful in scenarios when an exception needs to be expressed. For example, a healthcare provider can express the following rule using a negative attribute filter: "*A nurse can read a patient's record except for payment purposes.*" Using negative filters in rule expressions helps create a more concise authorization policy.
- Secondly, some proposed approaches such as in [68, 29] are unable to mine a high-quality policy when the given access log is not complete in the sense that every possible combination of attribute values is not included in the access log.
- Third, the proposed approaches are unable to mine a policy from noisy access logs containing over-assignments and under-assignments [41, 14]. Having noisy access records is a common problem in evolving domains such as IoT or social networks [40]. It is essential that an ABAC policy miner should be capable of handling a reasonable amount of noise to be applicable to real-world applications.
- Last but not least, the existing approaches do not include techniques for improving the mined policy after the first round of policy extraction. In addition, in scenarios where the authorization policies may change over time (such as in social networks with the addition and removal of various features), these approaches do not provide any guidelines for adjusting the policy. This makes the practical deployment of these approaches very difficult.

Furthermore, none of the existing work addresses these issues in an integrated way. Hence, some key challenges are

challenge 1. *How can an ABAC policy learning approach extract ABAC policy rules that contain both positive and negative attribute filters and relation conditions/constraints?*

challenge 2. *How can an ABAC policy learning approach be effective even with an incomplete set of access logs and in presence of noise?*

challenge 3. *How to measure the quality of the extracted ABAC policy in terms of its correctness and conciseness?*

challenge 4. *How to refine the extracted ABAC policy based on the available false positive and false negative records to improve its quality?*

Although ABAC has been shown to be superior to other existing models in many respects, its generality and flexibility come at a higher cost. ABAC models can be much more complex than other AC models. Furthermore, the increasing complexities of organizational systems and the need for federated access to their resources make the adoption of ABAC and related AC management tasks much more challenging.

Recent research efforts have been focused on exploring Artificial Intelligence (AI) and Machine Learning (ML) based approaches for developing and managing ABAC authorization systems, ranging from mining ABAC policies from access logs [35, 29, 33] to extracting such policies using deep learning (DL) algorithms [43] and employing natural language processing (NLP) tools for automating policy development and enforcement [4]. While supervised learning algorithms seem to be an option for ABAC policy development and management, they suffer from several limitations. First of all, supervised learning algorithms, especially DL algorithms, require a huge amount of labeled data showing which access requests should be permitted and which ones should be denied. Acquiring such labeled data is time-consuming and expensive and in some situations may not even be possible. Second, emerging and future computing and information systems are dynamic and highly interconnected. The authorization needs of the users and the attributes of the entities in the environment evolve rapidly. A supervised learning approach is incapable of adjusting to such dynamic settings or it needs a new set of labeled data to restart the learning process. Last but not the least, in most situations, the available access logs are often sparse and/or contain only partial activity logs, and hence, lack information about all possible access requests and the authorization decisions of the system for them. As a result, an ABAC model learned by a supervised learning algorithm over such data performs poorly in a real, constantly evolving system when encountering a new type of access request. We believe that an adaptive access control approach that learns authorization rules from feedback provided by the users is a promising

solution. Reinforcement Learning (RL) provides a suitable infrastructure for such adaptive authorization. Here, the key challenges are as follows:

challenge 5. *How can an RL-Based ABAC policy development and management approach adapt itself in a very dynamic and complex environment where the authorization needs of users and other system characteristics evolve rapidly?*

challenge 6. *How can an ABAC policy learning approach effectively learn policy rules given appropriate feedback from the users of the system?*

challenge 7. *How can an RL-Based ABAC policy development approach accelerate the learning process?*

In addition, the complexity of the ABAC model makes the manual detection and resolution of policy misconfiguration a very challenging task. ABAC policy misconfigurations have been a focus of several recently proposed approaches [27, 18, 19]. Such approaches aim to detect different types of anomalies in ABAC policy configuration by analyzing either the ABAC policy rules or the corresponding access logs. However, these approaches have several key limitations. The approach proposed in [27] only focuses on ABAC policy rules for detecting anomalies and they do not consider access logs in their analysis. Analyzing ABAC policy rules on their own and without considering the generated access logs may not reveal all possible misconfigurations. On the other hand, methods proposed in [18, 19] only focus on access logs for detecting policy anomalies. Access logs are often huge and analyzing them without considering the corresponding authorization rules is computationally expensive. In addition, all the existing solutions are designed for one-time anomaly detection of authorization policies. They do not account for updates in policy rules or the attributes of entities in the system. However, modern information systems evolve rapidly, from the addition of new attributes and attribute values to the updates in authorization rules. Hence, the anomaly detection process needs to be an ongoing procedure in current systems. Here, the key challenges are as follows:

challenge 8. *What are run-time policy misconfigurations and how are they different from design-time policy misconfigurations?*

challenge 9. *How can we efficiently detect run-time policy misconfigurations?*

challenge 10. *How can we refine the ABAC policy rules to prevent run-time policy misconfigurations?*

1.2 Organization

The rest of the dissertation proposal is organized as follows. In Chapter 2, we review the closely work related to the dissertation. In Chapter 3, we present the proposed automatic ABAC policy extraction method from access logs and compare it against the proposed approaches in the literature. In Chapter 4, we propose an adaptive ABAC policy learning approach using a Reinforcement Learning framework. In Chapter 5, we propose the ABAC policy misconfiguration discovery and resolution framework. Finally, in Chapter 6, we present the conclusions and future work.

2.0 Related Work

In this chapter, we present the related work relevant to this dissertation. We begin by presenting background on *policy learning approaches* (section 2.1). Then, we present *adaptive authorization models* (section 2.2). Finally, we introduce some background information on *ABAC policy misconfiguration detection and resolution* (section 2.3).

2.1 Policy Learning Approaches

As RBAC became popular, many organizations decided to equip their information systems with a 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 have addressed such a challenge by introducing automated role extraction algorithms [46, 65, 36, 54, 61, 62, 69, 24, 45, 59, 48]. 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.

Xu and Stoller are the first to propose various algorithms for mining ABAC policies from different given input data including RBAC policy [66], access logs [67], and access control list [68] plus attribute information. Their policy mining algorithms iterate over access control tuples (generated from available information, e.g., user permission relations and attributes) and construct candidate 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 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 (The analysis of their proposed algorithm is discussed in Section 3.4 in more details).

Following Xu and Stoller's proposed method, Medvet *et al.* in [41] propose a multi-objective evolutionary algorithm for extracting ABAC policies. The proposed approach is

a separate and conquer algorithm, in each iteration of which, a new rule is learned and the set of access log tuples becomes smaller. Their algorithm employs several search-optimizing features to improve the quality of the mined rules. Although their approach is a multi-objective optimization framework which incorporates requirements on both correctness and expressiveness, it suffers from the same issue as the approach proposed in [68].

Iyer and Masoumzadeh [29] propose a more systematic, yet heuristic ABAC policy mining approach which is based on the rule mining algorithm called PRISM. Although their proposed approach is the first to extract both positive and negative authorization rules, 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. Further, their proposed algorithm needs a complete dataset as the given input, meaning that every possible combination of attribute values has to be included in the access log which is not feasible in real-world scenarios.

Cotrini *et al.* propose an algorithm called Rhapsody for mining ABAC rules from sparse logs [14]. Their proposed approach is built upon subgroup discovery algorithms. They define a novel metric, *reliability* which measures how overly permissive an extracted rule is. In addition, they propose a universal cross-validation metric for evaluating the mined policy when the input log is sparse. However, their algorithm is not capable of mining policies from logs with a large number of attributes as the number of extracted rules grow exponentially with respect to the number of available attributes.

Jabal *et al.* in [30] first employ an association rule mining (ARM) algorithm called Apriori [2] to extract the associations between users and resources in the access log and extract the *ground rules* based on these associations. In the next step, they propose a statistical-based approach for generalizing such policy rules. They also employ an ML classifier to classify the records that were not covered by the extracted rules. Their proposed approach tries to generate ABAC policies that are *complete* and *correct*, however, they do not bring the size of such extracted policies into consideration. Hence, their method generates complex policies for real-world scenarios where a typical system contains a large number of users, resources, and corresponding attributes.

2.2 Adaptive Authorization Models

With rapid advances in computing systems, there is an increasing demand for more effective and efficient access control (AC) approaches. Further, the increasing complexities of organizational systems and the need for federated access to their resources make the task of AC management much more challenging. The manual management and maintenance of authorization systems are time-consuming and error-prone. There has been a number of studies on adaptive access control models in which the access control models are adjusted based on users' activity and other changes in the system.

Frias-Martinez *et al.* [22] are the first to introduce behavior-based access control (BB-NAC) policy where authorization mechanism is based on users' behavior profiles. The authorization policies are automatically generated and updated over time in order to adapt to behavioral changes. However, BB-NAC is based on manually pre-defined clusters of behavior that requires human intervention. Later, Frias-Martinez *et al.* in [21] present an enhanced version of the BB-NAC mechanism that automatically creates clusters of behaviors and propose an incremental learning algorithm to update the behavior-based access control policies. However, the generated models do not appropriately distinguish the behavioral patterns of different types of users.

Baracaldo and Joshi [7] incorporate risk and trust assessment in RBAC so the framework adapts to suspicious shifts in users' behavioral patterns by removing privileges when users' trust score is lower than a threshold. Such threshold is calculated based on a risk assessment process. They compute the risk values associated with permissions and roles by modeling a Coloured Petri-net. In their proposed approach, a user can activate a role if he has the minimum trust score that is required for that role.

Marinescu *et al.* [40] propose an approach for detecting authorization bugs in an online social network system and blocking access attempts that try to exploit such bugs. Their proposed approach learns authorization rules from data manipulation patterns and enforces such rules to prevent unauthorized access before code fixes are deployed.

Argento *et al.* [6] propose an adaptive access control model that exploits users' behavioral patterns to narrow their permissions when anomalous behavior is detected. Their approach

is based on Machine Learning techniques that dynamically refine ABAC policies to prevent exploitation of policy misconfigurations.

Al-Ali *et al.* [3] propose a self-adaptive authorization architecture in which a dynamic security adaptation controller updates the security rules of the system to adjust to the current system's situation.

2.3 ABAC Policy Misconfiguration Detection and Resolution

Policy misconfigurations may lead to a compromise of a system by permitting unauthorized access or denying legitimate ones. Moffett and Sloman in [44] analyze different kinds of overlap between policies which correspond to several types of policy conflicts. The main four categories include conflict of modality (positive/negative conflict), conflict between imperative and authority, conflict of priorities, and conflict of duties. Existing studies on access control policy misconfigurations focus on two types of anomalies in policy specification: policy rules conflicts and policy redundancy.

Hu *et al.* in [27] propose a policy-based segmentation technique that identifies policy anomalies and resolves them. Based on this technique, they divide the authorization space defined by an XACML policy into a set of disjoint segments. They identify an overlap relation (either conflicting or redundant) between XACML components associated with each segment. They also propose a policy conflict resolution strategy to resolve the identified anomalies.

El Hadj *et al.* in [18] propose a clustering-based approach for analyzing ABAC policy rules for detecting redundancy and conflicts between them. They first cluster authorization rules based on a rule similarity measure and then check for conflicting and redundant rules in each cluster.

El Hadj *et al.* in [19] propose a clustering-based approach for analyzing access logs to detect ABAC policy rules conflicts. Their proposed method first decomposes access logs into clusters and then analyzes each cluster separately. For each cluster, they find a cluster representative which later will be used for detecting contradicting access records (contradictions).

To detect conflicting policy rules, they first obtain a set of suspicious rules by analyzing contradictions and then find the conflicting rules from such set.

3.0 An Automatic Attribute-Based Access Control Policy Extraction from Access Logs

In this chapter, we present our clustering-based approach to extracting ABAC policy rules that contain both positive and negative attribute filters as well as positive and negative relation conditions. The proposed policy learning approach is effective even with an incomplete set of access logs and in presence of noise. As part of the proposed ABAC policy learning approach, we propose the rule pruning and policy refinement algorithms to enhance the quality of the mined policy and ease its maintenance. Finally, we propose a *policy quality metric* based on policy correctness and conciseness to be able to compare different sets of mined policy rules and select the best one based on the given criteria.

The rest of the chapter is organized as follows. In Section 3.1, we overview the ABAC model and its policy language as well as the unsupervised learning algorithm. In Section 3.2, we define the ABAC policy extraction problem, discuss the related challenges, and introduce the metrics for evaluating the extracted policy. In Section 3.3, we present the proposed ABAC policy extraction approach. In Section 3.5, we present the evaluation of the proposed approach on various sets of policies. Finally, Section 3.6 concludes the chapter.

3.1 Background

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

3.1.1 ABAC Model

In 2013, NIST published a "*Guide to ABAC Definition and Consideration*" [28], according to which, "*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/conditions, respectively.

Accordingly, U, O, S, OP are sets of users, objects, sessions, and operations in a system, and user attributes (A_u), object attributes (A_o), and session attributes (A_s) are mappings of subject attributes, object attributes, and environmental attributes as defined in the NIST Guide [28]. $E = U \cup O \cup S$ 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** [68]). 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** [68]). 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. $f_{a\_e}(dep1, crs) = \{cs101, cs601, cs602\}$ *indicates 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 8 *position* is a single-valued attribute while *crs* is a multi-valued attribute in Example 9. For simplicity, we only consider atomic attributes in this work. Actually, the process of extracting ABAC policy with multi-valued attributes is exactly the same as that with atomic attributes, however, we need to pre-process data to convert each multi-valued attribute to a set of atomic attributes. This can be done using various techniques such as defining dummy variables [56], 1-of-K scheme [10], etc. At the end of the process and when policy rules are extracted, we need one more step to convert back atomic attribute filters to the corresponding multi-valued attribute filters.

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

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

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*

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

Example 4. *Suppose* $A_u = \{dept, position, courses\}$. *The set of tuples* $\mathcal{F}_{\mathcal{U}} = \{\langle dept, CS\rangle, \langle position, grad\rangle\}$ *denotes a user attribute filter. Here, the graduate students in the CS department satisfy* $\mathcal{F}_{\mathcal{U}}$.

Definition 5. *(**Relation Condition**). A* relation condition *is defined as a set of tuples* $\mathcal{R} = \{\langle a, b|!b\rangle |\ a, b \in A \wedge\ a \neq b\}$. *Here* $\langle a, b\rangle$ *is a positive relation condition tuple that indicates* a *and* b *have the same values, and* $\langle a, !b\rangle$ *is a negative relation condition tuple that indicates* a *and* b *do not have the same values.*

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

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

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

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

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

An 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. *(**ABAC Rule** [68]). An* access rule *ρ is a tuple $\langle \mathcal{F}, \mathcal{R}, op | !op \rangle$, where $\mathcal{F}$ is an attribute filter, $\mathcal{R}$ is a relation condition, and op is an operation. !op is a negated operation that indicates the operation can have any value except op.*

Example 5. *Consider rule $\rho_1 = \langle \{\langle position, student \rangle, \ \langle location, campus \rangle, \langle type, article \rangle\}, \{\langle dept_u, dept_o \rangle\}, \ read \rangle$. It can be interpreted as "A student can read an article if he/she is on campus and his/her department matches the department of the article".*

Definition 10. *(**Rule Satisfaction**) An access request $q = \langle u, o, s, op \rangle$ is said to satisfy a rule ρ, denoted as $q \models \rho$, iff*

$$\langle u, o, s \rangle \models \mathcal{F} \wedge \langle u, o, s \rangle \models \mathcal{R} \wedge op_q = op_\rho.$$

Definition 11. *(**ABAC Policy** [68]). 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. *(**ABAC Policy Decision**). The decision of an ABAC policy π for an access request q denoted as $d_\pi(q)$ is* permit *iff:*

$$\exists \rho \in \pi : q \models \rho$$

otherwise, the decision is deny.

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

Table 1 summarizes the notations used in this chapter.

3.1.2 Policy Learning Algorithm

Policy learning algorithms try to infer a function that describes the structure of authorization data. In particular, given a set of authorization tuples, we employ a learning approach to mine and extract an *ABAC policy* that has high quality. ABAC policy extraction, in this case, can be considered as a mapping between authorization tuples to a set of clusters that are representative of the desired ABAC rules. Such a mapping can be expressed as a function, $h : \mathcal{X} \rightarrow \mathcal{Y}$, where:

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

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

3.2 Problem Definition

3.2.1 ABAC Policy Extraction Problem

Although organizations are interested in employing an ABAC model, adopting it is a big challenge for them. The manual development of such a policy is tedious and error-prone. *Policy Mining* techniques have been proposed to address such challenges in order to reduce the cost, time, and error of policy development/maintenance. ABAC policy mining algo-

Table 1: Notations

Notation	Definition
U,O, S, OP	Sets of users, objects, sessions, and operations
A_u, A_o, and A_s	Sets of user attributes, object attributes, and session attributes
$E = U \cup O \cup S$	Set of all entities
$A = A_u \cup A_o \cup A_s$	Set of all attributes
V_a	Attribute Range: set of all valid values for $a \in A$
$f_{a\_e}(e, a)$	Attribute Function: a function that maps an entity $e \in E$ to a value from V_a
$\mathcal{F} = \{\langle a, v \vert ! v \rangle \vert\ a \in A \wedge v \in V_a\}$	Attribute Filter
$\mathcal{R} = \{\langle a, b \rangle \vert\ a, b \in A \wedge\ a \neq b \wedge V_a = V_b\}$	Relation Condition
$q = \langle u, o, s, op \rangle$	Access Request
$t = \langle q, d \rangle$	Authorization Tuple, showing decision d made by the system for request q
$\mathcal{L}$	Access Log, set of authorization tuples
$\mathcal{L}^+ = \{\langle q, d \rangle \vert \langle q, d \rangle\ \in \mathcal{L} \wedge d = permit\}$	Positive Access Log
$\mathcal{L}^- = \{\langle q, d \rangle \vert \langle q, d \rangle\ \in \mathcal{L} \wedge d = deny\}$	Negative Access Log
$\rho = \langle \mathcal{F}, \mathcal{R}, op \vert ! op \rangle$	ABAC Rule
$\mathcal{P}$	Set of all policy rules
$\pi = \langle E, OP, A, f_{a\_e}, \mathcal{P} \rangle$	ABAC Policy
$d_\pi(q)$	The decision of an ABAC policy π for an access request q
$TP_{\pi\vert\mathcal{L}}$, $FP_{\pi\vert\mathcal{L}}$, $TN_{\pi\vert\mathcal{L}}$, and $FN_{\pi\vert\mathcal{L}}$	Relative True Positive, False Positive, True Negative, and False Negative Rates
$ACC_{\pi\vert\mathcal{L}}$	Relative Accuracy Rate
$F\text{-}score_{\pi\vert\mathcal{L}}$	Relative F-score
$WSC(\pi)$	Weighted Structural Complexity of policy π
$\mathcal{Q}_\pi$	Policy Quality Metric

rithms ease the migration to the ABAC framework by completely (or partially) automating the development of ABAC policy rules.

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

We define the ABAC policy extraction problem formally as follows:

Definition 13. *(**ABAC Policy Extraction Problem**). Let* $I =< E, OP, A, f_{a\_e}, \mathcal{L} >$*, where the components are as defined earlier, then the* ABAC policy extraction problem *is to find a set of rules* $\mathcal{R}$ *such that the ABAC policy* $\pi =< E, OP, A, f_{a\_e}, \mathcal{R} >$ *has high quality with respect to* $\mathcal{L}$*.*

3.2.2 Challenges and Requirements

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

1. *Correctness of Mined Policy*: The mined policy must be consistent with the original authorization log in that the access decision of the mined policy must result in the same access decision of the log entry. An inconsistent extracted policy may result in situations in which an originally authorized access is denied (*more restrictive*) or originally unauthorized access is permitted (*less restrictive*) by the system.
2. *Complexity of Mined Policy*: The policy mining algorithm should endeavor to extract a policy that is as concise as possible. Since the policy rules need to be manipulated by human administrators, the more concise they are, the more manageable and easier to interpret they would be. In addition, succinct rules are desirable as they are easier to audit and manage.
3. *Negative Attribute Filters*: The ABAC policy mining solution should support both positive and negative attribute filters which will result in more concise and manageable mined

policy.

4. *Relation Conditions*: The solution should support the extraction of relation conditions for policy mining in order to generate a more concise and manageable mined policy.
5. *Sparse Logs*: In the real world, the access log that is input to the policy mining algorithm may be sparse, representing only a small fraction of all possible access requests. The policy mining algorithm must be able to extract useful rules even from a sparse log.
6. *Mining Negative Authorization Rules*: An ABAC policy can contain both positive and negative rules which permit or deny access requests, respectively. The use of negative rules is helpful in situations where specifying exceptions to more general rules is important. Including negative policy rules would help in generating a more concise ABAC policy. Thus, the policy mining algorithm should be able to extract both positive and negative authorization rules.
7. *Noisy Authorization Log*: In the real world and with complex and dynamic information systems, it is possible to have a noisy authorization log consisting of over-assignments and under-assignments. These issues occur either due to a wrong configuration of the original authorization system or improper policy updates by administrators. The policy mining algorithm should be capable of extracting meaningful rules even in presence of an acceptable amount of noise in the input access log.
8. *Dynamic and Evolving Policies*: Modern information systems are often dynamic. The authorization needs of these systems and the attributes of the entities in the environment evolve rapidly. These changes will result in over-assignments or under-assignments. The proposed method should employ a mechanism to support the dynamicity of the information systems and their authorization policies and ease the maintenance of evolving systems.

Our proposed approach addresses all the requirements except the sixth one. Meaning that, for simplicity, we only focus on positive authorization rules, although, the proposed framework can be extended to extract both positive and negative AC rules. Positive AC rules are extracted from the permitted access requests and negative AC rules are extracted from the denied ones. However, the important step is to distinguish between the requests that

Table 2: State-of-the-art ABAC Rule Mining Techniques

	Xu *et al.* [68]	Medvet *et al.* [41]	Iyer *et. al* [29]	Cotrini *et al.* [14]	Proposed Approach
Policy Correctness	✓	✓	✓	✓	✓
Policy Complexity	✓	✓	✓	✓	✓
Negative Attribute Filters	✗	✗	✗	✗	✓
Relation Conditions	✓	✓	✓	✗	✓
Sparse Logs	✗	✓	✗	✓	✓
Negative Authorization Rules	✗	✗	✓	✗	✗
Noisy Authorization Log	✓	✗	✗	✗	✓
System Dynamicity	✗	✗	✗	✗	✓

were denied by an AC rule and the ones that were denied because they were not permitted by any rule.

Table 2 shows the challenges that are addressed by our proposed approach and how it improves upon the state-of-the-art policy mining techniques.

3.2.3 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 authorization decisions made by the extracted policy for a set of access requests should be similar to the decisions made by the original policy for that set of requests. As an example, if the decision of the original policy for an access request q is permit, then the decision of the mined policy for the same access request must be permit as well. If the mined policy denies the same access request, then we record this authorization tuple as a *False Negative*. We define *Relative True Positive*, *Relative False Positive*, *Relative True Negative*, and *Relative False Negative* rates, respectively, as follows:

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

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

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

Definition 15. *(**Relative False Positive Rate**). The relative false positive rate of π regarding $\mathcal{L}$ denoted as $FP_{\pi|\mathcal{L}}$ is the portion of negative access logs for which the decision of π is* permit*:*

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

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

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

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

The *relative precision* and *relative recall* are calculated as follows:

$$Precision_{\pi|\mathcal{L}} = \frac{TP_{\pi|\mathcal{L}}}{TP_{\pi|\mathcal{L}} + FP_{\pi|\mathcal{L}}}$$

$$Recall_{\pi|\mathcal{L}} = \frac{TP_{\pi|\mathcal{L}}}{TP_{\pi|\mathcal{L}} + FN_{\pi|\mathcal{L}}}$$

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

Definition 16. *(**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:*

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

As accuracy may be misleading in unbalanced data sets [64] (which is very probable in case of access logs), we use **relative F-score** to better evaluate the mined policy:

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

Policies with higher relative F-scores are better as they are more consistent with the original access log.

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 [46] and later extended for ABAC policies [68]. WSC is consistent with usability studies of access control rules, which indicates that the more concise the policies are the more manageable they become [8]. Informally, for a given ABAC policy, its WSC is a weighted sum of its elements. Formally, for an ABAC policy π with rules $\mathcal{P}$, its WSC is defined as follows:

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

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

$$WSC(\rho = \langle \mathcal{F}_{\mathcal{U}}, \mathcal{F}_{\mathcal{O}}, \mathcal{F}_{\mathcal{S}}, \mathcal{R}, op, d \rangle) = w_1 WSC(\mathcal{F}_{\mathcal{U}}) + w_2 WSC(\mathcal{F}_{\mathcal{O}}) + w_3 WSC(\mathcal{F}_{\mathcal{S}}) + w_4 WSC(\mathcal{R})$$

$$\forall s \in \{\mathcal{F}_{\mathcal{U}}, \mathcal{F}_{\mathcal{O}}, \mathcal{F}_{\mathcal{S}}, \mathcal{R}\} : WSC(s) = \sum |s|$$

where $|s|$ is the cardinality of set s and each w_i is a user-specified weight.

Van Rijsbergen proposes an effectiveness measure for combining two different metrics P and R in [49] as follows :

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

Given relative F-score and WSC measures for various mined policies resulting from running different mining algorithms over access log, it may not be straightforward to select the best algorithm and, hence, the mined policy with the highest quality. So, to be able to compare the quality of different mined ABAC policies, we combine the two metrics based on Van Rijsbergen's effectiveness measure [49] and define the **Policy Quality Metric** as follows:

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

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 reduction in the complexity with regards to the complexity of the most complex 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 the most complex mined policy.

Definition 17. *(**Most Complex Mined Policy**). The most complex mined policy is the mined policy with the highest weighted structural complexity. It is extracted by iterating through positive access log $\mathcal{L}^+$ and adding an access control rule for each authorization tuple if it's not already included in the mined policy. The corresponding rule for each authorization tuple includes all attributes of the user, object, and subject of that authorization tuple.*

Considering the equal importance of relative F-score and relative loss of complexity of 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 a higher F-score would have a higher policy quality. On the other hand, as the complexity of a policy increases, its quality will decrease. The intuition here is that once an extracted policy reaches a high F-score, adding additional rules will lead to a decrease in $\mathcal{Q}_\pi$.

For the most complex 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 most complex 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.

3.3 The Proposed Learning-based Approach

Our proposed learning-based ABAC policy extraction pro-cedure consists of the steps summarized in Figure 2.

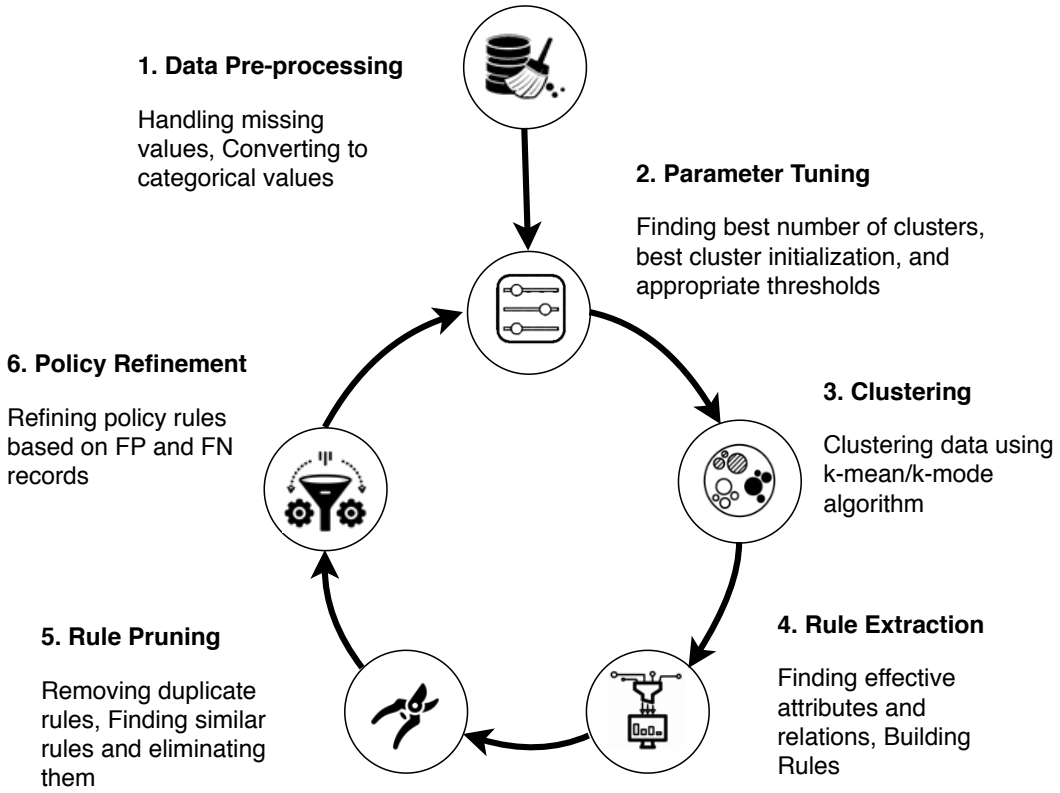


Figure 2: Overview of the Proposed Approach.

3.3.1 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. For example, in ABAC, environmental attributes deal with time, location, or dynamic aspects of the access control scenario. Hence, we need to pre-process and dis-

cretize such continuous variables to categorical ones (e.g. time of access to working hours and non-working hours) so our proposed algorithm is applicable to them.

We also need to handle *missing values* in this step. As the frequency of each attribute value is an important factor in our rule extraction algorithm (Section 3.3.4) for deciding if an attribute is effective or not, it is 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).

3.3.2 Selection of Learning Algorithm

We use the *K-modes algorithm* [13], which is a well-known unsupervised learning algorithm used for clustering categorical data. *K-modes* has been proved effective in mining ABAC policies [35]; this algorithm uses an initialization method based on both the distance between data points and the density of data points. Using both density and distance when initializing clusters helps avoid two problems: (i) clustering outliers as new clusters are based only on the distances; and (ii) creating new clusters surrounding one center based only on the density. Compared to a random initialization method, this method provides more robustness and better accuracy in the clustering process[13].

3.3.3 Parameter Tuning

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

3.3.3.1 Number of Clusters (k)

One of the main challenges in unsupervised learning is determining the number of clusters, k. In our sample policies, as we know the number of rules in each policy, we can set the number of clusters beforehand but in a real situation as we do not know the size of the rules in advance, making the correct choice of k is difficult. One of the popular methods for determining the number of clusters in an unsupervised learning model is the *Elbow Method*

[60, 23]. This method is based on the total within group sum of squares. k will be chosen as the number of clusters if adding another cluster doesn't give much better modeling of the data (i.e., the elbow point of the graph).

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

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

3.3.3.2 Cluster Initialization & Local Optima

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

3.3.4 Policy Rules Extraction

The main phase in our proposed approach is the extraction of ABAC policy rules. In the first step, we need to collect all the authorization tuples related to each rule of the policy. We use data clustering for this purpose. We divide the access log into clusters where the records in each cluster correspond to one AC rule in the system. This is done based on finding similar patterns between features (i.e., attribute values) of the records (i.e., access control tuples). In the second step, we extract the *attribute filters* of such a rule. We adapt the rule extraction algorithm in [35] and extend it to extract both positive and negative attribute filters. We define *effective positive attribute* and *effective negative attribute* as follows:

Definition 18. ***(Effective Positive (Negative) Attribute)***. *Let $S = \{\langle a, v\rangle\}$ be the set*

of all possible attribute-value pairs in a system; we define $\langle a_j, v_j \rangle \in S$ ($\langle a_j, !v_j \rangle \in S$) *as an* effective positive (negative) attribute *pair of* 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 *for* C_i *.*

Example 6. *Assume that* $A = \{dept, position, isChair, type, course, time, location\}$ *is the set of all possible attributes in a* university *information system. Here, these attributes are the features in the unsupervised learning algorithm. The training model clusters the training records into four different clusters. The centroid of the first cluster is shown in Table 3. In the university information system, the* ***type*** *has four possible values:* ***application***, ***gradebook***, ***roster***, *and* ***transcript***. *If this attribute has a normal distribution in the original data, then we expect the frequency of each of its values to be around 25% through all records. In the members of the first cluster, the frequency of* ***application*** *as the value of* ***type*** *is 60% which is 35% higher than the normal distribution. Having* $\mathcal{T} = 30\%$ *for the university data set,* ***type*** *becomes an effective positive attribute and* ***application*** *is its effective value in the corresponding rule. The attribute expression* $< type, application >$ *will be added to the extracted rule corresponding to this cluster. In the same way,* ***dept*** *and its value* ***CS***, ***time*** *and its value* ***BH*** *and* ***location*** *and its value* ***onCampus*** *are other effective positive attributes and their effective values, respectively. Eventually, the extracted rule from the first cluster is as follows:*

$$\rho_1 = \langle \{\langle dept, CS\rangle, \langle type, application\rangle, \langle time, BH\rangle, \langle location, onCampus\rangle\}, read\rangle$$

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

Definition 19. *(****Effective Positive (Negative) Relation****). Let* $R = \{\langle a, b\rangle\}$ *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 frequency of* a_j

Table 3: An Example of a Cluster Centroid

dept	*position*	*isChair*	*type*	*course*	*time*	*location*
CS	staff	false	application	cs101	BH	onCampus

equals b_j in all the records of cluster C_i is much higher (lower) than their frequency in the original data; this is determined based 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 for this cluster.

We note that the values of the thresholds $\mathcal{T}_P$, $\mathcal{T}_N$, θ_P, and θ_N will be different for each data set. To find the best threshold values for each data set, we run the rule extraction algorithm for different values of thresholds, and the values which result in the maximum accuracy over the cross-validation data set will be selected.

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

3.3.5 Policy Enhancement

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

3.3.5.1 Rule Pruning

During the rule extraction phase, it's possible to have two clusters that correspond to the same rule. As a result, the extracted rules of these clusters are very similar to each other. Having two similar rules in the final policy increases the complexity of the mined policy while it may not help the accuracy of the policy and as a result, it hurts the policy quality. To address such an issue, in the rule pruning step, we identify similar rules and eliminate the ones whose removal improves the policy quality more. If eliminating neither

Algorithm 1: Effective Attribute Extraction Algorithm

Input: C_i, A, V, $\mathcal{L}$, $\mathcal{T}_P$, $\mathcal{T}_N$

Output: $\mathcal{F}$

```
1  procedure Extract_Attribute_Filters
2      F ← ∅;
3      forall a ∈ A do
4          forall v_j ∈ V_a do
5              if Freq(v_j, C_i) − Freq(v_j, L) > T_P then
6                  F ← F ∪ ⟨a, v_j⟩;
7              end
8              if Freq(v_j, L) − Freq(v_j, C_i) > T_N then
9                  F ← F ∪ ⟨a, !v_j⟩;
10             end
11         end
12     end
13     return F
```

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 [31] as follows:

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

Based on this, we calculate the similarity between two rules ρ_1 and ρ_2 as follows:

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

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

Algorithm 2: Effective Relation Extraction Algorithm

Input: C_i, A, $\mathcal{L}$, θ_P, θ_N

Output: $\mathcal{R}$

```
1  procedure Extract_Relations
2      R ← ∅;
3      forall a ∈ A do
4          forall b ∈ A and b ≠ a do
5              if Freq(a = b, C_i) - Freq(a = b, L) > θ_P then
6                  R ← R ∪ ⟨a, b⟩;
7              end
8              if Freq(a = b, L) - Freq(a = b, C_i) > θ_N then
9                  R ← R ∪ ⟨a, !b⟩;
10             end
11         end
12     end
13     return R
```

3.3.5.2 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 7. *Consider the following two rules:*

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

$$\rho_2 = \langle\{\langle position, faculty\rangle, \langle dept_u, EE\rangle, \langle type, gradebook\rangle\}, setScore\rangle$$

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 a larger number of *FNs* as an access request that would be permitted by the original rule will be denied by the restricted rule.

Algorithm 3: Rule Pruning Algorithm

Input: $\mathcal{P}$

Output: $\mathcal{P}$

```
1  procedure Rule_Pruning
2      q ← Calc_Quality(P);
3      forall ρ_i ∈ P do
4          forall ρ_j ∈ P and ρ_i ≠ ρ_j do
5              if Similarity(ρ_i, ρ_j) > 0.5 then
6                  P_i ← P/ρ_i;
7                  P_j ← P/ρ_j;
8                  q_i ← Calc_Quality(P_i);
9                  q_j ← Calc_Quality(P_j);
10                 if q_i >= q and q_i >= q_j then
11                     P ← P_i;
12                 end
13                 if q_j >= q and q_j >= q_i then
14                     P ← P_j;
15                 end
16             end
17         end
18     end
19     return P
```

On the other hand, an extracted rule is more relaxed compared to the original rule if it misses some of the filters. In Example 7, ρ_1 is more relaxed than ρ_2. Such a relaxed rule would result in more *FPs* as it permits access requests that should be denied as per the original policies.

To address these issues, we propose a *policy refinement* procedure which is shown in Algorithm 4. Here, we try to refine the mined policy (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 the FP records. Here, our goal is to extract the patterns that are common between access requests that were permitted based on the mined policy while they should have been denied based on the original policy.

In each step of refinement, a rule from π_m that is similar to a rule from π_{FN} or π_{FP} based on the Jaccard similarity (Section 3.3.5.1) is selected and then refined in two ways as discussed below.

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 more restrictive. To resolve this issue, for each rule $\rho_i \in \pi_{FN}$:

- if there is a similar rule $\rho_j \in \pi_m$ then we refine ρ_j 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 might be 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 the 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

Input: A, $\mathcal{L}$
Output: π_m

procedure *Refine_Policy*
  $\mathcal{FN} \leftarrow Get\_FNs(\pi_m, \mathcal{L})$;
  $\pi_{FN} \leftarrow Extract\_Policy(\mathcal{FN})$;
  forall $\rho_i \in \pi_{FN}.\mathcal{P}$ **do**
    $R_s \leftarrow Get\_Similar\_Rules(\pi_{FN}.\mathcal{P}, \pi_m.\mathcal{P})$;
    if $|R_s| = 0$ **then**
      $\pi_m.\mathcal{P} \leftarrow \pi_m.\mathcal{P} \cup \rho_i$;
    end
    else
      forall $\rho_j \in R_s$ **do**
        forall $\mathcal{F} \in \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$ **do**
          $\mathcal{F}_{\rho_j} \leftarrow \mathcal{F}_{\rho_j} \backslash (\mathcal{F}_{\rho_j} \backslash \mathcal{F}_{\rho_i})$;
        end
      end
    end
  end
  $\mathcal{FP} \leftarrow Get\_FPs(\pi_m, \mathcal{L})$;
  $\pi_{FP} \leftarrow Extract\_Policy(\mathcal{FP})$;
  forall $\rho_i \in \pi_{FP}.\mathcal{P}$ **do**
    $R_s \leftarrow Get\_Similar\_Rules(\pi_{FP}.\mathcal{P}, \pi_m.\mathcal{P})$;
    if $|R_s| \,!= 0$ **then**
      forall $\rho_j \in R_s$ **do**
        forall $\mathcal{F} \in \mathcal{F}_{\mathcal{U}} \cup \mathcal{F}_{\mathcal{O}} \cup \mathcal{F}_{\mathcal{S}} \cup \mathcal{R}$ **do**
          $\mathcal{F}_{\rho_j} \leftarrow \mathcal{F}_{\rho_j} \cup (\mathcal{F}_{\rho_i} \backslash \mathcal{F}_{\rho_j})$;
        end
      end
    end
  end
  return π_m

3.4 Time Complexity

In this section, we analyze the time complexity of our proposed approach. Our proposed approach starts with clustering the access log using a k-mode algorithm. K-mode algorithm has $O(n^2)$ time complexity where n is the size of the input data (here $n = |\mathcal{L}|$). We then analyze the *Effective Attribute Extraction* algorithm (Algorithm 1). The most time consuming part of Algorithm 1 is the calculation of the frequency of the attribute-value pairs in the

original data. Let n be the number of records ($n = |\mathcal{L}|$) and d be the number of attributes ($d = |A|$) in the access log, the time complexity of Algorithm 1 is $O(nd)$. The nested loops in Algorithm 1 compare the frequency of each attribute-value pair in a given cluster with its frequency in the original data, so the time complexity of these loops is $O(m^d)$ where m is the number of values each attribute has in the system ($m = |V_a|$ where $a \in A$). So, the total running time of Algorithm 1 is $O(nd+m^d)$. Similarly, the running time of the *Effective Relation Extraction* algorithm (Algorithm 2) is $O(nd + m^d)$.

The main part of the *Rule Pruning* algorithm (Algorithm 3) compares each pair of extracted rules and calculates their similarity. So, the running time is $O(r^2d)$, where r is the number of rules ($r = |\mathcal{P}|$). The algorithm also calculates the quality of each subset of policy rules over the access log which results in $O(nr)$ time complexity. So, the overall running time of Algorithm 3 is $O(r^2d + nr)$.

The *Policy Refinement* algorithm (Algorithm 4) first extracts the rules (effective attribute/relation pairs) corresponding to FP and FN records, so its time complexity is $O(n'd)$ where n' is the size of FP/FN records. In the second part of the algorithm, each extracted rule is compared with FP/FN rules and its attribute-value pairs are updated. Hence, the time complexity of this part is $O(r'rd)$ where r' is the size of FP/FN rules. So the overall time complexity of Algorithm 4 is $O(n'd + r'rd)$. In practice, the size of FP/FN records and their corresponding rules is significantly smaller than the size of original records and the corresponding extracted rules. So, in the worst case, the time complexity of Algorithm 4 is $O(nd + r^2d)$.

K-mode algorithm is the most time consuming part of the proposed ABAC policy mining framework, therefore, the running time of the proposed ABAC policy mining algorithm is $O(n^2)$. In worst case and in case of a complete access log (where every combination of attribute values are presented in the log), size of the input records are exponential to the number of attributes ($n = m^d$). On the other hand, the size of the extracted rules is much smaller than the input size. Hence, the overall time complexity of the proposed algorithm is $O(m^{2d})$.

Table 4 compares the time complexity of the state-of-the-art policy mining techniques with our proposed algorithm.

Table 4: Comparison of Time Complexity of the State-of-the-art ABAC Rule Mining Techniques

Proposed Approach	Xu *et al.* [68]	Medvet *et al.* [41]	Iyer *et. al* [29]	Cotrini *et al.* [14]
$O(m^{2d})$	$O(m^{3d})$	$O(m^{3d})$	$O(m^{2d}d^5)$	$O(m^{4d})$

3.5 Experimental Evaluation

We have implemented a prototype of our proposed approach. Here, we present our experimental evaluation.

3.5.1 Datasets

We perform our experiments on multiple datasets including synthesized and real ones. The synthesized access logs are generated from two sets of ABAC policies. The first one is a manually written set of policies that is adapted from [68] to be compatible with our policy language. The second one includes a completely randomly generated set of policies. To synthesize our input data, for each ABAC policy (i.e., *University Policy*, *Healthcare Policy*, etc.), a set of authorization tuples is generated and the outcome of the ABAC policy for each access right is evaluated. The authorization tuples with *permit* as their outcomes are the inputs to our unsupervised learning model.

Our real datasets are built from access logs provided by Amazon in Kaggle competition [5] and available in the UCI machine learning repository [47].

Manual Policy - University: This policy is adapted from [68] and it controls access of different users including students, instructors, teaching assistants, etc., to various objects (applications, gradebooks, etc.).

Manual Policy - Healthcare: This policy is adapted from [68] and is used to control access by different users (e.g. nurses, doctors, etc.) to electronic health records (EHRs) and EHR items.

Table 5: Details of the Synthesized and Real Policies

#	π	$\lvert\mathcal{P}\rvert$	$\lvert A\rvert$	$\lvert V\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{L}^+\rvert$	$\lvert\mathcal{L}^-\rvert$
π_1	UniversityP	10	11	45	2,700K	231K	2,468K
π_2	HealthcareP	9	13	40	982K	229K	753K
π_3	ProjectManagementP	11	14	44	5,900K	505K	5,373K
π_4	UniversityPN	10	11	45	2,700K	735K	1,964K
π_5	HealthcarePN	9	13	40	982K	269K	713K
π_6	ProjectManagementPN	11	14	44	5,900K	960K	4,918K
π_7	Random Policy 1	10	8	27	17K	2,742	14K
π_8	Random Policy 2	10	10	48	5,250K	245K	5,004K
π_9	Random Policy 3	10	12	38	560K	100K	459K
π_{10}	Amazon Kaggle	-	10	15K	32K	30K	1897
π_{11}	Amazon UCI	-	14	7,153	70K	36K	34K

Manual Policy - Project Management: This policy is adapted from [68] and it controls access by different users (e.g. department managers, project leaders, employees, etc.) to various objects (e.g. budgets,schedules and tasks).

Random Policies: The authorization rules for this policy are generated completely randomly from random sets of attributes and attribute values. These randomly generated policies provide an opportunity to evaluate our proposed algorithm on access logs with various sizes and with varying structural characteristics. However, we note that, the performance of our algorithm on random policies might not be representative of its performance in real scenarios and over real policies.

Real Dataset - Amazon Kaggle: The Kaggle competition dataset [5] includes access requests made by Amazon's employees over two years. Each record in this dataset describes an employee's request to a resource and whether the request was authorized or not. A record consists of the employee's attribute values and the resource identifier. The dataset includes more than 12,000 users and 7,000 resources.

Real Dataset - Amazon UCI: This dataset is provided by Amazon in the UCI machine learning repository [47]. It includes more than 36,000 users and 27,000 permissions. Since the dataset contains over 33,000 attributes, our focus in this experiment is narrowed only to

the most requested 8 permissions in the dataset.

Partial Datasets: To check the efficiency of the proposed algorithm over sparse datasets, we generate sparse datasets (partial datasets) by randomly selecting authorization tuples from the complete dataset. For example, a 10% sparse (partial) dataset is generated by randomly selecting 10% of tuples from the complete access logs.

Noisy Datasets: To check the efficiency of the proposed algorithm over noisy datasets, we generate noisy datasets by randomly reversing the decision of authorization tuples. For instance, a 10% noisy dataset is generated by randomly reversing the decision of 10% of authorization tuples in the complete access logs.

For each of the manual policies, we consider two different sets of policy rules; the first one only contains positive attribute filters and relations while the second one includes both positive and negative attribute filters and relations. We have included these policies in Appendix A.

Table 10 shows the details of the manual and random access log datasets. In this table, $|\mathcal{P}|$ shows the number of rules in the original policy, $|A|$ and $|V|$ show the number of attributes and attribute values and $|\mathcal{L}|$, $|\mathcal{L}^+|$, $|\mathcal{L}^-|$ show the number of access control tuples, the number of positive access logs, and the number of negative access logs in the given dataset, respectively.

3.5.2 Experimental Setup

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

We use kmodes library [17] for clustering our data. The initialization based on density (CAO) [13] is chosen for cluster initialization in kmodes algorithm.

To find optimal k, we apply various methods to test different values of k. Figure 3 shows the *Elbow Method* graphs for three different sample policies as well as the actual value of k. As we can see in the figure, for these sample policies the curves are ambiguous and there are no clear elbows. So it seems that elbow method may not necessarily help in ABAC policy

extraction.

Figure 4 shows the accuracy of models as a function of the number of clusters for three different datasets. The approach works perfectly for University Policy as the selected k and the actual k are both equal. For Healthcare Policy the selected k (i.e., 8) is one level smaller than the actual k (i.e., 9). For Project Management Policy, the selected k (i.e., 12) based on the policy accuracy is one level higher than the actual k (i.e 11). After pruning (Sectin 3.3.5.1), the final policy has 9 rules. Again, using the technique of policy refinement (Section 3.3.5.2), we can revive the remaining rules and improve the model accuracy.

To generate the synthesized access log $\mathcal{L}$, we brute force through all attributes A and their values V_a to produce all possible combinations for the tuples. This method was used to generate a complete access log for the random and manual policy datasets. We generate two sets of partial datasets; the 10% partial datasets are used to check the efficiency of the proposed approach over sparse datasets (Table 6) and the 0.1% partial datasets are used to compare the proposed approach with previous work (Table 7). We also generate a set of noisy datasets to check the efficiency of the proposed algorithm over noisy access log. The results of such experiments are reported in Table 6.

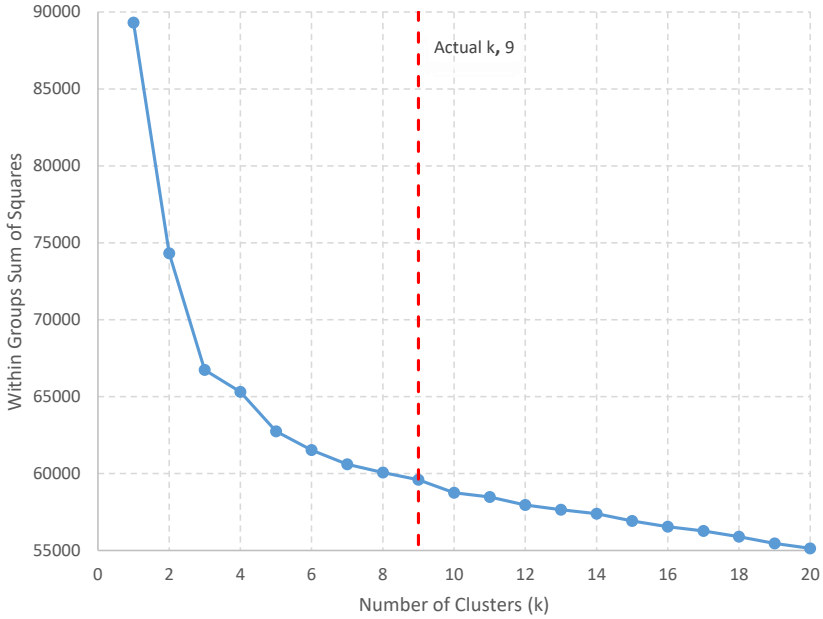
For all experiments, the optimal thresholds for selecting effective attributes and relations are between 0.2 and 0.3.

3.5.3 Results

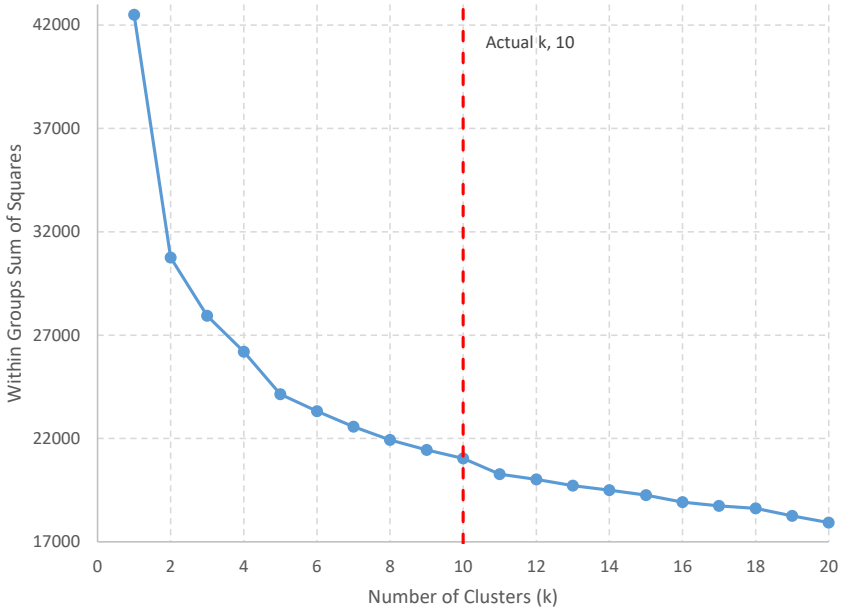
We first evaluate the performance of our policy mining algorithm on complete datasets. Table 6 shows the results of these experiments.

Our second set of experiments is on partial datasets. The algorithm proposed by Xu and Stoller [68] and the approach presented by Cotrini *et al.* [14] are not able to handle complete datasets as these datasets are huge. To be able to compare the performance of our proposed algorithm with their work, we generated 0.1% sparse (partial) datasets and run all algorithms over these partial datasets. The results of these experiments are shown in Table 7 and Figures 8, 9, and 10.

The algorithm proposed by Xu and Stoller [68] and the approach presented by Cotrini *et*

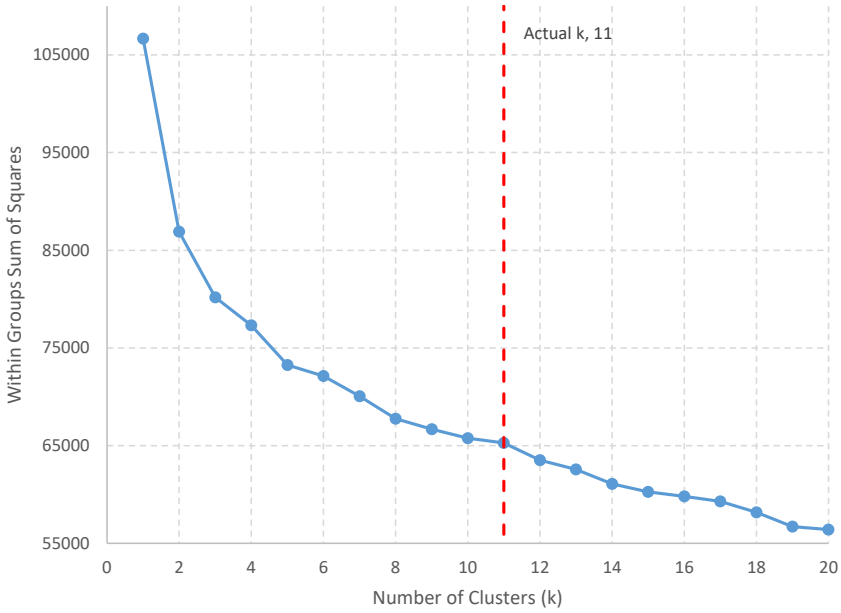


(a) Healthcare Policy



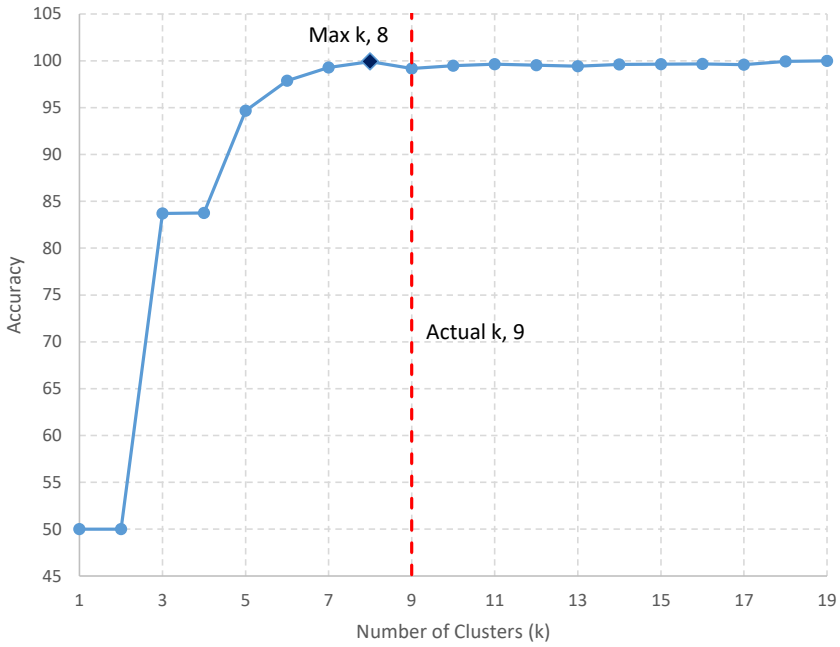
(b) University Policy

Figure 3: Elbow Method: K-means Clustering SSE vs. Number of Clusters for three different case studies

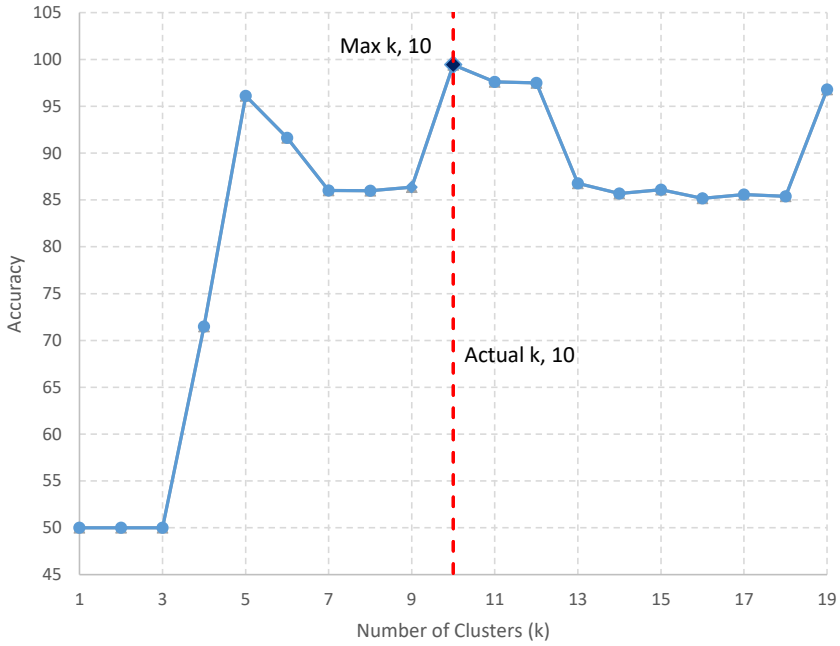


(c) Project Management Policy

Figure 3: Elbow Method: K-means Clustering SSE vs. Number of Clusters for three different case studies (Cont.)

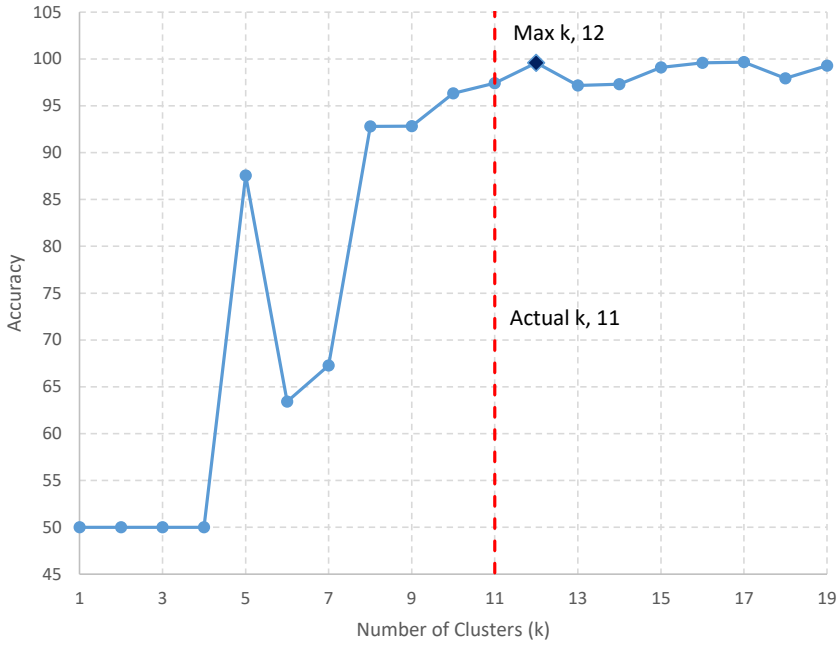


(a) Healthcare Policy



(b) University Policy

Figure 4: Accuracy of Clustering Models vs. Number of Clusters for three different sample policies



(c) Project Management Policy

Figure 4: Accuracy of Clustering Models vs. Number of Clusters for three different sample policies (Cont.)

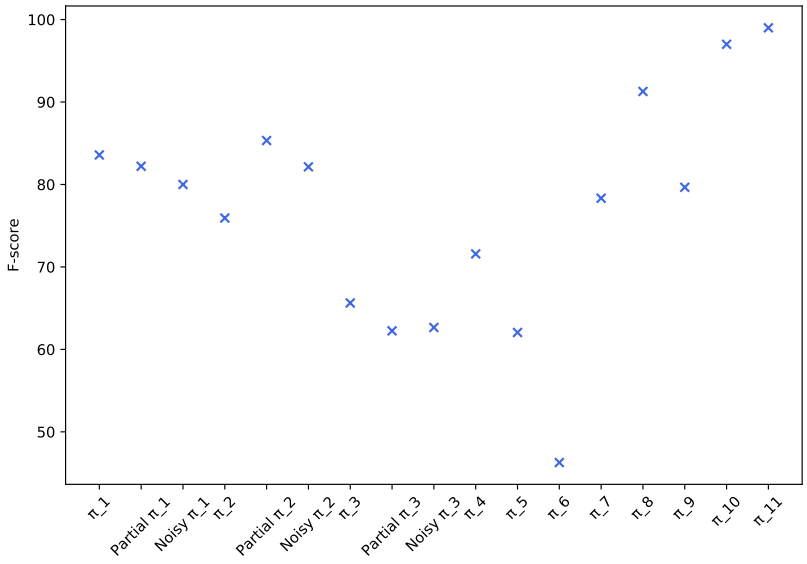


Figure 5: The F-Score of Our Proposed Approach over Complete Datasets

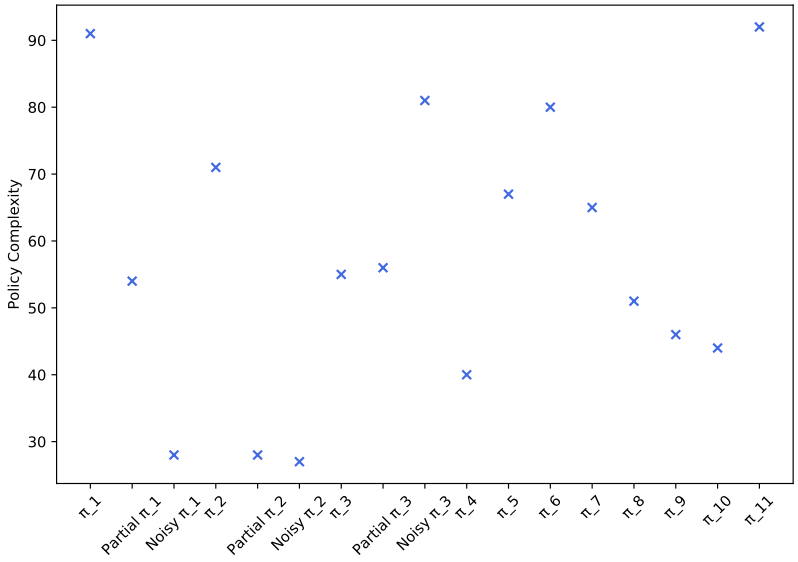


Figure 6: The Complexity (WSC) of Our Proposed Approach over Complete Datasets

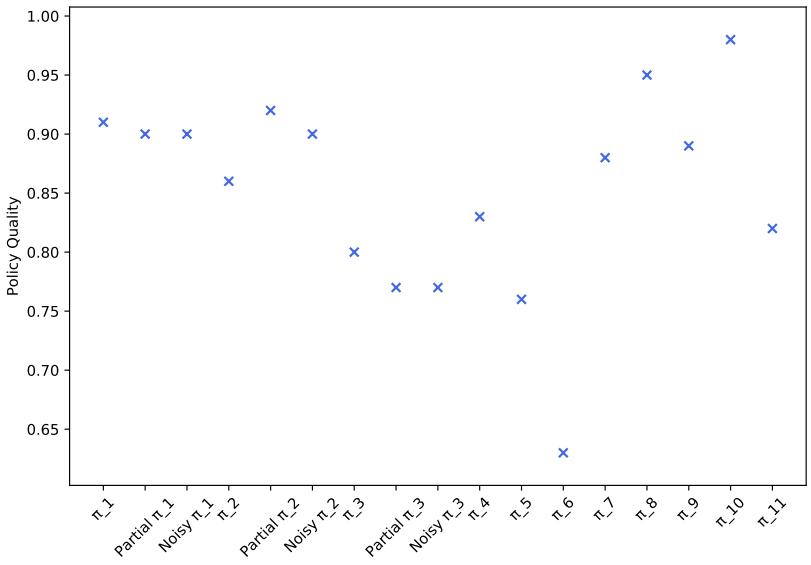


Figure 7: The Policy Quality of Our Proposed Approach over Complete Datasets

al. [14] do not generate policy rules with negative attribute filters and relations, however, we report the results of their algorithms over datasets related to policy rules including negations (policies π_4, π_5, π_6) to show how the quality of mined policies would be impacted if the mining algorithm does not extract rules that include negation.

3.5.3.1 The F-Score of the Mined Policies

Table 6 shows the final $F\text{-}score_{\pi|\mathcal{L}}$ of our proposed approach after several rounds of refinement over all complete datasets. As we can see in Table 6, the proposed approach achieves a high F-score across all experiments except for π_6. π_6 is a very complex dataset with both positive and negative attributes and relation filters including 14 attributes, 44 attribute values, and around six million access records. The final policy quality for this dataset is around 0.63, which is acceptable considering the complexity of the policy.

Table 7 and Figure 8 show the comparison of the F-Scores of policies mined by our proposed approach with that of previous work over partial datasets (with 0.1% of the complete

datasets). As we can see, the F-Score of policies mined by our algorithm is very close to the one done by the approach proposed by Cotrini *et al.* [14]. Actually, our proposed approach outperforms theirs in half of the experiments.

The algorithm proposed by Xu and Stoller [68] does not extract any policy rules in the four experiments. In two out of the four experiments that produced results, the F-Score of their approach is higher than those of other works.

3.5.3.2 The Complexity of the Mined Policies

In Table 6, we can see the final WSC of the policies mined by our proposed approach. All extracted policies have a complexity lower than 100 which is much lower than those of the most complex policies for individual datasets. According to *Definition 17*, the most complex policy for each dataset has the same complexity as the original positive access log ($\mathcal{L}^+$). Given numbers in Tables 10 and 6, the most complex policies for these scenarios are thousands of times more complex than the extracted policies by our approach.

We compare the complexity of the policies mined by different ABAC mining algorithms in Figure 9. Among three different approaches, the Cotrini *et al.* algorithm extracts the most complex policies with WSC greater than 1000 for some cases. The complexity of the policies mined by our algorithm is very close to the one extracted by the approach proposed by Xu and Stroller [68].

3.5.3.3 The Policy Quality of the Mined Policies

Finally, Table 6 shows the quality of the extracted policies through our proposed approach. We can see that out of all datasets that our proposed algorithm was applied on, around 75% of the cases reached the policy quality of more than 0.8, which is significant, considering the huge size of original access logs (each more than 30K records).

According to Figure 10, in most cases, the policy quality of the policies mined by our proposed approach is higher than those of the policies extracted by other ABAC mining algorithms.

Table 6: Results of Our Proposed Approach on Various Synthesized and Real Policy Datasets

π	Total Running Time (s)	Optimal k	$\mathcal{P}_{mined}$	$ACC_{\pi\|\mathcal{L}}$	$F\text{-}score_{\pi\|\mathcal{L}}$	WSC_{orig}	WSC_{mined}	$\mathcal{Q}_{\pi}$
π_1	9376.556	15	20	97.5%	83.6%	33	91	0.91
Partial π_1 (10%)	1994.769	15	13	97.29%	82.21%	33	54	0.90
Noisy π_1 (10%)	4979.56	10	8	96.94%	80%	33	28	0.90
π_2	2180.745	18	18	85.49%	75.93%	33	71	0.86
Partial π_2 (10%)	4787.98	10	8	96.94%	85.33%	33	28	0.92
Noisy π_2 (10%)	7339.91	8	15	72.22%	82.13%	33	27	0.90
π_3	7795.44	15	17	95.6%	65.63%	44	55	0.80
Partial π_3 (10%)	1347.29	6	10	95.2%	62.24%	44	56	0.77
Noisy π_3 (10%)	1912.72	15	15	94.47%	62.66%	44	81	0.77
π_4	13662.62	7	16	86.7%	71.58%	33	40	0.83
π_5	8681.64	15	15	78.11%	62%	33	67	0.76
π_6	12905.78	20	17	88.05%	46.28%	44	80	0.63
π_7	24.63	8	20	93%	78.33%	33	65	0.88
π_8	13081.20	10	14	99.12%	91.28%	33	51	0.95
π_9	2266.68	8	16	92.17%	79.66%	33	46	0.89
π_{10}	265.3	15	20	94%	97%	-	44	0.98
π_{11}	1010.43	24	25	98.49%	99%	-	92	0.82

3.5.4 Discussion and Limitations

Our proposed approach is able to achieve a practical level of performance when applied to both synthesized and real datasets. In the case of synthesized datasets, the proposed approach is capable of mining policies containing both positive and negative attribute filters from complete datasets. On the other hand, our proposed approach shows potential for use in sparse datasets. In addition, the real datasets contain a large number of attributes and attribute values as shown in Table 10. The ability of our proposed approach in mining high-quality policies for these datasets shows that the size of attributes and attribute values have minimal impact on the effectiveness of our approach.

The proposed approach is based on an unsupervised clustering algorithm. Since finding the proper number of clusters is a challenge related to clustering algorithms, our approach is affected by this issue as well. The same issue will also be valid in finding the best thresholds to extract effective attributes and relations.

We note that, as the proposed algorithm is based on tuning multiple parameters, it is possible that it gets stuck in minimum optima. For this reason, we do not claim that it will extract the policy with the highest quality in every scenario, nor do we claim that extracting

Table 7: Comparison of Our Proposed Approach with Previous Work on Various Synthesized and Real Policy Datasets

Mining Alg.	π	Time (s)	$ACC_{\pi\|\mathcal{L}}$	$F\text{-}score_{\pi\|\mathcal{L}}$	$\mathcal{P}_{\pi_{mined}}$	$WSC(\pi)$	$\mathcal{Q}_{\pi}$
Xu and Stoller [68]	Partial π_1 (0.1%)	348	96.1%	70.9%	7	23	0.83
Cotrini *et al.* [14]		126	80.74%	45.3%	132	508	0.58
Proposed Approach		7.3	96%	74.2%	7	29	0.85
Xu and Stoller [68]	Partial π_2 (0.1%)	3015	96.74%	95.47%	9	33	0.81
Cotrini *et al.* [14]		529	72.72%	64%	65	272	0.75
Proposed Approach		7.9	79.78%	68.23%	13	49	0.81
Xu and Stoller [68]	Partial π_3 (0.1%)	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$
Cotrini *et al.* [14]		3587	91.57%	54.124%	24	77	0.70
Proposed Approach		11.44	94.96%	51.31%	12	55	0.78
Xu and Stoller [68]	Partial π_4 (0.1%)	2897	74.19%	19.3%	7	23	0.32
Cotrini *et al.* [14]		204	93.55%	88.5%	385	1389	0.86
Proposed Approach		15	89.3%	80%	10	40	0.89
Xu and Stoller [68]	Partial π_5 (0.1%)	6740	93.26%	86.72%	9	33	0.92
Cotrini *et al.* [14]		3587	86.46%	79.2%	123	462	0.83
Proposed Approach		8.8	87.2%	76.3%	15	66	0.86
Xu and Stoller [68]	Partial π_6 (0.1%)	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$
Cotrini *et al.* [14]		2848	82.75%	62.66%	31	100	0.77
Proposed Approach		22.67	81.2%	49.4%	12	44	0.66
Xu and Stoller [68]	π_{10}	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$
Cotrini *et al.* [14]		237	84.25%	91.39%	1055	2431	0.92
Proposed Approach		265.3	94%	97%	20	44	0.98
Xu and Stoller [68]	π_{11}	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$	$-^*$
Cotrini *et al.* [14]		1345	70.93%	75.64%	466	1247	0.85
Proposed Approach		1010.43	98.49%	99%	24	92	0.99

$*$ Xu and Stoller [68] did not terminate nor produced any output for these datasets even after running for more than 24 hours.

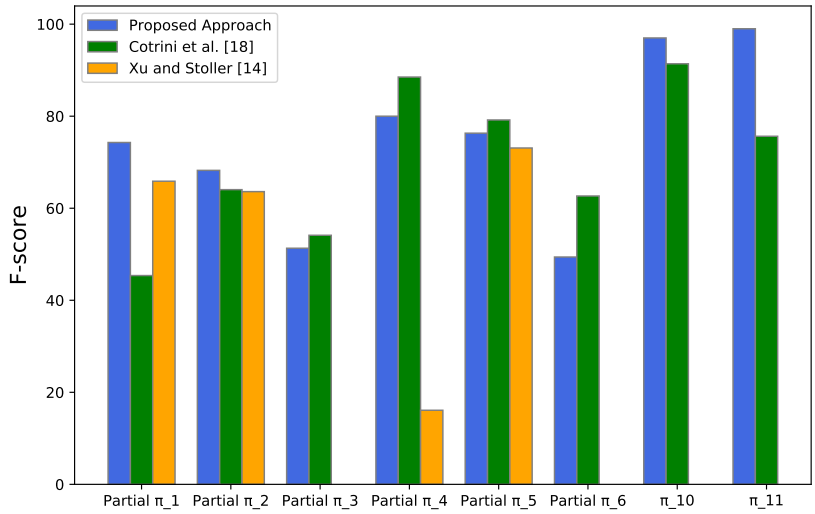


Figure 8: The F-Score of the Policies Mined by ABAC Mining Algorithms

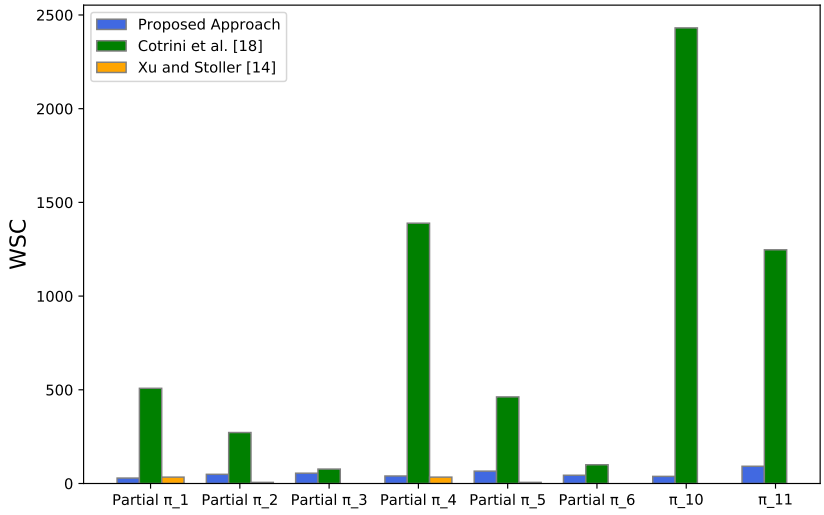


Figure 9: The Complexity of the Policies Mined by ABAC Mining Algorithms

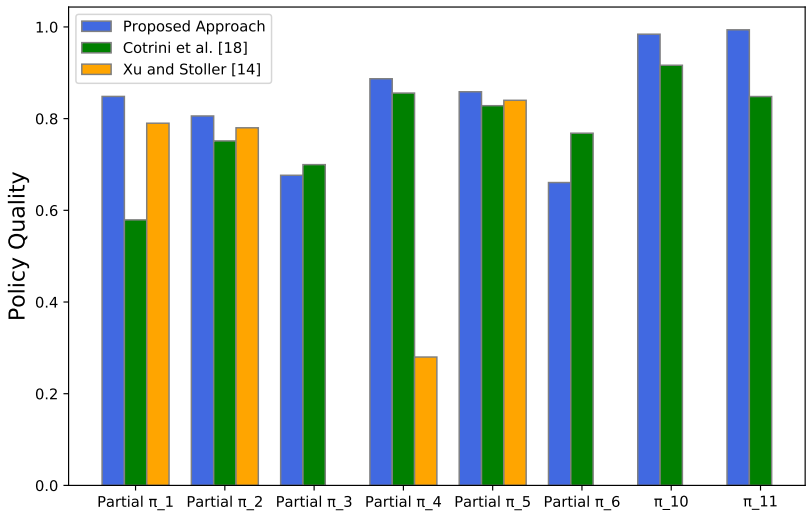


Figure 10: The Quality of the Policies Mined by ABAC Mining Algorithms

rules with negative attribute filters and relations would always result in policy with higher quality; however, by trying more randomization in cluster initialization and a wider range of parameters, we can get one that is closer to global optima.

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

3.6 Conclusion

We propose an unsupervised learning based approach to automating an ABAC policy extraction process. 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 proposed rule pruning and policy refinement algorithms. Such refinement algorithms are based on false positive and false negative records and they help in increasing the quality of the mined policy.

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

4.0 Adaptive ABAC Policy Learning: A Reinforcement Learning Approach

In this chapter, we present our RL-based approach for adapting an ABAC policy in a system. We utilize RL and, more specifically, contextual bandit, to establish a mapping between access requests and the appropriate authorization decisions for such requests. The authorization engine (AE) is considered as an agent in our proposed framework and its authorization decision (permit or deny) is an action according to the state of the system. Attributes of entities involved in an access request as well as the contextual factors form the state of the system. The AE learns authorization policies by interacting with the environment without having explicit knowledge of the original access control policy rules. We propose four methods for initializing the learning model and a planning approach based on hierarchies over attribute values to accelerate the learning process. We develop the proposed adaptive ABAC policy learning model, using a home IoT environment as a running example.

The rest of the chapter is organized as follows. In Section 4.1, we overview the Attribute Based Access Control, Reinforcement Learning, Home IoT, and its authorization framework. In Section 4.2, we propose the reinforcement learning based framework for adaptive attribute based authorization learning while focusing on Home IoT as a running example. In Section 4.3, we evaluates the proposed approach. Finally, Section 4.5 concludes the chapter.

4.1 Background

In this section, we overview Attribute Based Access Control (ABAC), Reinforcement Learning (RL), Home IoT, and its authorization framework. Table 8 summarizes the notations used in this chapter.

4.1.1 Attribute Based Access Control

In this section, we consider the ABAC model that has the following components:

U, O, and OP represent sets of users, objects, and operations in a system, and UA, OA, and EA correspond to sets of user attributes, object attributes, and environmental attributes, respectively. $E = U \cup O$ and $ATTR = UA \cup OA \cup EA$ are the sets of all entities and all attributes in the system.

Definition 20. *(**Attribute Range**). Given an attribute $attr \in ATTR$, the* attribute range *$V(attr)$ is the set of all valid values for $attr$ in the system.*

Definition 21. *(**Attribute Filter**). An* attribute filter *is defined as a set of tuples $F = \{\langle attr, v\rangle | \ attr \in ATTR \text{ and } v \in V(attr)\}$. Here $\langle a, v\rangle$ is an attribute filter tuple that indicates $attr$ has value v.*

Definition 22. *(**Access Request**). An* access request *q is a tuple $q = \langle u, o, op, ea\rangle$ where user $u \in U$ is the requester requesting to perform operation $op \in OP$ on object $o \in O$ while environmental attributes $ea \in EA$ holds.*

Definition 23. *(**Authorization Tuple/Access Log**). An* authorization tuple *is a tuple $l = \langle q, d\rangle$ containing the final decision d made by the authorization engine for request q. An* Access Log *$\mathcal{L}$ is a set of such tuples.*

The decision d of an authorization tuple can be *permit* or *deny*. The tuple with *permit* decision means that user u can perform operation *op* over an object o under environmental attributes *ea*. The authorization tuple with *deny* decision means the user cannot get such access.

Definition 24. *(**ABAC Rule**). An* ABAC rule *ρ is a tuple $\rho = \langle uaf, oaf, eaf, op, d\rangle$, where uaf, oaf, and eaf are user attribute filter, object attribute filter and environmental attribute filter, respectively, op is a corresponding operation, and d shows the decision of the ABAC rule for such combination of attributes and requested operation.*

Definition 25. *(**ABAC Policy**). An ABAC policy π_{ABAC} is a tuple $\pi_{ABAC} = \langle E, ATTR, OP, \mathcal{P}\rangle$ where E, ATT, OP, and $\mathcal{P}$ are sets of entities, attributes, operations, and ABAC rules in the system, respectively.*

Table 8: Notations

Notation	Definition
U, O, and OP	Sets of users, objects, and operations
UA, OA, and EA	Sets of user attributes, object attributes, and environmental attributes
$E = U \cup O$	Set of all entities in the system
$ATTR = UA \cup OA \cup EA$	Set of all attributes in the system
$V(attr)$	Set of all valid values for $attr$ in the system
$F = \{\langle attr, v\rangle \mid attr \in ATTR \text{ and } v \in V(attr)\}$	Attribute Filter
$q = \langle u, o, op, ea\rangle$	An access request where user u is the requester requesting to perform operation op over object o while environmental conditions ea holds
AE	Authorization Engine (i.e. RL agent)
$l = \langle q, d\rangle$	Authorization tuple including decision d made by the AE for request q
$\mathcal{L}$	Access log (i.e. set of authorization tuples)
$\rho = \langle uaf, oaf, eaf, op, d\rangle$	ABAC policy rule
$\mathcal{P}$	Set of all ABAC policy rules in the system
$\pi_{ABAC} = \langle E, ATTR, OP, \mathcal{P}\rangle$	ABAC policy
$\mathcal{S}$, $\mathcal{A}$, and π	RL state space, action space, and policy
$s_t = [ua_t, oa_t, ea_t, op]$	system state at time step t
a_t and r_t	chosen action and reward (feedback) at time step t
TP_w, TN_w, FP_w and FN_w	reward function items, showing agreement or disagreement between owner and agent decision
λ_{TP}, λ_{TN} , λ_{FP} and λ_{FN}	weight of reward function items
d_w and d_{AE}	decision of owner of an object and the authorization agent for an access request, respectively
$owner(o)$	a function that returns owners of object o
VH_{attr}	Attribute Value Hierarchy
$get\_state(q_t)$	a function that returns a state s_t corresponding to an access request q_t
$get\_request(s)$	a function that returns an access request q_s corresponding to a state s

4.1.2 Reinforcement Learning

Reinforcement Learning (RL) refers to a set of algorithms that train an agent to make a sequence of decisions through an interaction with an unknown environment to attain a goal (i.e., maximize the expected cumulative discounted reward) [57, 42, 63, 58, 16]. The environment is modeled as a Markov decision process (MDP). At each time step t, the agent observes the current state s_t of the environment from the state space $\mathcal{S}$. The agent takes an action a_t from the action space $\mathcal{A}$ according to a policy π. Following the action, the agent receives a reward signal r_t, and the state of the environment transits to s_{t+1}. The goal of the agent is to attain a policy that maximizes the expected return R which is the sum of future discounted rewards

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

where $\gamma \in [0, 1]$ is a discount rate that determines the significance of future rewards.

Policy π is a probability distribution over actions given the states,

$$\pi : \pi(s, a) \rightarrow [0, 1].$$

Here, $\pi(s, a)$ is the likelihood of action a in state s. For each MDP, there exists an optimal policy π^* that is at least as good as all other policies, expressed as follows:

$$\pi^* \geq \pi \qquad \forall \pi$$

The value function of policy π in state s is the expected total reward for an agent starting at state s:

$$V^{\pi}(s) = \mathbb{E}[R_t \mid s_t = s]$$

Similarly, the Q-function of policy π is defined as the expected return from choosing action a in state s, and the following policy π afterward:

$$Q^{\pi}(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a]$$

The optimal Q-function denotes the maximum reward we can expect by selecting action a in state s:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Knowing the optimal Q-function, we can easily extract the optimal policy by selecting an action which results in the maximum $Q^*(s,a)$ for each state:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \qquad \forall s \in \mathcal{S}$$

It is common to use a *function approximator* to estimate the function Q, especially when there are many possible {state, action} pairs. A function approximator has a set of adjustable parameters, θ, referred to as *policy parameters*.

4.1.2.1 Contextual Bandit

Contextual bandit is an extension of n-armed bandit problem which is a simplified form of RL problems. A contextual bandit algorithm formulates a round-by-round interaction between a learner and an environment while introducing contextual information in the interaction loop [38]. Actually, the contextual bandit strategy to reinforcement learning frames choices between separate actions in a given context. In the contextual bandit problem, the learner observes a context and uses the contextual information to select the best action in each round. Then the learner perceives a cost/reward for the chosen action only. Contextual bandit algorithms utilize side information (context) to help with real-world decision-making scenarios. Contextual bandit based approaches are suitable for various real-world interactive machine learning problems. Specifically, they operate nicely on choosing actions in dynamic environments where options change rapidly, and the set of available actions is limited. As a learner receives limited feedback from the environment, exploration plays an important role in contextual bandit algorithms.

The contextual bandit approach fits the adaptive authorization policy learning problem nicely, specially in dynamic environments, as the available set of actions (i.e. deny or permit) is limited and each state of the environment is independent from other states. Hence, the

perceived cost/reward in each round is based on the chosen action only. In addition, contextual information (i.e. attributes of involved entities) plays an important role in choosing the appropriate action.

Considering the aforementioned properties, we propose to model the ABAC policy learning as a contextual bandit problem.

4.1.3 Home IoT and Its Authorization Framework

IoT is one of the emerging paradigms that is evolving quickly. It allows users to be connected to various objects and exchange data across their networks. Home IoT is one example of such an environment where users in a household interact with internet-connected devices that include sensors and appliances in their daily routines. Home IoT devices are usually managed through a central controller that handles the communication between devices, enforces users' authorization policies, and often allows for the execution of programs and applications over such devices. ABAC model is a good fit as an authorization approach for such an environment. However, despite traditional information systems with single-user devices (e.g. computers, phones, etc.), in a home IoT environment, multiple users interact with each device. As a result, traditional authorization frameworks fail to provide usable and flexible access-control specification and enforcement in such settings. With numerous IoT devices, a wide range of attributes and contextual factors, and various relationships between users and devices, it is unthinkable to expect ordinary users to be able to manage such complex authorization requirements and enforcement infrastructure. In this paper, we focus on a home IoT as a running example and apply our proposed approach for adapting an access control model for it.

We focus on a capability-centric authorization model instead of a device-centric one as suggested by [26] where a capability is defined as an operation (e.g., play music) that can be performed on an IoT device (e.g., Google Home). The capability-based authorization model is more suitable for home IoT, as each home IoT device may have several capabilities with different sensitivity levels that results in different access control policy rules for various capabilities of an IoT device. For example, assume that a smart door lock has various ca-

pabilities such as unlocking, locking, viewing the state of the lock, deleting logs, and so on. In this example, unlocking the door is more sensitive than viewing a state of the lock and hence needs a more restricted AC policy rule.

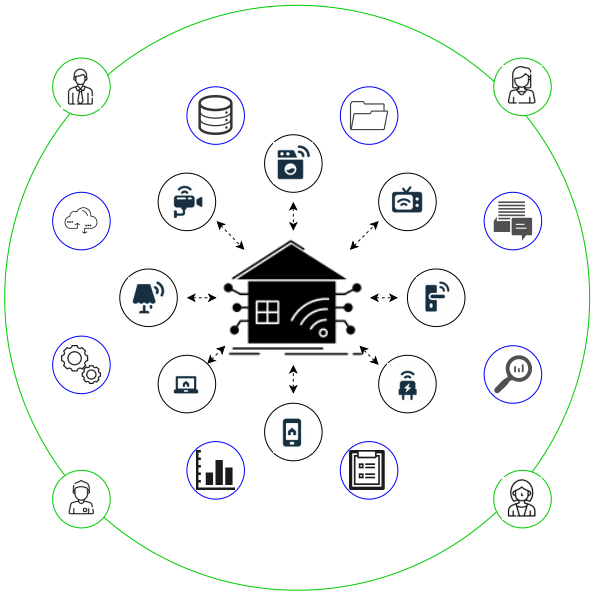


Figure 11: Overview of Home IoT Environment.

As shown in Figure 11, a home IoT environment consists of a set of users interacting with a set of home IoT devices. Each IoT device has a set of capabilities and a set of owners/administrators who manage the provisioning of such capabilities. Here, users, devices, and capabilities are equivalent to set of users U, set of objects O, and set of operations OP, respectively, in the corresponding ABAC authorization model.

When a user wants to access a capability of a device (e.g. the user wants to play music on Google Home), his access request is directed to the authorization engine (AE). The AE evaluates the access request and sends its authorization decision back to the system to be enforced. The authorization decision is either *permit* or *deny*, granting or preventing requested access to the user, respectively. The AE evaluates the access requests based on the states of the system and the policy it learned through reinforcement learning algorithm.

4.2 Adaptive Attribute-Based Policy Learning

In this section, we introduce the ABAC-RL, an adaptive reinforcement learning based ABAC policy learning framework. We present the key components of ABAC-RL in detail and propose several methods for improving the model.

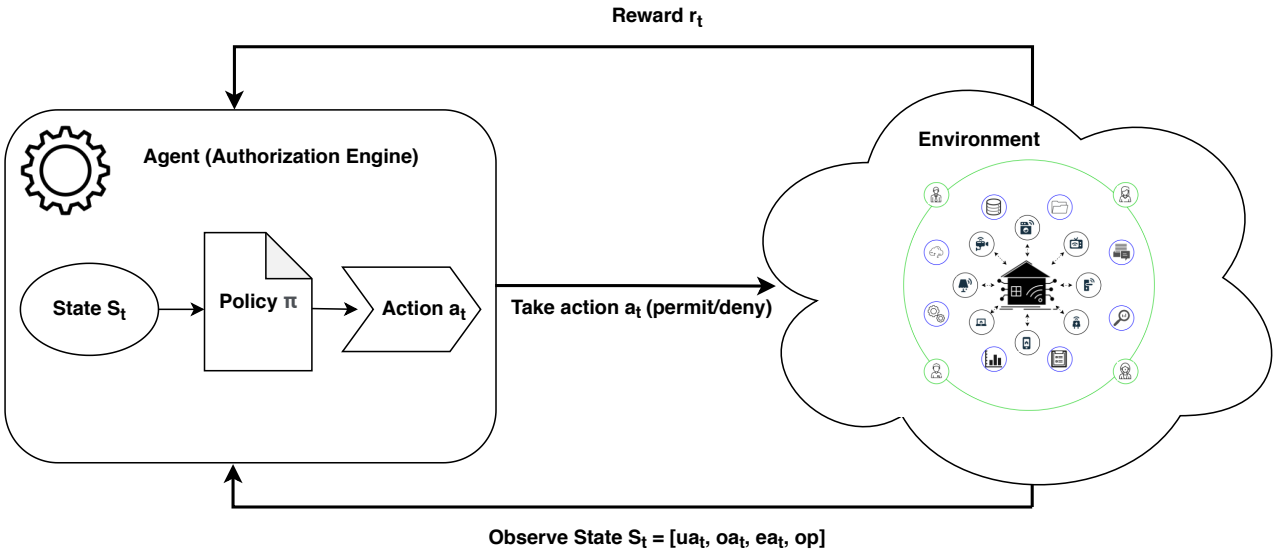


Figure 12: Adaptive Reinforcement Learning Based ABAC Policy Learning

4.2.1 ABAC-RL Framework

Figure 12 shows the overall ABAC-RL framework. Here, an agent corresponds to the authorization engine (AE) that adapts an ABAC model through a feedback control loop by interacting with the environment. The environment is considered to mainly have the users/administrators of the system. An agent begins by knowing nothing or very little about the authorization policies of the administrators of the system and learns through reinforcements (i.e. rewards/penalties received through feedback from the environment showing how well it is deciding on access requests).

At each time step t (i.e. when an access request q_t is received by the system), the AE agent observes a state, s_t, from the state space, $\mathcal{S}$, and accordingly takes an action, a_t, from the action space, $\mathcal{A}$, determining authorization decision based on the policy, π. In our model, the state observed by an AE agent for characterizing the environment consists of several parts: attributes ua_t of user u_t requesting the access, attributes oa_t of object o_t, environmental attributes, ea_t, and operation op_t that is requested to be performed on object o_t. Hence, the state can be shown as $s_t = [ua_t, oa_t, ea_t, op_t]$. Accordingly, the action (i.e., the authorization decision) is either *permit* or *deny*, granting or preventing requested access to the requester, respectively.

Following an action, an agent receives a reward, r_t, determined by the feedback from the users of the system showing their agreement/disagreement with the authorization decision of the AE agent. For this purpose, the authorization decision of AE for each request is recorded. Later on, the administrator of the requested object checks the records and submits his feedback to the system showing his agreement/disagreement with the authorization decision of AE. We note that administrators usually give feedback when they do not agree with the authorization decision of AE, otherwise they may not give any feedback. Hence, we assume no feedback from the administrator means he agreed with the decision.

The feedback provided by administrators of a requested object will be used in the reward function to calculate the reward of the action that was taken by an AE agent for the corresponding access request. The reinforcement learning algorithm will use the calculated reward to update its policy with the goal of making better decisions in the future. We explain the details of the reward function in the following section.

4.2.2 Reward Function

At each time step and for each access request, the AE agent takes an action (i.e. permit or deny) according to the current state, $s_t \in \mathcal{S}$ and based on its policy π. The goal of the agent is to minimize unauthorized access to the objects. In order to reach this objective, the authorization decision of the agent should match the collective authorization decisions of the owners/administrators of the corresponding object. Therefore, the reward function

Table 9: Reward Function Items Based on Agent and Owner decisions for an Access Request

$TP_w = \begin{cases} 1 & \text{if } d_w = permit \text{ and } d_{AE} = permit \\ 0 & \text{otherwise} \end{cases}$	$FP_w = \begin{cases} 1 & \text{if } d_w = deny \text{ and } d_{AE} = permit \\ 0 & \text{otherwise} \end{cases}$
$FN_w = \begin{cases} 1 & \text{if } d_w = permit \text{ and } d_{AE} = deny \\ 0 & \text{otherwise} \end{cases}$	$TN_w = \begin{cases} 1 & \text{if } d_w = deny \text{ and } d_{AE} = deny \\ 0 & \text{otherwise} \end{cases}$

d_w and d_{AE} represents decision of the owner of an object and the authorization agent for an access request, respectively.

is set up in a way to achieve this goal. The input to the reward function is the feedback of all owners/administrators of the requested object. If the feedback shows an agreement with AE's decision, the agent should get a positive reward to learn that its decision was correct. On the other hand, when the feedback shows a disagreement between an owner and the decision of an AE, the agent should receive a negative reward so it will adjust its policy with the goal of making better decisions in the future. Hence, we propose a reward function as follows:

$$r_t = \sum_{w \in owner(o_t)} \lambda_{TP} \cdot TP_w + \lambda_{TN} \cdot TN_w - \lambda_{FP} \cdot FP_w - \lambda_{FN} \cdot FN_w$$

where, $owner(o_t)$ is a function that returns all owners (or administrators) of object o_t that was requested at time step t, and TP_w, TN_w, FP_w and FN_w are reward function items that are calculated based on the decision of AE and the feedback from owner of the device (see details in Table 9), and λ_{TP}, λ_{TN} , λ_{FP} and λ_{FN} are their corresponding weights. Here, true positives and true negatives are represented as positive rewards as the goal of the agent is to maximize them while both false positives and false negatives are represented as penalties, as the goal is to minimize these measures.

4.2.3 Policy Initialization Techniques

Reinforcement learning algorithms start with the initialization of a target policy. We propose four different approaches for initializing an ABAC-RL policy in our framework. These initialization methods may overlap and the real-world system can employ any of these approaches or all of them, together. Here again, we use Home-IoT as our running example.

4.2.3.1 Initialization with General AC Policy Rules

As studied by He *et al.* in [26], a set of desired access control policies are typically consistent among IoT Home users. For example, it is desirable that all users be capable of controlling the lights and thermostats when they are at home. As another example, deleting the lock log should be denied for all users of the system except the owners. Multiple such candidate general policies have been suggested by He *et al.* in [26]. These general AC policies are a good starting point for initializing the ABAC-RL policy.

4.2.3.2 Initialization with Default User Settings

At the beginning of employing an RL for learning authorization policies of a Home-IoT system or when a new user (e.g. a baby sitter) is added to the Home-IoT, the owners of the devices can define a few access control policy rules as a default setting of the authorization framework. These settings are employed by the RL algorithm to initialize the corresponding policies so it will converge to the desired authorization policies faster and prevent over-privileged or under-privileged accesses. For example, the default authorization policy rule for "neighbor" could be *deny* while the default policy rule for "parent" could be *permit*.

4.2.3.3 Default Decision for Capabilities

Different capabilities of Home IoT devices have different levels of sensitivity. For a highly sensitive capability, any over-privileged access could result in a serious loss of security or privacy and is against the *principle of least privilege* [50]. The default decision (i.e., RL action) for an access request corresponding to such a capability should be *deny*. On the other hand,

for nonsensitive capabilities, an under-assignment could result in an authorized user being denied from accessing the capability which is very inconvenient. Such under-assignments could adversely affect the availability of an object/capability and should be avoided. Hence, the default decision for an access request corresponding to such a capability should be *permit*. The default decisions for such capabilities can be set as part of the initialization of the authorization policy. For example, the default decision for "check_temperature" is *permit*.

4.2.3.4 Initialization with Past Access Logs

Information systems' owners often desire to employ modern access control models and migrate from outdated authorization models to new ones. Various policy learning methods have been proposed to automate such migration [68, 14, 29, 35, 33]. In the case of policy learning, available access logs from former AC models can be used to form a new refined policy. In our proposed approach, we utilize the available access logs to initialize the corresponding RL model.

4.2.4 Policy Learning with Planning

In a reinforcement learning model, an agent improves its decision making strategy by interacting with the environment. In each state, the agent chooses an action and receives feedback for the chosen action. Based on the received feedback, it will update its policy for decision making in that state. By experiencing more and more state-action-feedback sequences, the agent policy will get close to the optimal policy. However, for a large space of states, the agent does not have complete information for all the states. Its information is partial as it may not visit all the states. The agent can improve its information on an unseen state by utilizing the information of neighboring states.

In our ABAC-RL policy learning framework, we propose a planning strategy to enhance the learned policy in presence of partial information. Such a strategy will help the agent to make a better decision for previously unseen states. The proposed planning algorithm is based on pre-defined hierarchies of attribute values in the system. Attribute value hierarchy defines a quasi ordering between different values of an attribute in the model. Formally, we

define an attribute value hierarchy as follows:

Definition 26. ***Attribute Value Hierarchy*** *For each attribute* $attr \in (UA \cup OA \cup EA)$*, the attribute value hierarchy* $VH_{attr} \subseteq V(attr) \times V(attr)$ *is a partial order on* $V(attr)$ *called closeness relation, written as* $v_1 \succeq v_2$*, where* v_1 *is called the upper value and* v_2 *is called the lower value in the relation.*

The proposed planning strategy identifies neighboring states and concludes the access decision of one state based on another. For this purpose, we formally define the neighboring states as follows:

Definition 27. ***Neighboring States*** *Two states* s_{t_1} *and* s_{t_2} *are called neighboring states where for all attribute* $attr \in (UA \cup OA \cup EA)$*, except* $attr'$ *the two states have the same value,* $attr' = v_1$ *in state* s_{t_1} *and* $attr' = v_2$ *in state* s_{t_2}*, and* $attr'$ *has a value hierarchy where* v_1 *and* v_2 *are in closeness relationship (i.e.* $v_1 \succeq v_2$ *or* $v_2 \succeq v_1$*).*

The *Upper Neighboring State* and *Lower Neighboring State* are defined as follows to distinguish between concluding "permit" and "deny" decisions for neighboring states.

Definition 28. ***Upper and Lower Neighboring States*** *Given two neighboring states* s_{t_1} *and* s_{t_2} *where the value of all their attributes except* $attr'$ *are the same and* $attr' = v_1$ *in state* s_{t_1} *and* $attr' = v_2$ *in state* s_{t_2}*, the state* s_{t_1} *is called the upper neighboring state, and state* s_{t_2} *is called the lower neighboring state if* v_1 *and* v_2 *are in a closeness relationship and* $v_1 \succeq v_2$*.*

The intuition behind the planning strategy is that the authorization decision for a state is similar to the authorization decision for its neighboring state (with higher/lower order attribute value in the same hierarchy). In our planning process, when the AE receives feedback for an access log, it records the decision for the access log and all its neighboring states. For each access log, we only consider the first-level neighboring states, meaning that we only consider the unseen states that differ in one attribute value from the given state. Algorithm 5 shows the details of the planning process.

As an example assume that "minor" and "teenager" are two possible values of attribute "age_range" in Home-IoT. There is a closeness relation between these two attribute values written as $teenager \succeq minor$ meaning that states $s_{teenager}$ and s_{minor} are neighboring states

Algorithm 5: Planning Algorithm

Input: $\mathcal{L}$

Output: $\mathcal{L}$

```
1  procedure Planning
2      forall l = [q_t, d_t] ∈ L do
3          s_t = get_state(q_t);
4          if d_t == "permit" then
5              S_Upper = get_upper_neighbors(s_t);
6              forall s ∈ S_Upper do
7                  if s ∉ L then
8                      q_s = get_request(s);
9                      L = L ∪ [q_s, "permit"];
10                 end
11             end
12         end
13         if d_t == "deny" then
14             S_Lower = get_lower_neighbors(s_t);
15             forall s ∈ S_Lower do
16                 if s ∉ L then
17                     q_s = get_request(s);
18                     L = L ∪ [q_s, "deny"];
19                 end
20             end
21         end
22     end
23     return L
```

and given that all other attribute values of these two states are equal, permitting an action in the state with $age\_range = teenager$ is expected if authorization decision for an action

in the state including $age\_range = minor$ is *permit*. So if an action is permitted for a minor child in a home IoT environment it will be permitted for a teenager in the same situation. On the other hand, we decide to deny an action in a state for a minor child if the action is denied for a teenager in the same circumstances.

4.3 Experimental Evaluation

We have implemented a prototype of our proposed approach presented in Section 4.2. In this section, we present our experimental evaluation.

4.3.1 Experiment Setup

Evaluation over real authorization data would be ideal, however, as we only have access to one real access log from Amazon, we developed various sample policies, attribute data, and their corresponding synthesized access logs. We have developed two sets of sample policies, one with manually written policy rules and the other with randomly generated policy rules. Each sample policy is the desired policy that the ABAC-RL algorithm aims to learn. We generate a synthesized access log for each sample policy. To generate the synthesized access log, we brute force through all attributes and their values to produce all possible combinations for the access tuples. We use this method to generate a complete access log for the manual and random policies. In the synthesized access logs, each access tuple corresponds to an access request and the desired authorization decision based on the original policy. To check the feasibility of our approach over sparse data, we also consider a partial dataset for each sample policy. We produce the sparse dataset (partial dataset) by randomly selecting access log records from the complete dataset. The ABAC-RL learning algorithm is run over each dataset to see how it will learn the authorization policy gradually.

The log generation and the proposed adaptive ABAC-RL learning algorithm are written in Python 3. We use Vowpal Wabbit (VW) [1] for implementing various contextual bandit

[1] https://vowpalwabbit.org

methods (see details in Section 4.3.2) as well as a supervised learning algorithm. The supervised learning algorithm is a one-against-all logistic regression method provided by VW. The experiments were performed on a 64-bit Windows 10 machine having 8 GB RAM and an Intel Core i7 processor.

The performance of each method on a dataset with n records is measured by the *progressive validation loss* (P.V.Loss) [11] that is calculated as follows:

$$PVL = \frac{1}{n}\sum_{t=1}^{n} c_t a_t$$

where a_t is the loss for each chosen action ($a_t = 0$ if the chosen action matches the decision of the original access tuple and $a_t = 1$ otherwise) in time step t and c_t is the corresponding cost ($c_t = 1$ in our experiments). In our experiments, we assume each object has one owner and the weights of reward function items (λs) equal 1.

4.3.1.1 Datasets

To evaluate the proposed model, we perform our experiments on multiple datasets including synthesized and real ones. The synthesized authorization records are generated based on two sets of ABAC policies: a set of ABAC policies with manually written policy rules and one with randomly generated policy rules. The real dataset is built from the records provided by Amazon in the Kaggle competition [5].

Real Dataset - Amazon Kaggle: The Kaggle competition dataset [5] includes access requests made by Amazon's employees in a two-year period. Each access tuple in this dataset corresponds to an employee's request to access a resource and shows whether the access was permitted or not. The access log consists of the employees' attribute values and the resources' identifier. The dataset includes more than 12,000 users and 7,000 resources.

Synthesized dataset - manually written policies: We developed a set of sample policies including manually written rules and attribute data. We generated synthesized access log data for each manual policy. The access log consists of access requests and the desired decision based on the corresponding policy rules. To generate the synthesized access log, we brute force through all attributes and attribute values to produce all of their possible

Table 10: Details of Datasets

π_{ABAC}	$\lvert\mathcal{P}\rvert$	$\lvert A\rvert$	$\lvert V\rvert$	$\lvert\mathcal{L}\rvert$
Kaggle [5]	-	9	15626	33K
Manual Policy 1 (π_{m1})	11	5	30	6K
Manual Policy 2 (π_{m2})	11	5	29	5K
Manual Policy 3 (π_{m3})	38	5	44	48K
Synthetic Policy 1 (π_{s1})	5	8	30	21K
Synthetic Policy 2 (π_{s2})	10	10	34	70K
Synthetic Policy 3 (π_{s3})	15	12	37	200K

combinations. This procedure generates a complete access log for each sample policy. To check the feasibility of our approach over sparse data, we also consider partial datasets for each sample policy. We produce each partial dataset by randomly selecting the access log records from the complete dataset. All incomplete logs in our experiments are 20% partial datasets that are generated by randomly selecting 20% of tuples from the complete access logs. Appendix B shows the details of the manually written policies.

Synthesized dataset - randomly generated policies: The authorization rules for these policies are generated completely randomly from random sets of attributes and attribute values. These randomly generated policies provide an opportunity to evaluate our proposed approach on datasets with various sizes and with varying structural characteristics.

Table 10 shows the details of the access log datasets. In this table, $|\mathcal{P}|$ shows the number of ABAC rules in the original policy, $|ATTR|$ and $|V| = \sum_{attr \in ATTR} |V(attr)|$ show the number of attributes and attribute values, respectively, and $|\mathcal{L}|$ shows the size of the access log.

4.3.2 Contextual Bandit Algorithms

We empirically evaluate four contextual bandit algorithms for our proposed model. The algorithms are as follows:

- ϵ-greedy algorithm [37]: The algorithm greedily exploits the best action learned with probability $1 - \epsilon$ and explore uniformly over all actions with probability ϵ.
- Explore-first: The algorithm exclusively explores the first k trials and then exploits the best action learned afterward.
- Bagging: The algorithm trains multiple policies using bootstrapping. Given the context, the algorithm samples from the distributions over the actions provided by these policies.
- Online cover [1]: The algorithm explores all available actions while keeping only a small subset of policies active.

4.3.3 Experimental Results

In this section, we compare various contextual bandit algorithms for policy learning over different databases. We also evaluate our proposed ABAC-RL framework against several baselines. Table 11 reports the result of the best parameter settings for each algorithm over different databases.

4.3.3.1 Kaggle Access Control Dataset

The results of different contextual bandit algorithms, as well as a supervised classifier over the Kaggle dataset [5], are shown in Fig 13 and reported in Table 11. The graph shows the progressive validation loss (P.V.Loss) [11] for each algorithm. We can see that under full information and by using a supervised algorithm, we get a pretty good predictor with an average loss rate of 5.5%. All contextual bandit algorithms show comparable performance on this dataset. Specifically, both Online Cover (with a cover set of size 2) and Explore-first (with first 10 records) algorithms get an average loss rate of 5.8% which is very impressive considering the fact that compared to the full information supervised learning scenario they only have access to partial information.

Table 11: Progressive validation loss, best hyperparameter values, and running times of various algorithms on different datasets

Databases		**Algorithms**					
		ϵ-greedy	Explore-first	Bagging	Online Cover	Planning	Supervised
Kaggle [5]	**P.V.Loss**	0.065	0.058	0.059	0.058	-	0.055
	Best Hyperparam	$\epsilon = 0.01$	10 first	2 bags	cover $n = 2$	-	NA
	Running Time (s)	0.588	0.400	0.617	0.459	-	0.351
Manual Policy 1 (π_{m1})	**P.V.Loss**	0.16	0.2	0.13	0.11	-	0.14
	Best Hyperparam	$\epsilon = 0.01$	1500 first	4 bags	cover $n = 2$	-	NA
	Running Time (s)	0.115	0.132	0.148	0.162	-	0.130
Manual Policy 2 (π_{m2})	**P.V.Loss**	0.13	0.17	0.11	0.08	-	0.10
	Best Hyperparam	$\epsilon = 0.02$	300 first	2 bags	cover $n = 2$	-	NA
	Running Time (s)	0.095	0.132	0.093	0.121	-	0.139
Manual Policy 3 (π_{m3})	**P.V.Loss**	0.07	0.1	0.04	0.03	0.02	0.05
	Best Hyperparam	$\epsilon = 0.01$	10 first	2 bags	cover $n = 2$	cover $n = 2$	NA
	Running Time (s)	0.346	0.355	0.377	0.388	.401	0.301
Synthetic Policy 1 (π_{s1})	**P.V.Loss**	0.15	0.14	0.09	0.08	0.07	0.12
	Best Hyperparam	$\epsilon = 0.02$	1500 first	10 bags	cover $n = 1$	cover $n = 1$	NA
	Running Time (s)	0.203	0.275	0.303	0.217	.295	0.232
Synthetic Policy 2 (π_{s2})	**P.V.Loss**	0.11	0.09	0.08	0.06	0.05	0.06
	Best Hyperparam	$\epsilon = 0.03$	1500 first	6 bags	cover $n = 1$	cover $n = 1$	NA
	Running Time (s)	0.466	0.742	0.620	0.410	0.495	0.569
Synthetic Policy 3 (π_{s3})	**P.V.Loss**	0.12	0.11	0.07	0.06	0.05	0.07
	Best Hyperparam	$\epsilon = 0.01$	1500 first	8 bags	cover $n = 1$	cover $n = 1$	NA
	Running Time (s)	6.2	6.1	6.1	6.1	6.1	6.1

4.3.3.2 Manual and Synthetic Policy Datasets

Figures 14 and 15 show the results of various learning algorithms over complete and partial datasets for manually written policies as well as randomly generated ones. Interestingly, in most cases, one or more contextual bandit algorithms outperform the supervised learning algorithm. For all complete datasets, the online cover algorithm converges to the lowest validation loss. As we expect, algorithms achieve lower loss over complete datasets compared to partial ones as they have more data available in their training phase. In the same way, as is shown in Fig. 15, as datasets get larger, the final losses of algorithms over them become lower.

4.3.3.3 Comparison with Other Classification Methods

As we can see in Table 11, the Online Cover algorithm (with a cover set of size 2) performs the best among other RL algorithms for almost all datasets. To compare our proposed

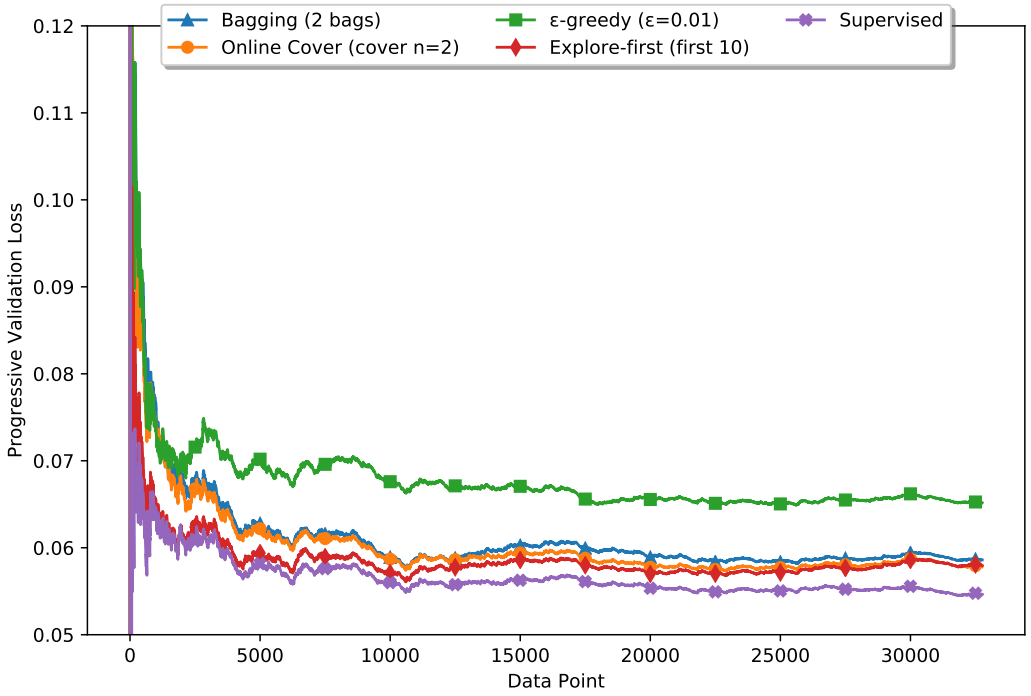
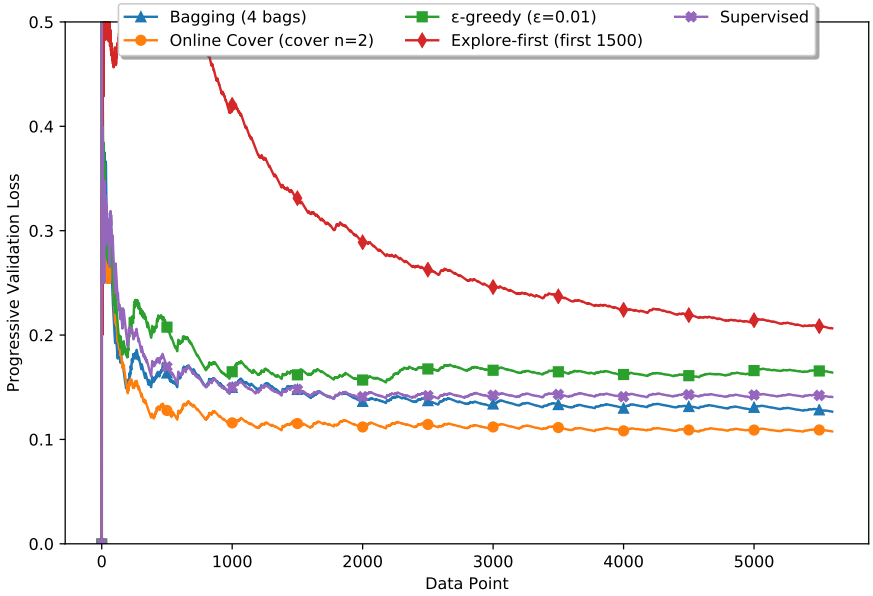
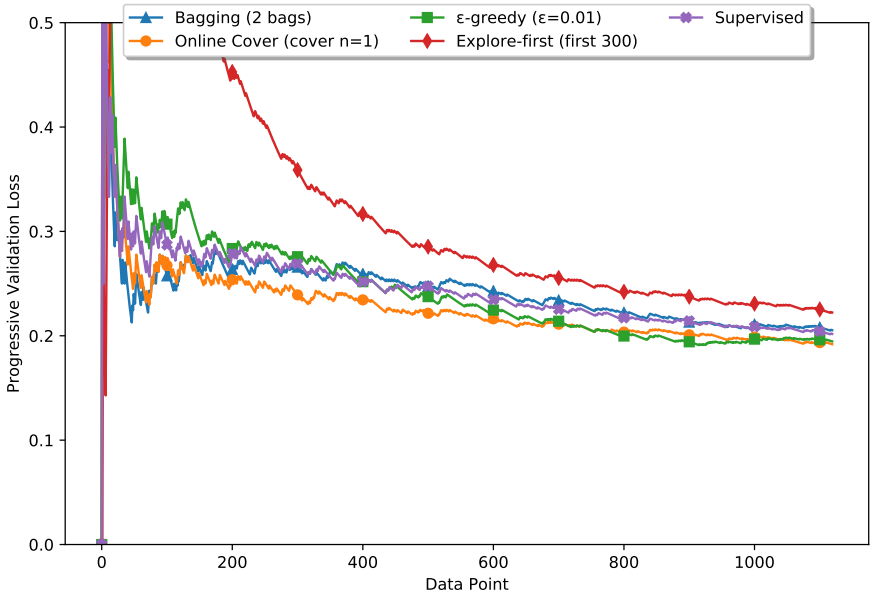


Figure 13: Progressive validation loss of various algorithms on Kaggle dataset [5]

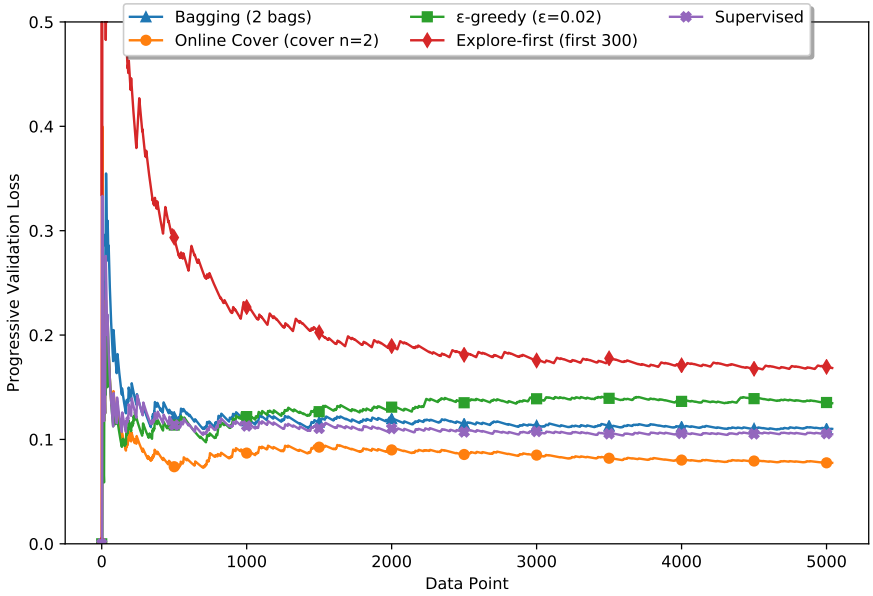


(a) π_{m1} Complete Dataset

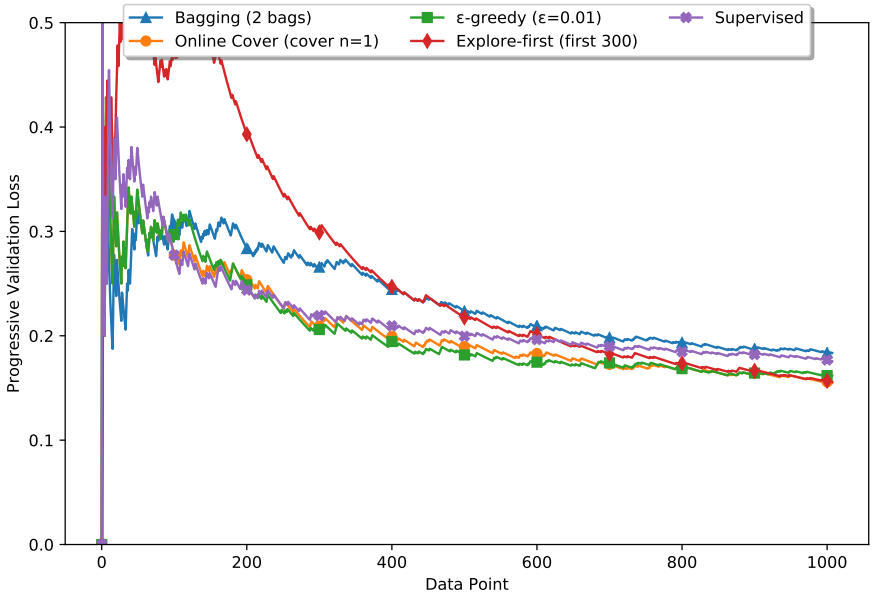


(b) π_{m1} Partial Dataset

Figure 14: P.V.Loss of various algorithm on manual policies' complete and partial datasets

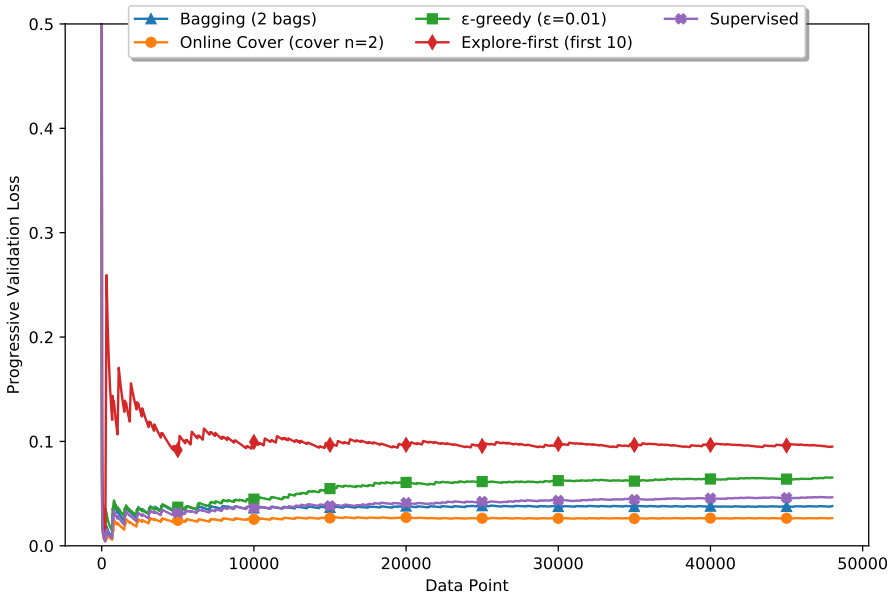


(c) π_{m2} Complete Dataset

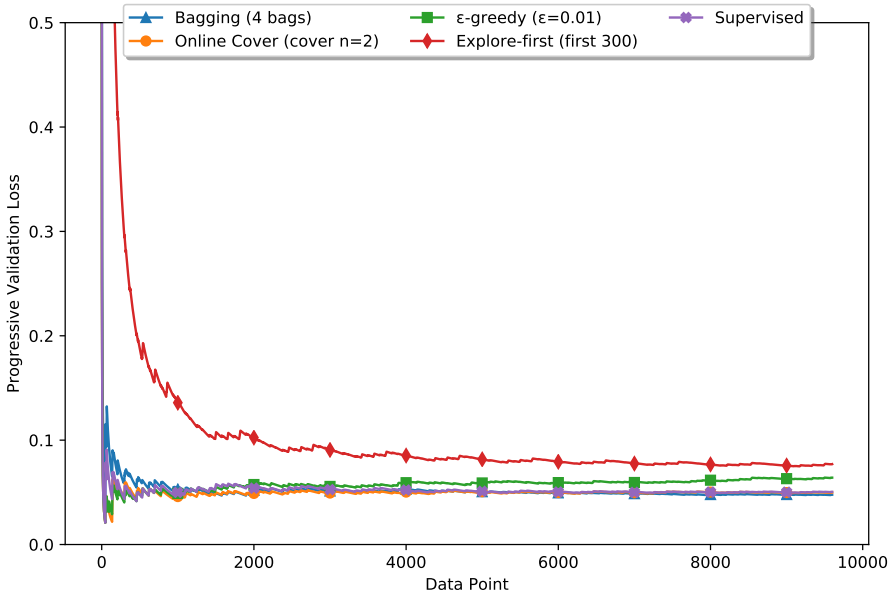


(d) π_{m2} Partial Dataset

Figure 14: P.V.Loss of various algorithm on manual policies' complete and partial datasets (Cont.)

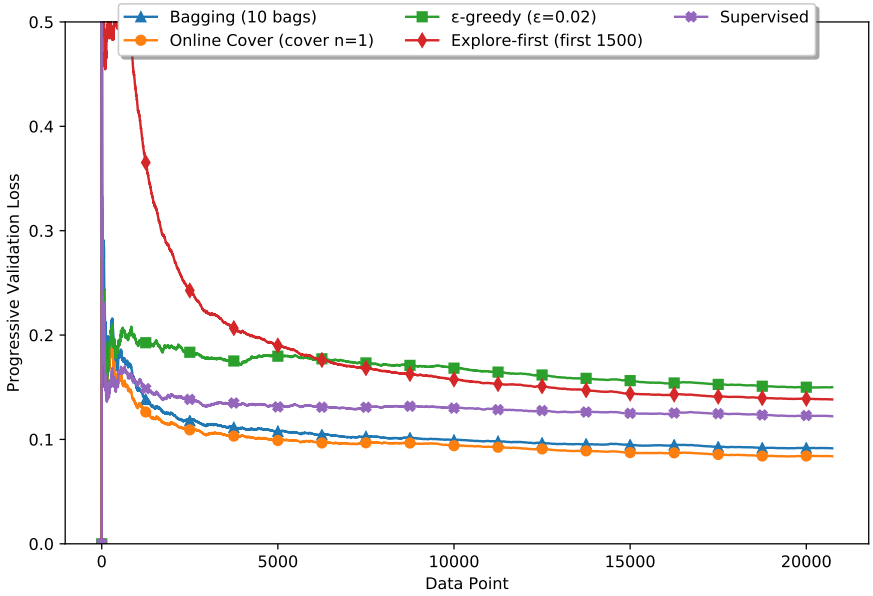


(e) π_{m3} Complete Dataset

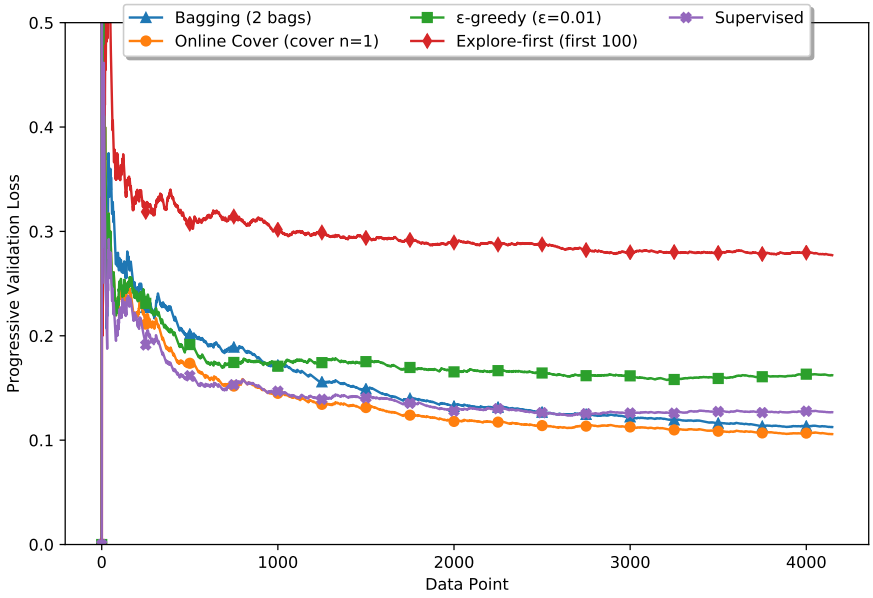


(f) π_{m3} Partial Dataset

Figure 14: P.V.Loss of various algorithm on manual policies' complete and partial datasets (Cont.)

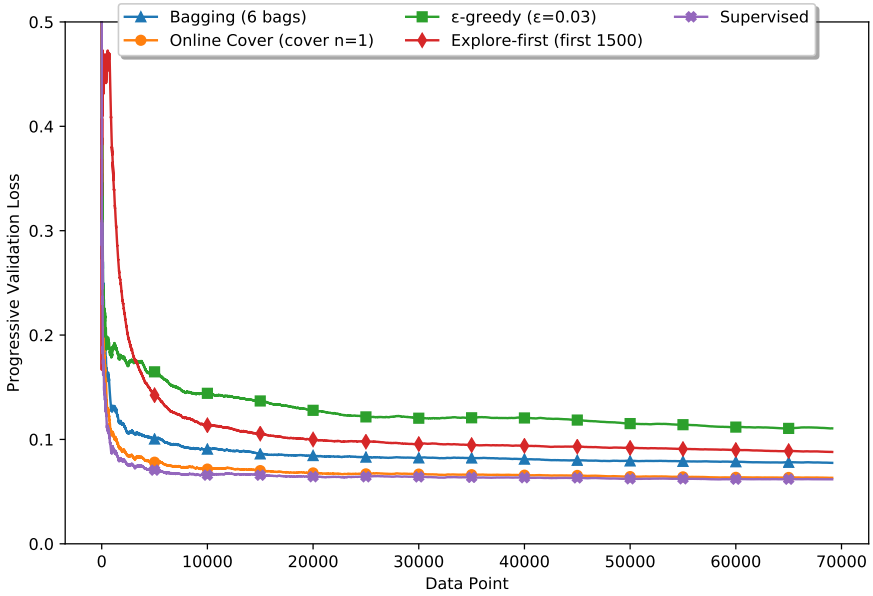


(a) π_{s1} Complete Dataset

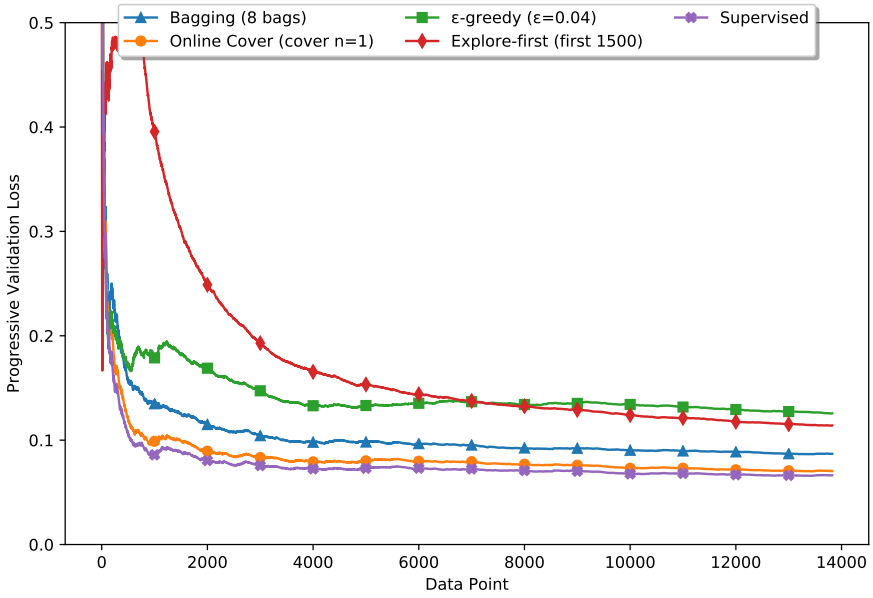


(b) π_{s1} Partial Dataset

Figure 15: P.V.Loss of various algorithm on synthetic policies' complete and partial datasets

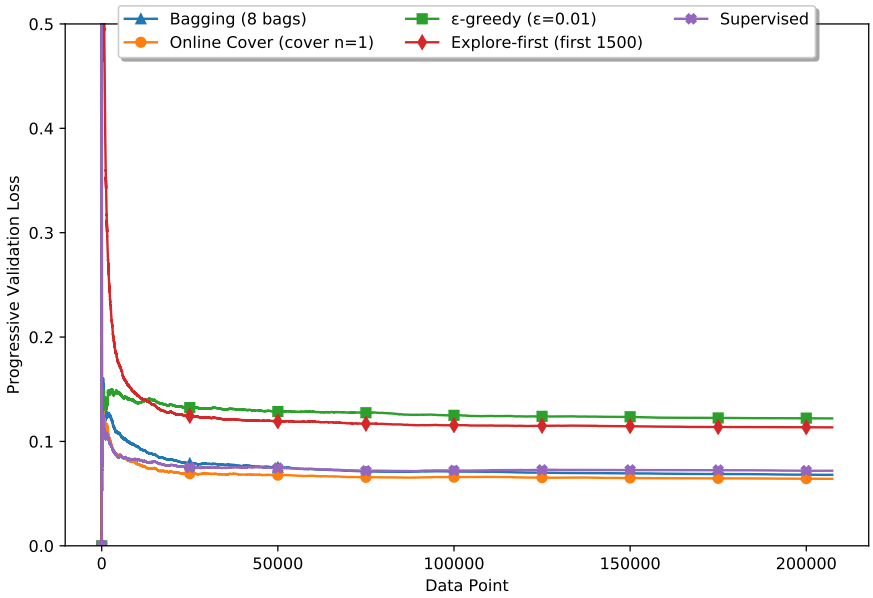


(c) π_{s2} Complete Dataset

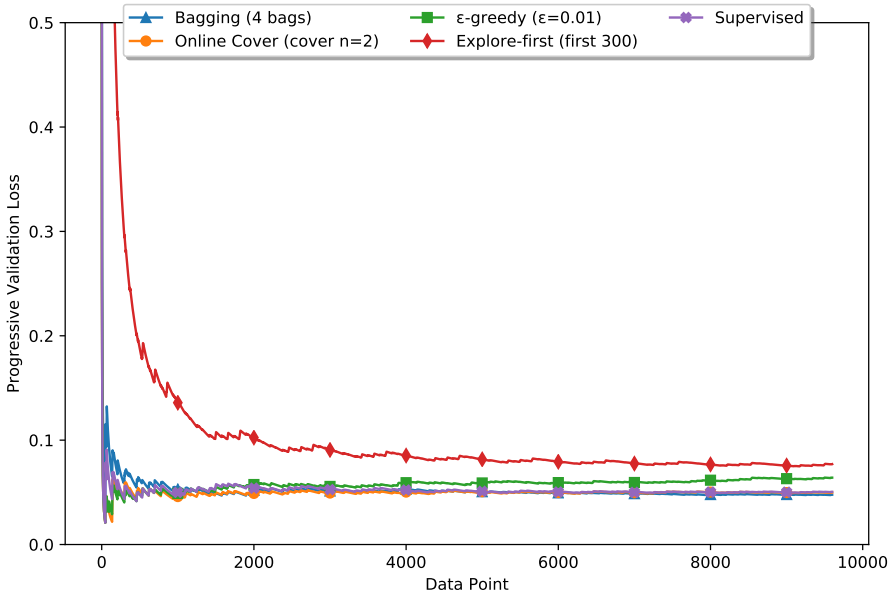


(d) π_{s2} Partial Dataset

Figure 15: P.V.Loss of various algorithm on synthetic policies' complete and partial datasets (Cont.)



(e) π_{s3} Complete Dataset



(f) π_{s3} Partial Dataset

Figure 15: P.V.Loss of various algorithm on synthetic policies' complete and partial datasets (Cont.)

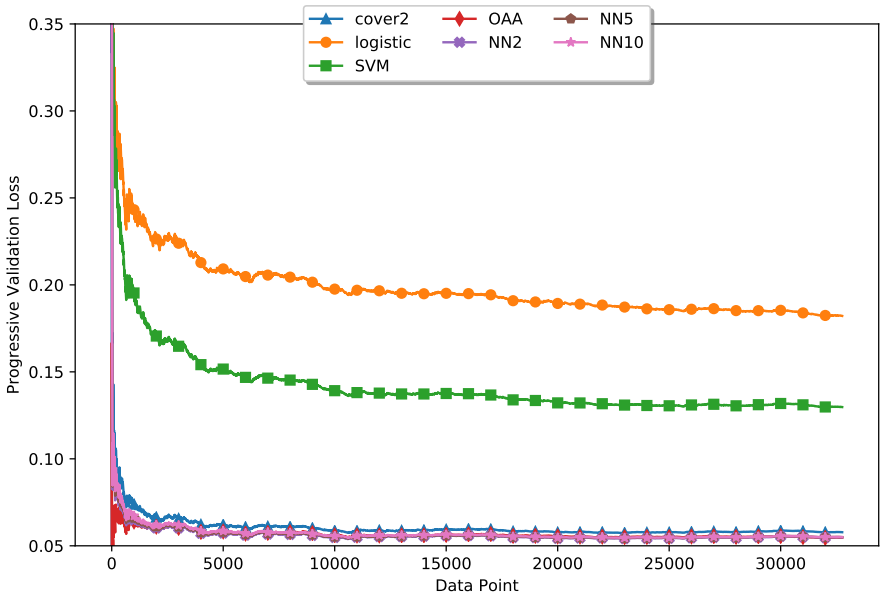
approach with other classification methods, we run a set of experiments in which we evaluate the performance of the Online Cover algorithm (with $n = 2$) and several classification algorithms including *logistic regression*, *support vector machine (SVM)*, *one against all (OAA)*, and various configurations of *neural network (NN)* algorithms. Fig 16 shows the P.V.Loss of these algorithms over various datasets. As we can see, different configurations of NN algorithms have the best performance over various datasets. The performance of our proposed approach (using Cover 2 algorithm) is very close to NN algorithms. As shown in the graphs, logistic regression and SVM don't perform well over all datasets which is understandable. Logistic regression has a linear decision surface and assumes a linear relationship between dependent variables and independent variables which does not hold in the problem of ABAC policy learning. And SVM algorithm does not perform well when the dataset is big, the dataset is noisy, or when target classes are overlapping. Hence, SVM is not a good choice for the given problem.

4.3.3.4 Policy Shift

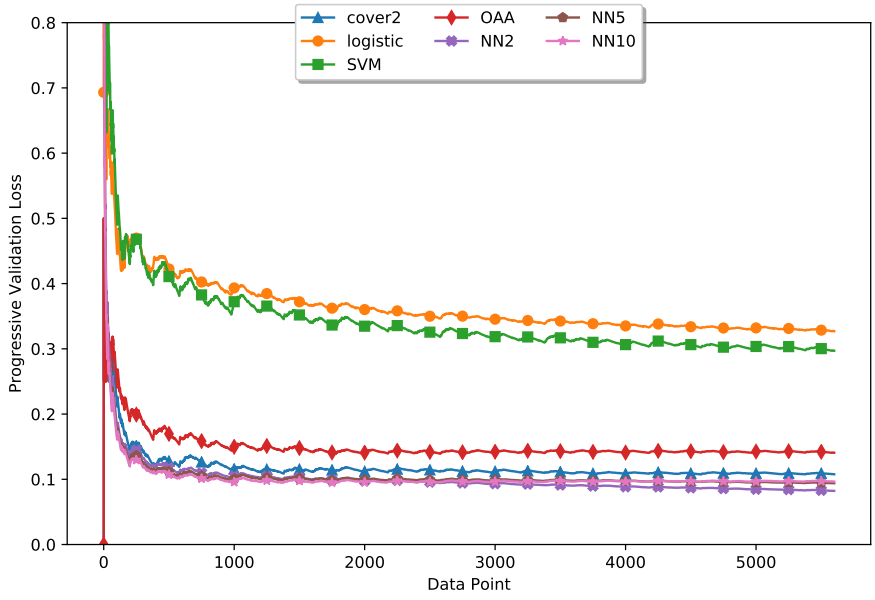
Next, we examine how various algorithms respond to a shift in the authorization policy. Fig 17 shows the P.V.Loss of these algorithms over a policy that has a shift after $t = 5600$. The authorization policy shifts from π_{m1} to π_{m2} after this timestamp. Note that all algorithms have a slight increase in P.V.Loss after the shift but they adapt to the change in the policy. For this specific simulation, Online Cover (with a cover set of size 2) adjusts quicker than other algorithms and converge to a better P.V.Loss value. Interestingly, the two contextual bandit algorithms (i.e. Online Cover and Bag) outperform the supervised learning algorithm in terms of adapting to the shift in the policy.

4.3.3.5 Policy Initialization Techniques

Fig 18 shows the results of various initialization techniques over manual policy π_{m3} dataset. As shown in the graph, the initialization with general rules resulted in the highest decline in the average loss and the initialization based on the default capability action resulted in the lowest decline in the average loss.

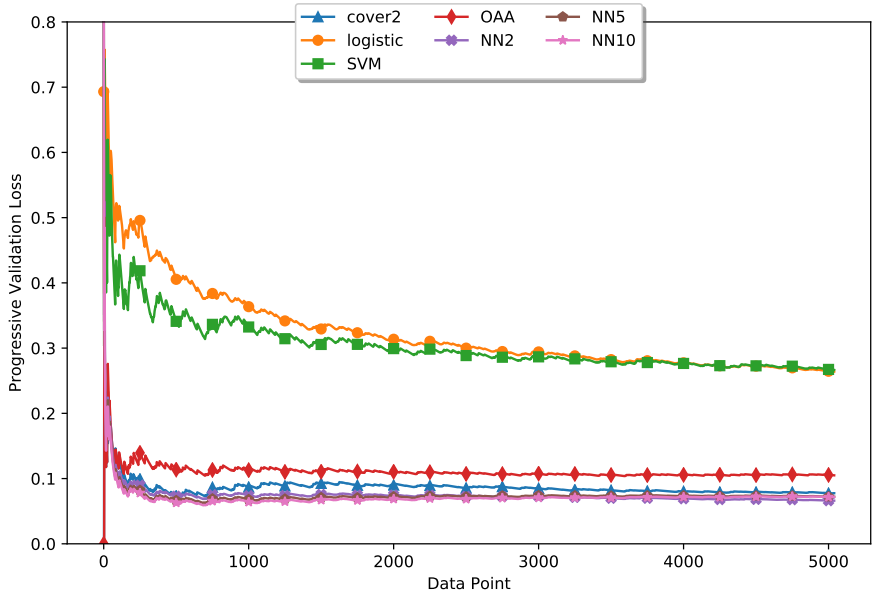


(a) Kaggle Dataset

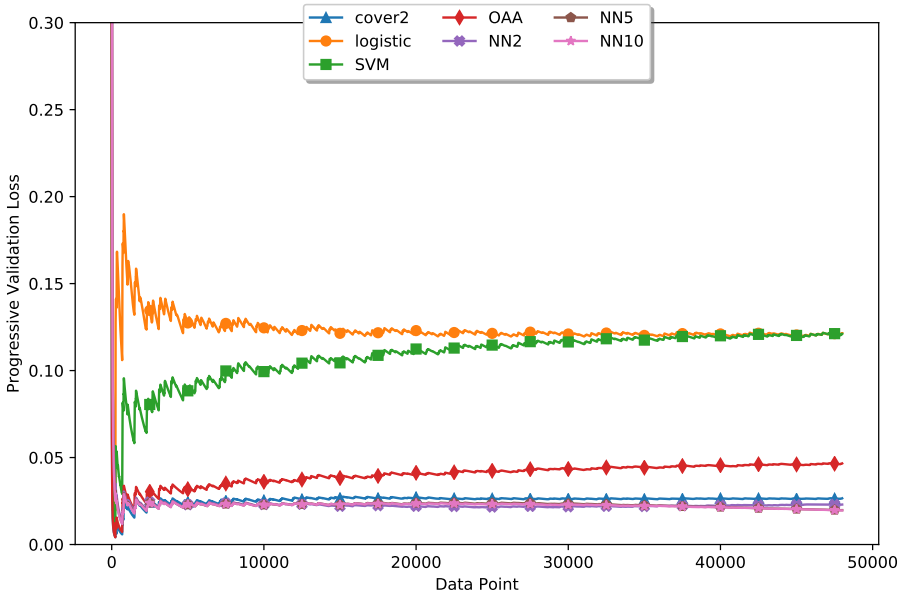


(b) π_{m1} Complete Dataset

Figure 16: Comparison of Online Cover algorithm with various supervised algorithms on different datasets



(c) π_{m2} Complete Dataset



(d) π_{m3} Complete Dataset

Figure 16: Comparison of Online Cover algorithm with various supervised algorithms on different datasets (Cont.)

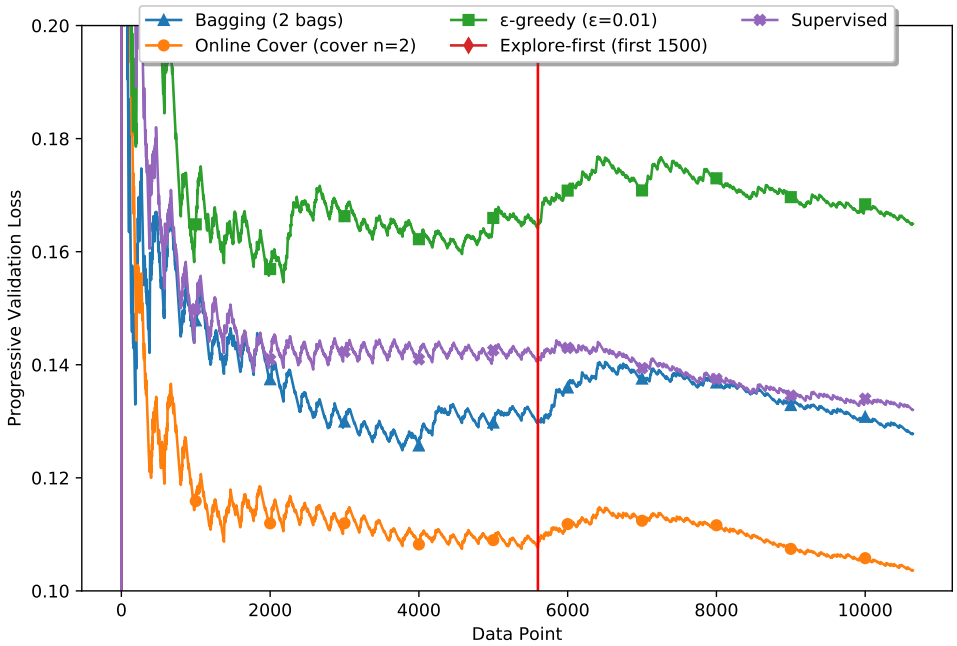


Figure 17: Adaptation of various algorithms to policy shift

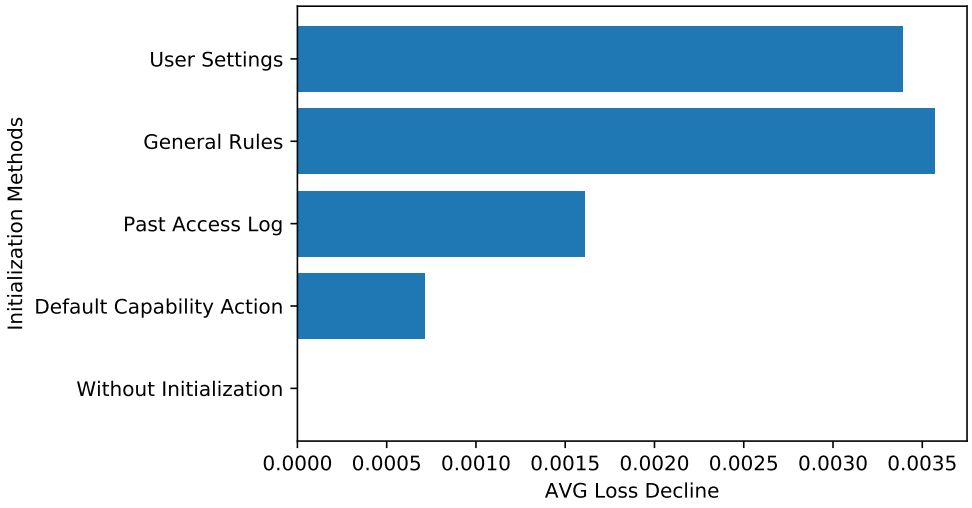


Figure 18: The effect of various initialization techniques in reducing average loss of Online Cover model on π_{m3} dataset

4.3.3.6 Policy Learning with Planning

Fig 19 shows the results of the Online Cover algorithm over manual policy π_{m3} with and without planning. As we can see in the figure, the learning algorithm with planning decreased the PVL by 25% for this dataset. We should note that Online Cover learned a model with the lowest PVL for this dataset and this is a significant reduction for such a model. Table 11 shows the results of the planning algorithm over various databases. As we can see from the results, the planning algorithm decreases the PVL for at least 10% for all cases that the planning was applied.

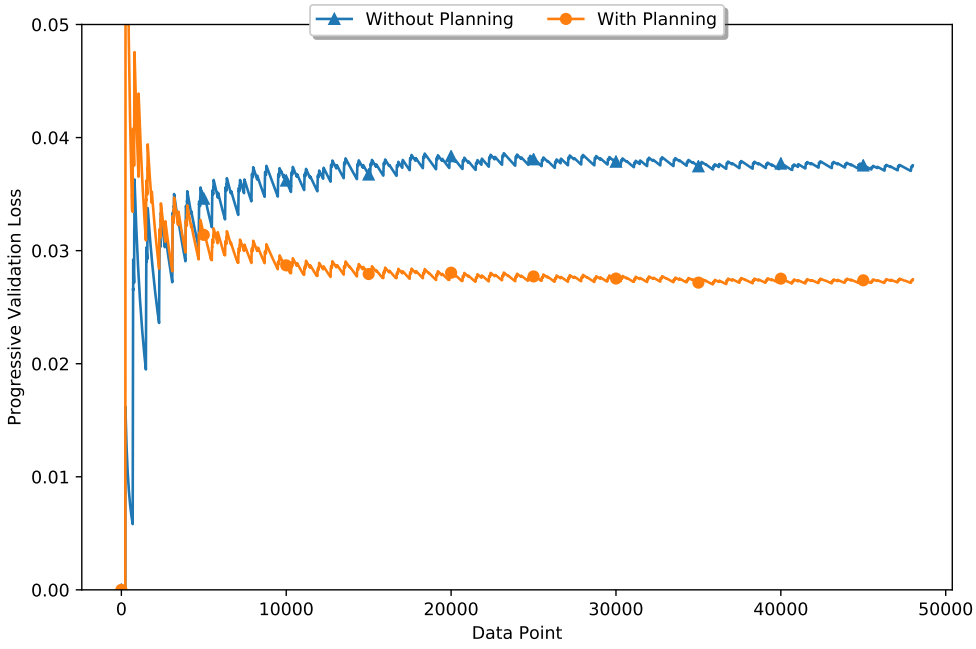


Figure 19: The effect of learning with planning on reducing average loss of Online Cover model on π_{m3} dataset

4.4 Discussion

While an adaptive ABAC policy learning through an RL model seems a promising approach for policy development and management in complex and evolving systems, there are some considerations that need to be reviewed carefully. In this section, we discuss such challenges.

4.4.1 Managing Incorrect Authorization Decisions

The learning process needs to go through multiple rounds of state-action-feedback sequences to achieve good performance. Hence, some incorrect authorization decisions during the learning phase is inevitable. To address the issue, the system could employ the following approaches to minimize the consequences of the wrong authorization decisions of the RL agent throughout the learning process:

- The system can employ a simpler AC model (e.g., RBAC or DAC) while the learning process is taking place.
- The system can continue using its legacy AC model while the learning process is taking place and before completely migrating to the newly learned ABAC model. This approach has two advantages: first, the decision of the legacy system can be used as feedback to the learning agent, and second, the migration can happen when the administrator feels confident about the decisions of the agent (i.e., the loss is less than a desirable threshold).
- For sensitive resources, the default action can be set to "deny" (except for an administrator) and the RL model can be initialized with these default decisions as suggested in the proposal.

4.4.2 Model Convergence

The learning process of the proposed model is assumed to have been converged when the loss of the RL model is less than a desired threshold set by the administrator of the system. However, in the case of dynamic systems, for any newly added attribute/attribute values or

an update in authorization policies, a learning process will continue. As such, the RL agent will always work in the background.

For example, as we can see in Figures 14 and 15, the more the agent receives feedback from the users, the lower the loss of the RL model would be. If we assume the desired loss of the model is set at 0.15 by the administrator of the system, then the Online Cover algorithm converges for all the datasets.

On the other hand, when we have an update in the authorization policy (as described in Section 4.3.3.4 and was shown in Figure 17), although the RL model was converged before the policy update, the loss rate would increase after such an update. Hence, the agent needs more feedback from the users to learn the new policy. Therefore, the loss rate decreases and the model converges again.

4.5 Conclusion

In this section, we take a reinforcement learning, more specifically, a contextual-bandit approach, to support adaptive ABAC policy learning. We have proposed a simple and reliable method for learning ABAC policy rules by incorporating the feedback of users on access decisions made by the authorization engine. We have focused on Home IoT as a running example throughout the section. We have proposed four different procedures for initializing the learning model and a planning approach based on attribute value hierarchies to accelerate the learning process. We have evaluated our proposed approach over real and synthetic data including both complete and sparse datasets. Our experiments show that the proposed learning model not only fits the dynamic nature of modern information systems but it also achieves performance comparable to the full information supervised learning methods in many scenarios and even outperforms them in several situations.

5.0 ABAC Policy Misconfiguration Detection and Resolution

Authorization methods are critical components of each system and are vital for ensuring the confidentiality and availability of its information. By the advent of emerging technologies (e.g. Online Social Networks (OSNs), Cloud computing, Internet of Things (IoT), etc.), a need for a reliable access control (AC) mechanism has become more salient than ever before. However, current information systems are still suffering from unintended information leakages through unauthorized access to their resources. Developing and managing access control policies are often error-prone as there is not an effective mechanism for analyzing such policies. Furthermore, in distributed/federated systems, the authorization policies can be designed and managed by multiple authorities, hence they may contain various anomalies including conflicts and redundancies. Besides, as modern systems evolve rapidly, their authorization policies go through several updates which if not done correctly could result in policy misconfigurations.

Attribute Based Access Control (ABAC) has shown to be a promising AC model for providing flexible and comprehensive authorization mechanisms in emerging complex systems. However, its flexibility comes with a cost; its development and management is much more complex than other AC models. The complexity of ABAC model makes the manual detection and resolution of policy misconfiguration a very challenging or even an impossible task.

In this chapter, we propose an ABAC policy misconfiguration detection and resolution framework that is based on a continuous analysis of ABAC policy rules as well as their corresponding access log. The rest of the chapter is organized as follows. In Section 5.1, we overview the ABAC model and its policy language that we use through out this chapter. In Section 5.2, we present the policy misconfiguration detection problem and enumerate various forms of policy misconfigurations. In Section 5.3, we present the proposed ABAC policy misconfiguration discovery and resolution framework. In Section 5.4, we present the evaluation of the proposed approach. Finally, Section 5.5 concludes the chapter.

5.1 ABAC Model

Throughout this chapter, U, O, and OP represent sets of users, objects, and operations in a system, and UA, OA, and EA correspond to sets of user attributes, object attributes, and environmental attributes, respectively. $E = U \cup O$ and $A = UA \cup OA \cup EA$ are the sets of all entities and all attributes in the system.

Definition 29. *(**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 30. *(**Attribute Function**). Given an attribute $a \in A$ and an entity $e \in E$, an* attribute function $Attr(a, e)$ *is a function that maps an entity to a specific value from the attribute range. Specifically, $Attr(a, e)$ returns the value of attribute a for entity e.*

Example 8. *$Attr(position, John)$ = faculty indicates that the value of attribute* position *for* John *is* faculty*.*

Definition 31. *(**Attribute Filter**). An* attribute filter *is defined as a set of tuples $F = \{\langle a, v\rangle |\ a \in A$ and $v \in V(a)\}$. Here $\langle a, v\rangle$ is an attribute filter tuple that indicates a has value v.*

Attribute filters are used to filter entities based on their attribute values.

Example 9. *Tuple $\langle department, admissions\rangle$ points to all entities in the system that have "*admissions*" as the value of their "*department*" attribute.*

Definition 32. *(**Attribute Filter Satisfaction**). An entity $e \in E$ satisfies an attribute filter F, denoted as $e \models F$, iff*

$$\forall \langle a_i, v_i\rangle \ \in F : Attr(a_i, e) = v_i$$

Example 10. *Suppose $A_u = \{dept, position, courses\}$. The set of tuples $F_u = \{\langle dept, CS\rangle, \langle position, grad\rangle\}$ denotes a user attribute filter. Here, the graduate students in the CS department satisfy F_u.*

Definition 33. *(**Access Request**). An* access request *q is a tuple $q = \langle u, o, op, ea\rangle$ where user $u \in U$ is the requester requesting to perform operation $op \in OP$ on object $o \in O$ while environmental attributes $ea \in EA$ holds.*

Definition 34. *(**Access Record/Access Log**). An* access record *is a tuple $l = \langle q, d\rangle$ containing the decision d made by the system for request q. An* Access Log *$\mathcal{L}$ is a set of such tuples.*

The decision d for an access request can be *permit* or *deny*. A tuple with *permit* as its decision means that user u is authorized to perform operation *op* over an object o under environmental attributes *ea*. an access record with *deny* decision means the user is not authorized to get such access.

Definition 35. *(**ABAC Rule**). An* ABAC rule *ρ is a tuple $\rho = \langle f_u, f_o, f_e, op, d\rangle$, where f_u, f_o, and f_e are user attribute filter, object attribute filter and environmental attribute filter, respectively, op is a corresponding operation, and d shows the decision of the ABAC rule for such combination of attributes and requested operation.*

Example 11. *Consider rule $\rho_1 = \langle\{\langle position, student\rangle, \langle dept_u, CS\rangle\}, \{\langle type, article\rangle, \langle dept_o, CS\rangle\}, \{\langle location, campus\rangle\}, read, permit\rangle$. It can be interpreted as "A student from CS department is permitted to read an article from CS department if he/she is on campus".*

Definition 36. *(**Rule Satisfaction**) An access request $q = \langle u, o, op_q, ea\rangle$ is said to satisfy a rule $\rho = \langle f_u, f_o, f_e, op_\rho, d\rangle$, denoted as $q \models \rho$, iff*

$$u \models f_u \wedge o \models f_o \wedge ea \models f_e \wedge op_q = op_\rho.$$

Definition 37. *(**Rule Coverage**). The* rule coverage *of ρ over access log $\mathcal{L}$ denoted as $[\![\rho]\!]_{\mathcal{L}}$ is a set of access right tuples which the rule ρ can be applied to, formally:*

$$[\![\rho]\!]_{\mathcal{L}} = \{\langle q, d\rangle \in \mathcal{L} \mid q = \langle u, o, op_q, ea\rangle \wedge q \models \rho\}.$$

Definition 38. *(**Overlapping Coverage**). The* overlapping coverage *of ρ over access log $\mathcal{L}$ denoted as $[\![\rho]\!]^{\mathcal{O}}_{\mathcal{L}}$ is a set of access right tuples in $[\![\rho]\!]_{\mathcal{L}}$ that are also part of the coverage of one or more other rules, formally:*

$$[\![\rho]\!]^{\mathcal{O}}_{\mathcal{L}} = \{\langle q, d\rangle \in [\![\rho]\!]_{\mathcal{L}} \mid \exists \rho' \wedge \langle q, d\rangle \in [\![\rho']\!]_{\mathcal{L}}\}.$$

Definition 39. *(**Disjoint Coverage**). The* disjoint coverage *of ρ over access log $\mathcal{L}$ denoted as $[\![\rho]\!]^{\mathcal{D}}_{\mathcal{L}}$ is a set of access right tuples in $[\![\rho]\!]_{\mathcal{L}}$ that are not part of the coverage of any other rules, formally:*

$$[\![\rho]\!]^{\mathcal{D}}_{\mathcal{L}} = \{\langle q, d\rangle \in [\![\rho]\!]_{\mathcal{L}} \mid \nexists \rho' \wedge \langle q, d\rangle \in [\![\rho']\!]_{\mathcal{L}}\}.$$

Definition 40. *(**ABAC Policy**). An ABAC policy π_{ABAC} is a tuple $\pi_{ABAC} = \langle E, A, OP, \mathcal{P}\rangle$ where E, ATT, OP, and $\mathcal{P}$ are sets of entities, attributes, operations, and ABAC rules in the system, respectively.*

Definition 41. *(**ABAC Policy Decision**). The decision of an ABAC policy π for an access request q is denoted as $d_\pi(q)$.*

$d_\pi(q)$ *is* permit *iff:*

$$(\exists \rho_1 \in \pi : d_{\rho_1} = permit \wedge q \models \rho_1) \wedge (\nexists \rho_2 \in \pi : d_{\rho_2} = deny \wedge q \models \rho_2)$$

$d_\pi(q)$ *is* deny *iff:*

$$(\exists \rho_1 \in \pi : d_{\rho_1} = deny \wedge q \models \rho_1) \wedge (\nexists \rho_2 \in \pi : d_{\rho_2} = permit \wedge q \models \rho_2)$$

$d_\pi(q)$ *is* unknown *iff:*

$$(\exists \rho_1 \in \pi : d_{\rho_1} = permit \wedge q \models \rho_1) \wedge (\exists \rho_2 \in \pi : d_{\rho_2} = deny \wedge q \models \rho_2)$$

The system needs to apply conflict resolution techniques to conclude the decision for unknown situations.

Definition 42. *(**Rule Cluster**). The* rule cluster *of ρ_i denoted as $\mathcal{C}(\rho_i)$ is a set of access right tuples in the rule coverage $[\![\rho_i]\!]_{\mathcal{L}}$ that has the same policy decision as the rule decision, formally:*

$$C(\rho_i) = \{\langle q, d\rangle \in [\![\rho_i]\!]_{\mathcal{L}} \mid d_\pi(q) = d_{\rho_i}\}.$$

Definition 43. *(**Rule Cluster Center**). The rule cluster center of ρ_i denoted as μ_i is the arithmetic mean of all the access right tuples belonging to the rule cluster, formally:*

$$\mu_i = \{\mu_i[1], \mu_i[2], ..., \mu_i[m]\} : \mu_i[j] = \frac{1}{n_j} \sum_{x \in \mathcal{C}(\rho_i)[j]} x, \;\; n_j = |\mathcal{C}(\rho_i)|.$$

Definition 44. *(**Rule Coverage Purity**). Rule coverage purity of ρ, denoted as $Purity(\rho)$, is a measure of the extent to which rule coverage is disjoint, formally:*

$$Purity(\rho) = \frac{|[\![\rho]\!]^{\mathcal{D}}_{\mathcal{L}}|}{|[\![\rho]\!]_{\mathcal{L}}|}$$

Definition 45. *(**Policy Purity**). Policy purity of π, denoted as $Purity(\pi)$, is a measure of the extent to which the coverage of the rules in the policy is disjoint, formally:*

$$Purity(\pi) = \frac{1}{n} \sum_{\rho_i \in \mathcal{P}_\pi} \frac{|[\![\rho_i]\!]^{\mathcal{D}}_{\mathcal{L}}|}{|[\![\rho_i]\!]_{\mathcal{L}}|}, \;\; n = |\mathcal{P}_\pi|$$

5.2 Policy Misconfiguration

In large distributed systems, conflicts between policies are inevitable. Manual resolution of such conflicts is a tedious task. In order to be able to resolve policy conflicts automatically, we first need to distinguish between different types of conflicts that may occur in policy configuration. Moffett and Sloman in [44] classify policy conflicts into two main categories: *Conflict of Modalities* and *Conflict of Goals*. *Conflict of Modality* points to two types of conflicts: *Positive-Negative Conflict* that happens when a subject is both permitted and denied for the same action on an object and *Conflict between Imperative and Authority* that occurs when a subject is required to initiate an action on an object and at the same time, he is forbidden to carry out such action. *Conflict of Goals* comprises multiple subcategories including *Conflict of Priorities* that occurs when two policies require the use of more resources than what is available, *Conflict of Duties* which is also known as a failure of the control principle of separation of duties, and *Conflict of Interests* which describes a situation where a subject is authorized to perform two different operations and carrying out both together is forbidden.

In ABAC policy domain, we are usually concerned about two types of anomalies in authorization policy specifications, *policy rules conflict* and *policy rules redundancy*. *Policy rules conflict* happens when two rules overlap (they both cover a common set of access requests) and they yield different decisions. Authorization systems often employ various

conflict resolution techniques (e.g., deny precedence, order precedence, recency precedence, etc.) to overcome such conflicts, however, these conflict could still result in safety issues (e.g., permitting unauthorized access) or availability issues (e.g., denying legitimate access). On the other hand, *policy rules redundancy* occurs when two rules overlap and they have same decision for every access request. Having redundant policy rules may increase the response time of authorization engine and worsen the policy evaluation performance [27].

Existing work in the literature focuses on discovery and resolution of design-time policy anomalies that can be found by analyzing the set of policy rules at the time of system set up. Such conflicts and redundancies that are identifiable at policy design time are formally defined as follows.

Definition 46. *(**Policy Rules Conflict**). Rule ρ_1 and rule ρ_2 have conflict (denoted as $\rho_1 \mathcal{X} \rho_2$) if they have common attribute filters and their operations are the same but they have different decisions. Formally,*

$$\rho_1 \mathcal{X} \rho_2 \iff$$
$$(f_{u1} \cap f_{u2} \neq \varnothing) \wedge (f_{o1} \cap f_{o2} \neq \varnothing) \wedge (f_{e1} \cap f_{e2} \neq \varnothing) \wedge op_1 = op_2 \wedge d1 \neq d2.$$

Definition 47. *(**Policy Rules Redundancy**). Rule ρ_1 and rule ρ_2 have redundancy (denoted as $\rho_1 \approx \rho_2$) if they have common attribute filters and their operations and decisions are the same. Formally,*

$$\rho_1 \approx \rho_2 \iff$$
$$(f_{u1} \cap f_{u2} \neq \varnothing) \wedge (f_{o1} \cap f_{o2} \neq \varnothing) \wedge (f_{e1} \cap f_{e2} \neq \varnothing) \wedge op_1 = op_2 \wedge d1 = d2.$$

Many studies have focused on discovering design-time policy misconfigurations and resolving them [27, 18]. However, not all misconfigurations are identifiable at design time and with only analyzing the policy rules. A more in-depth analysis with respect to both policy rules and access log tuples is required to prevent more complex policy anomalies. For example, assume that rule ρ_1 is part of the *University* access control policy. It allows users with *registrar* as their positions to read *rosters* when they're *on campus*.

$\rho_1 = \langle \{\langle position, registrar \rangle\}, \{\langle type, roster \rangle\}, \{\langle location, campus \rangle\}, read \rangle$

Now, assume rule ρ_2 is added to the system that allows anyone in *registration* department to read *rosters* when they're *on campus*.

$\rho_2 = \langle\{\langle department, registration\rangle\}, \{\langle type, roster\rangle\}, \{\langle location, campus\rangle\}, read\rangle$

While analyzing the rules attribute filters for identifying design-time anomalies, no misconfiguration will be found within these two rules. However, if we know that all the users in *registration* department have *registrar* as their position, then the second rule becomes redundant.

Such shortcomings motive us to employ anomaly detection techniques to detect misconfigurations in policy rules specifications.

5.3 ABAC Policy Misconfiguration Discovery and Resolution

In our proposed approach, we mainly focus on run-time anomalies that may occur in authorization specification, especially in highly dynamic systems. Such misconfigurations may happen due to various results, including:

- **Policy Update:** Any update in policy rules such as adding a new rule or changing policy rule attribute filters may result in a misconfiguration. For example, assume that a new rule is added to the policy which has a high shared coverage with another rule in the system. This could result in either *policy rule conflict* or *policy rule redundancy*.
- **Policy Attribute Update:** An update in the list of attributes of the system such as addition/deletion of an attribute from policy attributes may increase the shared coverage between rules and cause policy anomalies.
- **Entity Update:** In a highly dynamic system, updates in the statistical population of entities in a system are inevitable. Entities' attributes and attribute values may evolve continuously. Such updates may result in a correlation between attributes which may cause policy conflict/redundancy or an update in an attribute value that may make a policy rule ineffective. For example, during the pandemic, most users started to work from home, so the value of their *location* attribute shifted from *"On-Campus"* to *"Out-of-*

Campus". Such updates in users' attribute values, may lead to policy misconfigurations that need to be addressed by system administrators.

To be able to detect all of the aforementioned policy misconfigurations, we propose a multi-level anomaly detection framework as follows.

- **High-Level Policy Analysis** In this level, the focus is completely on the authorization policy and its features. The features such as number of rules, number of access requests, percentage of permitted requests, percentage of denied requests, percentage of total shared coverage, percentage of total disjoint coverage, etc., are recorded and analyzed periodically. Any anomaly in such records could be a sign of a misconfiguration in the authorization policy and needs to be examined further. The high-level policy analysis could reveal any misconfiguration that occurs due to a policy update, a policy attribute update, or an entity update.
- **Mid-Level Policy Rules Analysis** In this level, each policy rule goes through a separate line of analysis and its features will be used for anomaly detection purposes. The features that are being monitored regularly include the number of policy rule attribute filters, size of the rule coverage, percentage of rule shared coverage, percentage of rule disjoint coverage, the attribute values of rule center, the frequency of each attribute value in the rule coverage, etc. The analysis at this level could also reveal any misconfiguration due to policy updates, policy attribute updates, and entity updates.
- **Low-Level Access Records Analysis** Finally, we will run an anomaly detection algorithm on all access records corresponding to each rule. If a rule has a positive decision (i.e *permit*), we feed the anomaly detection algorithm with permitted access records and if a rule has a negative decision (i.e *deny*), we focus on the rejected access records. This level of analysis would locate any anomalies that were uncovered by the other two levels.

Figure 20 shows the proposed multi-level policy anomaly detection framework.

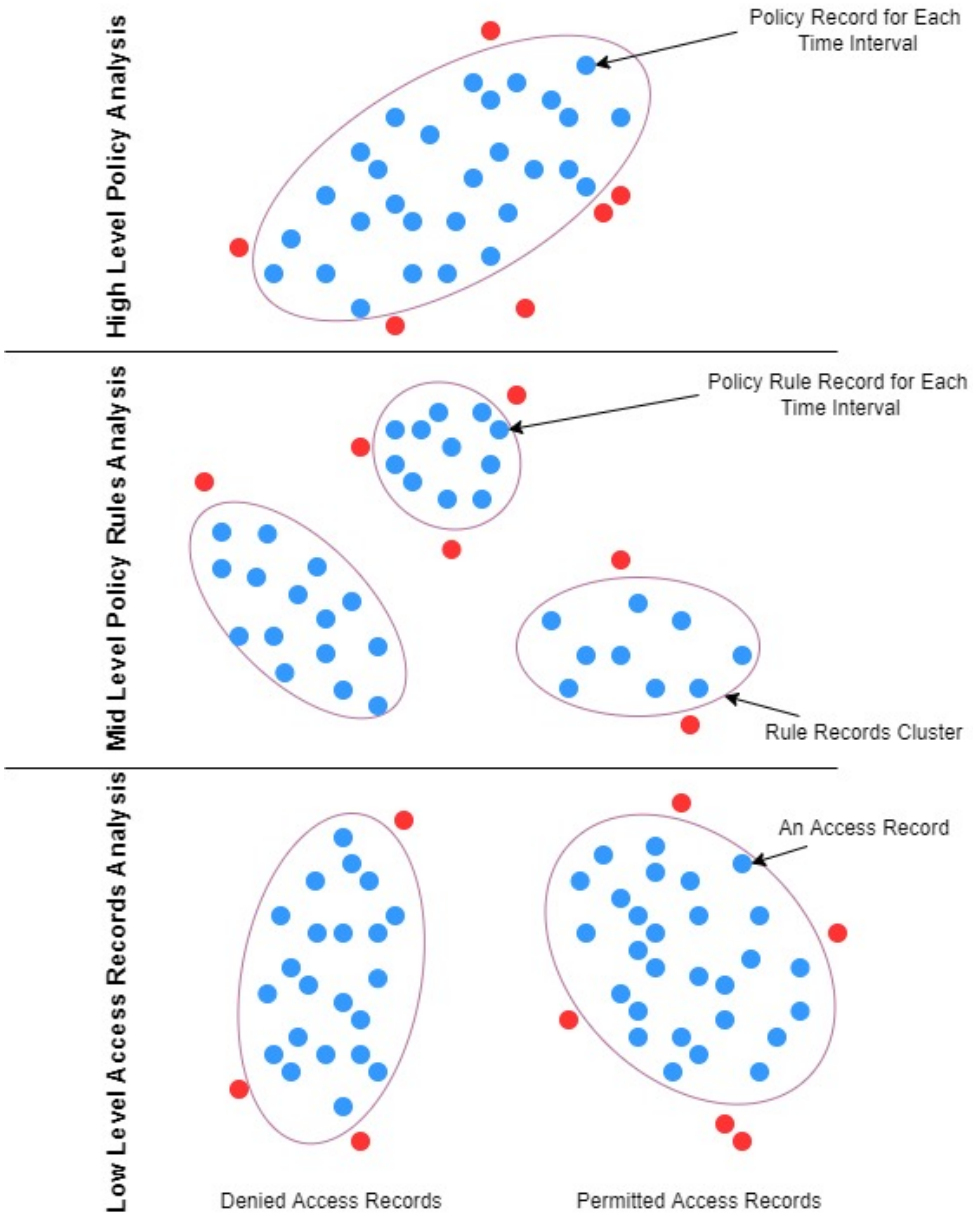


Figure 20: Overview of the proposed multi-level ABAC policy anomaly detection framework

5.3.1 ABAC Policy Misconfiguration Resolution

When an anomaly is detected in an ABAC policy by the proposed framework, the most recently updated policy rule is a potential candidate for causing the anomaly. However, as was discussed previously, there are various reasons that may result in misconfiguration in an ABAC policy, hence finding the candidate rules and tuning them is not straightforward. To be able to resolve a misconfiguration in an ABAC policy, first, we select the rules that correspond to the records that were found anomalous by the employed anomaly detection algorithm. Among the candidate rules, we focus on the rule that has the lowest *Rule Coverage Purity*. Such a rule is the candidate rule that needs to be tuned. For this purpose, we select another policy rule that has the highest overlapping coverage with the candidate rule. Our goal is to reduce the overlapping coverage between these two policy rules. Overlapping coverage often happens when there is a correlation between attribute values of two rules. Hence, the next step for resolving the misconfiguration is to find the correlated attribute values between the given rules. Finally, we add a negated *Attribute Filter* corresponding to the correlated attribute value to each rule to decrease the correlation and the overlapping coverage.

Example 12. *Assume the following two rules have high overlapping coverage.*

$\rho_1 = \langle\{\langle position, registrar\rangle\}, \{\langle type, roster\rangle\}, \{\langle location, campus\rangle\}, read\rangle$

$\rho_2 = \langle\{\langle department, registration\rangle\}, \{\langle type, roster\rangle\}, \{\langle location, campus\rangle\}, read\rangle$

Assume the overlapping coverage is the result of a high correlation between two attribute values, $\langle position, registrar\rangle$ *and* $\langle department, registration\rangle$*. To resolve the issue, we add the negation of each attribute filter to the other rule. Hence, the tuned rules will be as follows:*

$\rho'_1 = \langle\{\langle position, registrar\rangle, \langle \boldsymbol{department}, \boldsymbol{!registration}\rangle\}, \{\langle type, roster\rangle\}, \{\langle location, campus\rangle\}, read\rangle$

$\rho'_2 = \langle\{\langle department, registration\rangle, \}, \{\langle type, roster\rangle\}, \{\langle location, campus\rangle\}, read\rangle$

Algorithm 6 shows the details of the ABAC policy misconfiguration resolution process.

Algorithm 6: ABAC Policy Misconfiguration Resolution Algorithm

Input: $\mathcal{L}$, π_{ABAC}

Output: π_{ABAC}

procedure *Misconfiguration Resolution*

$anomalous\_records = anomaly\_detection\_algorithm(\mathcal{L},\pi_{ABAC})$

$\mathcal{R} = get\_candidate\_rules(anomalous\_records)$;

$\rho_1 = get\_lowest\_rule\_coverage\_purity(\mathcal{R}, \mathcal{L},\pi_{ABAC})$;

$\rho_2 = get\_highest\_overlapping\_coverage(\rho_1, \mathcal{L},\pi_{ABAC})$;

$\langle a_1, v_1\rangle, \langle a_2, v_2\rangle = find\_correlated\_attribute\_values(\rho_1, \rho_2, \mathcal{L})$;

$\rho_1 = \rho_1 \cup \langle a_2, !v_2\rangle$;

$\rho_2 = \rho_2 \cup \langle a_1, !v_1\rangle$;

return π_{ABAC}

5.4 Evaluation

In order to evaluate the performance of the proposed approach in terms of detecting and resolving ABAC policy anomalies, we set up an extensive experimental study. In this section, we describe the datasets, the anomaly detection algorithms, and the evaluation metrics.

5.4.1 Datasets

Since it is hard to get a set of the real-world access log, as they are often considered to be highly confidential by organizations, we generated synthetic datasets to check the scalability and performance of the proposed framework. The synthesized datasets are generated based on a set of manually generated policies that are used in Chapter 4. Appendix B shows the details of the manual policies used in the policy misconfiguration detection experiments. For each manual policy, we generate a complete access log (that includes every possible combination of attribute values) and in each time interval, we select a random set of access requests from the complete access log.

Table 12 shows the details of the access log datasets. In this table, $|\mathcal{P}|$ shows the number

Table 12: Details of Datasets

π_{ABAC}	$\lvert\mathcal{P}\rvert$	$\lvert A\rvert$	$\lvert V\rvert$	$\lvert\mathcal{L}\rvert$
Manual Policy 1 (π_{m1})	11	5	30	5K
Manual Policy 2 (π_{m2})	11	5	29	5K
Manual Policy 3 (π_{m3})	38	5	44	15K

of ABAC rules in the policy, $|A|$ and $|V| = \sum_{a \in A} |V(a)|$ show the number of attributes and attribute values, respectively, and $|\mathcal{L}|$ shows the size of the access log in each time interval.

5.4.2 Anomaly Detection Algorithms

We empirically evaluated three anomaly detection algorithms for our proposed approach. The algorithms are as follows:

- **Isolation Forest:** Isolation Forest is an unsupervised anomaly detection algorithm that employs the *random forest* algorithm (decision trees) for discovering outliers in the dataset [39]. The algorithm tries to divide the records in a way that each data point gets isolated from others. Usually, the anomalies are far away from other records, hence, it's easier to isolate them compared to normal data points. The Isolation Forest anomaly detection algorithm generates a ranking list that shows the degree of the anomaly.
- **Local Outlier Factor:** The *Local Outlier Factor* is an algorithm that calculates the density of data points and uses it as a metric for detecting anomalies [12]. The metric is called *anomaly score* and measures how isolated a record is with respect to its surrounding neighborhood.
- **One-Class SVM:** The *Support Vector Machine* (SVM) was originally proposed as a classification algorithm that predicts a hyperplane to separate two or more classes of data points. For *one-class SVM* where we have one class of records, the goal is to find a hyperplane that separates the cluster of data points from the anomalies [55].

Table 13: Performance of various anomaly detection algorithms over different policies (High Level Policy Analysis)

Metrics		**Databases**		
		π_{m_1}	π_{m_2}	π_{m_3}
Accuracy	**Isolation Forest**	88.6	90.5	89.6
	One Class SVM	74.2	98	86.8
	Local Outlier Factor	97.9	98.4	95.5
Precision	**Isolation Forest**	53.9	52.5	56.2
	One Class SVM	51.7	58.3	54.2
	Local Outlier Factor	62.9	61.8	49.2
Recall	**Isolation Forest**	94.2	95.2	94.7
	One Class SVM	84	87.5	85.2
	Local Outlier Factor	81.8	99.2	48.5
F1 Score	**Isolation Forest**	54.3	52.2	58.3
	One Class SVM	45.8	63.2	54.3
	Local Outlier Factor	68.2	68.7	48.9

5.4.3 Experimental Results

In this section, we compare various anomaly detection algorithms for detecting policy anomalies over different datasets. To have a baseline for comparing the performance of various anomaly detection algorithms, we label the records corresponding to the rules with the lowest *Rule Coverage Purity* as anomalous. Table 13 reports the performance of different anomaly detection algorithms over each dataset for the *high-level policy analysis*. As we can see from the results, the *Local Outlier Factor* has the highest performance among the three anomaly detection algorithms. Figure 21 shows the increase in *Policy Purity* of each dataset after applying the proposed anomaly detection and resolution algorithms. As we can see, the third policy has the highest increase in the *Policy Purity*, however, the proposed anomaly resolution algorithm resulted in at least %25 increase in *Policy Purity* for all datasets. Table 14 reports the performance of different anomaly detection algorithms over the rules with the highest shared coverage in Policy π_{m_1} in the *mid-level policy analysis*. Here again, the *Local Outlier Factor* has the highest performance among the three anomaly detection algorithms.

Table 14: Performance of various anomaly detection algorithms over different rules of policy π_{m_1} (Mid Level Policy Analysis)

Metrics		**Databases**			
		ρ_2	ρ_3	ρ_5	ρ_6
Accuracy	**Isolation Forest**	83	84	84.8	80
	One Class SVM	85.6	86.5	94.3	98.7
	Local Outlier Factor	99.5	99.3	99.3	99.6
Precision	**Isolation Forest**	51.4	51.5	51.6	51.2
	One Class SVM	51.6	51.8	54	64
	Local Outlier Factor	73	83.2	81.7	78.9
Recall	**Isolation Forest**	91.5	92	92.4	90
	One Class SVM	92.8	93.2	97.1	99.4
	Local Outlier Factor	82.3	83.2	81.7	78.9
F1 Score	**Isolation Forest**	48.1	48.5	48.9	46.8
	One Class SVM	49.3	49.7	56	71.5
	Local Outlier Factor	76.8	74.5	72.6	78

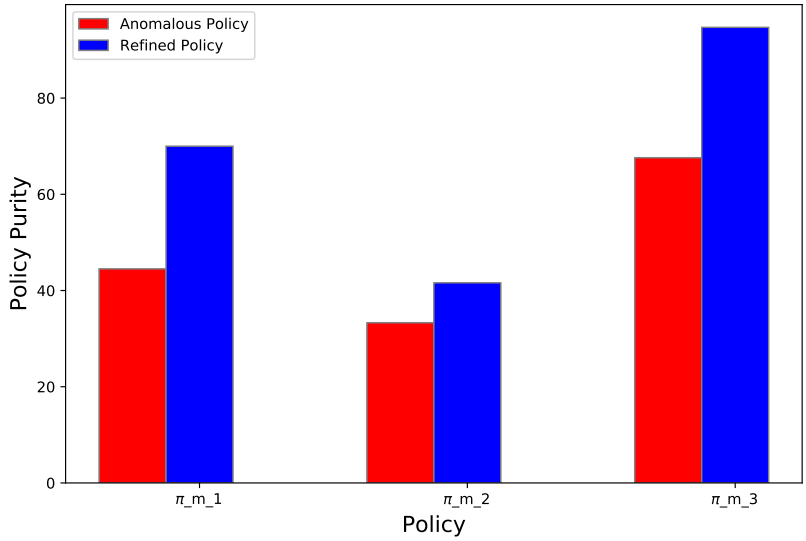


Figure 21: Policy Purity Increment After Applying the Proposed Anomaly Detection and Resolution Algorithms

5.5 Conclusion

In this section, we have proposed an anomaly detection based approach to finding ABAC policy misconfigurations and resolving them. The proposed approach is capable of detecting misconfigurations on different levels including *policy level*, *policy rule level*, and *access records level*. Furthermore, our proposed approach is capable of refining policies with the goal of resolving anomalies in authorization policy specifications and increasing policy quality.

6.0 Conclusions and Discussions

As part of this dissertation, we have presented three complementary frameworks that aim at partially/completely automating the ABAC policy development and maintenance.

We have presented a clustering-based approach to automatically extract ABAC policy rules from a given set of access logs. Our goal is to extract policy rules with both positive and negative attribute filters while minimizing the policy complexity. To achieve such an objective, we first propose to cluster the given access log based on the values of attributes of entities of the system. In our proposed framework, each cluster corresponds to one ABAC policy rule. Hence, we need to extract policy rule items for each cluster. For this purpose, we have defined *Effective Attribute Filter* and *Effective Relation* which are used to identify the policy rule items corresponding to the access tuples in each cluster. The extracted policy rules may need further processing to gain higher quality. We propose *Rule Pruning* and *Policy Refinement* algorithms to polish the extracted rules. The Rule Pruning algorithm helps with removing duplicate rules and the goal of the Policy Refinement algorithm is to identify too relaxed or too restricted policy rules and fix them. Furthermore, to be able to compare a set of extracted policies, we have proposed a *Policy Quality Metric* which observes the size of the extracted policy and the relative F-score of the extracted policy considering the given access log. The extracted policy with lower complexity (smaller size) and higher F-score (higher correctness) would have higher policy quality.

Additionally, we have presented an adaptive ABAC policy development and management framework in which we utilized Reinforcement Learning (RL) and, more specifically, contextual bandit, to establish a mapping between access requests and the appropriate authorization decisions for such requests. The authorization engine (AE) is considered as an agent in our proposed framework and its authorization decision (permit or deny) is an action according to the state of the system. Attributes of entities involved in an access request as well as the contextual factors form the state of the system. The AE learns authorization policies by interacting with the environment without having explicit knowledge of the original access control policy rules. We have proposed four methods for initializing the learning

model and a planning approach based on hierarchies over attribute values to accelerate the learning process.

Our third proposed framework is designed to detect ABAC policy misconfiguration based on a continues analysis of ABAC policy rules as well as their corresponding access log. We focused on detecting two types of run-time anomalies in ABAC policies: conflict between policy rules and redundant policy rules. We introduced run-time rule coverage (that shows the access requests covered by a policy rule in a period of time) as well as *overlapping coverage* and *disjoint coverage*. Such definitions are used for detecting overlapping rules and anomalous configurations. We have presented a multi-level anomaly detection framework that detects misconfigurations in ABAC policy rules and resolves them based on a proposed set of policy refinement techniques.

6.1 Limitations

There are some limitations with regards to the research conducted in this dissertation which we discuss below.

First, in this domain access to real datasets are very limited as access control records are sensitive and organizations are not willing to make them publicly available. Generating manual policies that are realistic and non-trivial is also challenging. These policies are mostly simple and small in size. On the other hand, synthetic ABAC policies can be generated in all sizes and with varying structural characteristics. To check the feasibility of our proposed approaches in a more complex setting, we used a set of randomly generated policies and showed the performance of our proposed approaches over them. However, we should note that the performance of our proposed methods on random policies might not be representative of their performance in real scenarios and over real policies.

Secondly, in our proposed approaches, the learned policy needs to go through multiple rounds of refinement to achieve good performance. Hence, some incorrect authorization decisions during the learning phase is inevitable. As a result, the proposed approaches may not be applicable/feasible in highly sensitive environments where an incorrect authorization

decision has a severe impact and is intolerable. In Section 4.4.1, we have suggested a few techniques to decrease the impact of incorrect decisions during the learning process.

In addition, in the first part of the framework, the proposed approach is based on an unsupervised clustering algorithm. Since finding the proper number of clusters is a challenge related to clustering algorithms, our approach is affected by this issue. The same issue will also be valid in finding the best thresholds to extract effective attributes and relations. Besides, as the proposed algorithm is based on tuning multiple parameters, it is possible that it gets stuck in minimum optima, however, by trying more randomization in cluster initialization and a wider range of parameters, we can get one that is closer to global optima.

Finally, in the second part of the framework, where we employed RL for learning the ABAC policy, the agent is dependent on the feedback it receives from the users of the system. While this feedback is limited to incorrect decisions made by the agent, it still needs some user intervention. It's important to develop a user-friendly solution to minimize the burden on users for giving feedback on incorrect decisions.

6.2 Future Research

There are several areas for future research in the domain of automatic ABAC policy design and management. While we have developed multiple frameworks based on different techniques for automating ABAC policy learning, there are still several research directions to explore for future work. One direction is to apply other learning algorithms for mining ABAC policies from access logs. It might be interesting to study more complex algorithms/techniques such as *Neural Network*, *Deep Learning*, and other *Evolutionary algorithms* for solving such problems.

Most of the experiments in this dissertation were done over synthesized datasets. As a potential future research direction and to be able to check the feasibility of various policy learning frameworks and compare them, it's necessary to conduct extensive experiments based on real-world datasets.

In addition, in this dissertation, the synthesized datasets are generated randomly without

considering the distribution of the real data. With respect to this, future work includes extracting the distributions of real data for various environments and synthesizing datasets based on such distributions.

In our proposed frameworks, we assumed all involved parties are trusted and will not deviate from the defined protocols. However, in real-world scenarios, such an assumption may not hold. Hence, it's important to study the presence of an adversarial attacker, who may try to deceive the learning algorithms and bypass security controls.

With respect to policy anomaly detection, future work includes extending the policy misconfiguration framework with information visualization techniques. Such visualization will provide an intuitive cognitive sense for anomalies in the ABAC policy and facilitate the policy management task.

Appendix A Chapter 3 Sample Policies

In the following, we list the policy rules we used in our experiments in Chapter 3. UniversityP, HealthcareP, and ProjectManagementP only contain positive attribute filters and relations while UniversityPN, HealthcarePN, and ProjectManagementPN include both positive and negative attribute filters and relations.

Table 15: Manual policies with only positive filters

Policy	Rules
UniversityP	$\langle\{\langle type,\ gradebook\rangle\}, \{\langle crsTaken,\ crs\rangle\},\ readScore\rangle$ $\langle\{\langle type,\ gradebook\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ addScore\rangle$ $\langle\{\langle position,\ faculty\rangle,\ \langle type,\ gradebook\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ changeScore\rangle$ $\langle\{\langle department,\ registrar\rangle,\ \langle type,\ roster\rangle\},\ read\rangle$ $\langle\{\langle position,\ faculty\rangle,\ \langle type,\ roster\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ read\rangle$ $\langle\{\langle type,\ transcript\rangle,\ \{\langle uid,\ student\rangle\},\ read\rangle$ $\langle\{\langle isChair,\ true\rangle,\ \langle type,\ transcript\rangle\},\ \{\langle udepartment,\ rdepartment\rangle\},\ read\rangle$ $\langle\{\langle department,\ registrar\rangle,\ \langle type,\ transcript\rangle\},\ read\rangle$ $\langle\{\langle type,\ application\rangle\},\ \{\langle uid,\ student\rangle\},\ checkStatus\rangle$ $\langle\{\langle department,\ admissions\rangle,\ \langle type,\ application\rangle\},\ setStatus\rangle$
HealthcareP	$\langle\{\langle position, nurse\rangle, \langle type,\ HR\rangle\}, \{\langle uward,\ rward\rangle\},\ addItem\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle teams,\ treatingTeam\rangle\},\ addItem\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle uid,\ patient\rangle\},\ addNote\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle agentFor,\ patient\rangle\},\ addNote\rangle$ $\langle\{\langle type,\ HRitem\rangle\}, \{\langle uid,\ author\rangle\},\ read\rangle$ $\langle\{\langle position, nurse\rangle, \langle type,\ HRitem\rangle, \langle topic,\ nursing\rangle\}, \{\langle uward,\ rward\rangle\},\ read\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle specialty,\ topic\rangle\}, \{\langle teams,\ treatingTeam\rangle\},\ read\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle topic,\ note\rangle\}, \{\langle uid,\ patient\rangle\},\ read\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle topic,\ note\rangle\}, \{\langle agentFor,\ patient\rangle\},\ read\rangle$
ProjectManagementP	$\langle\{\langle adminRole, manager\rangle, \langle type,\ budget\rangle\}, \{\langle udepartment, odepartment\rangle\},\ approve\rangle$ $\langle\{\langle\rangle, \langle type,\ schedule\rangle\}, \{\langle projectsLed, project\rangle\},\ read\rangle$ $\langle\{\langle\rangle, \langle type,\ schedule\rangle\}, \{\langle projects, project\rangle\},\ read\rangle$ $\langle\{\langle\rangle, \langle type,\ task\rangle\}, \{\langle task, rid\rangle\},\ setStatus\rangle$ $\langle\{\langle\rangle, \langle type, task\rangle, \langle proprietary, false\rangle\}, \{\langle projects, project\rangle, \langle oexpertise, rexpertise\rangle\}, read\rangle$ $\langle\{\langle isEmployee, True\rangle, \langle type, task\rangle, \langle proprietary, false\rangle\}, \{\langle projects, project\rangle,$ $\langle oexpertise, rexpertise\rangle\}, request\rangle$ $\langle\{\langle adminRole, auditor\rangle, \langle type,\ budget\rangle\}, \{\langle projects, project\rangle\},\ read\rangle$ $\langle\{\langle adminRole, accountant\rangle, \langle type,\ budget\rangle\}, \{\langle projects, project\rangle\},\ write\rangle$ $\langle\{\langle adminRole, accountant\rangle, \langle type,\ task\rangle\}, \{\langle projects, project\rangle\},\ setCost\rangle$ $\langle\{\langle adminRole, planner\rangle, \langle type,\ schedule\rangle\}, \{\langle projects, project\rangle\},\ write\rangle$ $\langle\{\langle adminRole, planner\rangle, \langle type,\ task\rangle\}, \{\langle projects, project\rangle\},\ setSchedule\rangle$

Table 16: Manual policies with both positive and negative filters

Policy	Rules
UniversityPN	$\langle\{\langle type,\ gradebook\rangle\}, \{\langle crsTaken,\ crs\rangle\},\ readScore\rangle$ $\langle\{\langle type,\ gradebook\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ addScore\rangle$ $\langle\{\langle position,\ faculty\rangle,\ \langle type,\ gradebook\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ changeScore\rangle$ $\langle\{\langle department,\ registrar\rangle,\ \langle type,\ roster\rangle\},\ read\rangle$ $\langle\{\langle position,\ faculty\rangle,\ \langle type,\ roster\rangle\},\ \{\langle crsTaught,\ crs\rangle\},\ read\rangle$ $\langle\{\langle type,\ transcript\rangle,\ \{\langle uid,\ student\rangle\},\ !write\rangle$ $\langle\{\langle isChair,\ true\rangle,\ \langle type,\ transcript\rangle\},\ \{\langle udepartment,\ !rdepartment\rangle\},\ !write\rangle$ $\langle\{\langle department,\ !registrar\rangle,\ \langle type,\ transcript\rangle\},\ !write\rangle$ $\langle\{\langle type,\ application\rangle\},\ \{\langle uid,\ student\rangle\},\ !setStatus\rangle$ $\langle\{\langle department,\ !admissions\rangle,\ \langle type,\ application\rangle\},\ !setStatus\rangle$
HealthcarePN	$\langle\{\langle position, nurse\rangle, \langle type,\ HR\rangle\}, \{\langle uward,\ rward\rangle\},\ addItem\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle teams,\ treatingTeam\rangle\},\ addItem\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle uid,\ patient\rangle\},\ addNote\rangle$ $\langle\{\langle type,\ HR\rangle\}, \{\langle agentFor,\ patient\rangle\},\ addNote\rangle$ $\langle\{\langle type,\ HRitem\rangle\}, \{\langle uid,\ author\rangle\},\ read\rangle$ $\langle\{\langle position, nurse\rangle, \langle type,\ HRitem\rangle, \langle topic,\ !nursing\rangle\}, \{\langle uward,\ rward\rangle\},\ !write\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle specialty,\ !topic\rangle\}, \{\langle teams,\ !treatingTeam\rangle\},\ !addNote\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle topic,\ note\rangle\}, \{\langle uid,\ !patient\rangle\},\ !addItem\rangle$ $\langle\{\langle type,\ HRitem\rangle, \langle topic,\ note\rangle\}, \{\langle agentFor,\ !patient\rangle\},\ !write\rangle$
ProjectManagementPN	$\langle\{\langle adminRole, manager\rangle, \langle type,\ budget\rangle\}, \{\langle udepartment, odepartment\rangle\},\ approve\rangle$ $\langle\{\langle\rangle, \langle type,\ schedule\rangle\}, \{\langle projectsLed, project\rangle\},\ read\rangle$ $\langle\{\langle\rangle, \langle type,\ schedule\rangle\}, \{\langle projects, project\rangle\},\ read\rangle$ $\langle\{\langle\rangle, \langle type,\ task\rangle\}, \{\langle task, rid\rangle\},\ setStatus\rangle$ $\langle\{\langle\rangle, \langle type, task\rangle, \langle proprietary, false\rangle\}, \{\langle projects, project\rangle, \langle oexpertise, rexpertise\rangle\}, read\rangle$ $\langle\{\langle isEmployee, True\rangle, \langle type, task\rangle, \langle proprietary, false\rangle\}, \{\langle projects, project\rangle,$ $\langle oexpertise, rexpertise\rangle\}, request\rangle$ $\langle\{\langle adminRole, auditor\rangle, \langle type,\ budget\rangle\}, \{\langle projects, project\rangle\},\ !write\rangle$ $\langle\{\langle adminRole, accountant\rangle, \langle type,\ budget\rangle\}, \{\langle projects, !project\rangle\},\ !write\rangle$ $\langle\{\langle adminRole, accountant\rangle, \langle type,\ !task\rangle\}, \{\langle projects, project\rangle\},\ setCost\rangle$ $\langle\{\langle adminRole, !planner\rangle, \langle type,\ schedule\rangle\}, \{\langle projects, project\rangle\},\ !write\rangle$ $\langle\{\langle adminRole, planner\rangle, \langle type,\ !budget\rangle\}, \{\langle projects, project\rangle\},\ setSchedule\rangle$

Appendix B Chapter 4 Sample Policies

In the following, we present details of the manual policies we used in our experiments in Chapter 4. We defined three sample manual policies for our experiments. Table 17 and Table 18 show their operations, attributes, and corresponding attribute values. Table 19 shows the authorization policy rules for each manual policy.

Table 17: Operations in sample manual policies

Policy	Operation
Manual Policy 1 (π_{m1})	lights_on_off order_online set_temperature turn_on_cooler turn_on_heater install_software_update mower_on_off connect_new_device view_lock_state play_music
Manual Policy 2 (π_{m2})	lights_on_off order_online set_temperature play_music turn_on_cooler turn_on_heater camera_on_off view_temperature_log answer_door
Manual Policy 3 (π_{m3})	lights_on_off order_online set_temperature play_music turn_on_cooler turn_on_heater camera_on_off view_temperature_log answer_door mower_on_off

Table 18: Attributes and their corresponding values of sample manual policies

Policy	Attribute	Attribute Value
Manual Policy 1 (π_{m1})	Time	day midday night midnight
	Role	mother father child visiting_family guest
	Location	inside_home outside_home yard basement
Manual Policy 2 (π_{m2})	Time	morning afternoon evening night
	Role	mother father child baby_sitter neighbor
	Location	kitchen living_room bedroom1 bedroom2
Manual Policy 3 (π_{m3})	Time	day morning afternoon evening night midnight
	Location	kitchen living_room bedroom1 bedroom2 inside_home outside_home yard basement
	Role	parent mother father child minor_child teenager guest baby_sitter neighbor visiting_family

Table 19: Policy rules of sample manual policies

Policy	Rules
Manual Policy 1 (π_{m1})	⟨"location" : "inside_home","capability" : "lights_on_off"⟩ ⟨"capability" : "order_online","role" : "mother"⟩ ⟨"capability" : "order_online","role" : "father"⟩ ⟨"location" : "inside_home","capability" : "set_temperature"⟩ ⟨"time" : "day","capability" : "play_music"⟩ ⟨"time" : "night","capability" : "play_music","role" : "mother"⟩ ⟨"capability" : "turn_on_cooler","role" : "father"⟩ ⟨"capability" : "install_software_update","role" : "father"⟩ ⟨"capability" : "connect_new_device","role" : "father"⟩ ⟨"capability" : "view_lock_state"⟩ ⟨"location" : "yard","capability" : "mower_on_off","role" : "visiting_family"⟩
Manual Policy 2 (π_{m2})	⟨"time" : "morning","capability" : "play_music"⟩ ⟨"time" : "evening","capability" : "play_music","role" : "mother"⟩ ⟨"capability" : "lights_on_off","role" : "mother"⟩ ⟨"capability" : "turn_on_heater","role" : "father"⟩ ⟨"location" : "living_room","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"capability" : "view_temperature_log"⟩ ⟨"capability" : "order_online","role" : "father"⟩ ⟨"time" : "morning","location" : "living_room","capability" : "lights_on_off", "role":"neighbor" ⟩ ⟨"capability" : "answer_door","role" : "mother"⟩ ⟨"capability" : "answer_door","role" : "father"⟩ ⟨"capability" : "set_temperature"⟩
Manual Policy 3 (π_{m3})	⟨"time" : "morning","capability" : "play_music"⟩ ⟨"time" : "afternoon","capability" : "play_music"⟩ ⟨"time" : "day","capability" : "play_music"⟩ ⟨"location" : "yard","capability" : "mower_on_off","role" : "visiting_family"⟩ ⟨"location" : "outside_home","capability" : "mower_on_off","role" : "visiting_family"⟩ ⟨"location" : "yard","capability" : "mower_on_off","role" : "neighbor"⟩ ⟨"location" : "outside_home","capability" : "mower_on_off","role" : "neighbor"⟩ ⟨"location" : "yard","capability" : "mower_on_off","role" : "baby_sitter"⟩ ⟨"location" : "outside$_h$ome","capability" : "mower_on_off","role" : "baby_sitter"⟩ ⟨"location" : "yard","capability" : "mower_on_off","role" : "guest"⟩ ⟨"location" : "outside$_h$ome","capability" : "mower_on_off","role" : "guest"⟩ ⟨"location" : "living$_r$oom","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "living$_r$oom","capability" : "answer_door","role" : "neighbor"⟩ ⟨"location" : "living$_r$oom","capability" : "answer_door","role" : "visiting_family"⟩ ⟨"location" : "living_room","capability" : "answer_door","role" : "guest"⟩ ⟨"location" : "kitchen","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "kitchen","capability" : "answer_door","role" : "neighbor"⟩ ⟨"location" : "kitchen","capability" : "answer$_d$oor","role" : "visiting$_f$amily"⟩ ⟨"location" : "kitchen","capability" : "answer_door","role" : "guest"⟩ ⟨"location" : "bedroom1","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "bedroom1","capability" : "answer_door","role" : "neighbor"⟩ ⟨"location" : "bedroom1","capability" : "answer_door","role" : "visiting_family"⟩ ⟨"location" : "bedroom1","capability" : "answer_door","role" : "guest"⟩ ⟨"location" : "bedroom1","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "bedroom2","capability" : "answer_door","role" : "neighbor"⟩ ⟨"location" : "bedroom2","capability" : "answer_door","role" : "visiting_family"⟩ ⟨"location" : "bedroom2","capability" : "answer_door","role" : "guest"⟩ ⟨"location" : "bedroom2","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "inside$_h$ome","capability" : "answer_door","role" : "baby_sitter"⟩ ⟨"location" : "inside$_h$ome","capability" : "answer_door","role" : "neighbor"⟩ ⟨"location" : "inside$_h$ome","capability" : "answer_door","role" : "visiting_family"⟩ ⟨"location" : "inside_home","capability" : "answer_door","role" : "guest"⟩ ⟨"time" : "evening","capability" : "play_music","role" : "mother"⟩ ⟨"time" : "night","capability" : "play_music","role" : "mother"⟩ ⟨"time" : "evening","capability" : "play_music","role" : "father"⟩ ⟨"time" : "night","capability" : "play_music","role" : "father"⟩ ⟨"time" : "evening","capability" : "play_music","role" : "parent"⟩ ⟨"time" : "night","capability" : "play_music","role" : "parent"⟩

Bibliography

[1] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*, pages 1638–1646, 2014.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] Rima Al-Ali, Petr Hnetynka, Jiri Havlik, Vlastimil Krivka, Robert Heinrich, Stephan Seifermann, Maximilian Walter, and Adrian Juan-Verdejo. Dynamic security rules for legacy systems. In *Proceedings of the 13th European Conference on Software Architecture-Volume 2*, pages 277–284, 2019.

[4] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. Towards an automated extraction of abac constraints from natural language policies. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 105–119. Springer, 2019.

[5] Amazon.com. Amazon employee access challenge. Kaggle.

[6] Luciano Argento, Andrea Margheri, Federica Paci, Vladimiro Sassone, and Nicola Zannone. Towards adaptive access control. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 99–109. Springer, 2018.

[7] Nathalie Baracaldo and James Joshi. An adaptive risk management and access control framework to mitigate insider threats. *Computers & Security*, 39:237–254, 2013.

[8] Matthias Beckerle and Leonardo 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.

[9] D Elliott Bell and Leonard J LaPadula. Secure computer systems: Mathematical foundations. Technical report, MITRE CORP BEDFORD MA, 1973.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[11] Avrim Blum, Adam Kalai, and John Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 203–208, 1999.

[12] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

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

[14] Carlos Cotrini, Thilo Weghorn, and David Basin. Mining abac rules from sparse logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 31–46. IEEE, 2018.

[15] Federal Chief Information Officer Council. Federal identity, credential, and access management (ficam) roadmap and implementation guidance, version 2.0, 2011.

[16] Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

[17] Devos, Nico and Hes, Robin. Kmodes implementation.

[18] Maryem Ait El Hadj, Meryeme Ayache, Yahya Benkaouz, Ahmed Khoumsi, and Mohammed Erradi. Clustering-based approach for anomaly detection in xacml policies. In *SECRYPT*, pages 548–553, 2017.

[19] Maryem Ait El Hadj, Ahmed Khoumsi, Yahya Benkaouz, and Mohammed Erradi. Efficient security policy management using suspicious rules through access log analysis. In *International Conference on Networked Systems*, pages 250–266. Springer, 2019.

[20] Philip WL Fong and Ida 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.

[21] Vanessa Frias-Martinez, Joseph Sherrick, Salvatore J Stolfo, and Angelos D Keromytis. A network access control mechanism based on behavior profiles. In *2009 Annual Computer Security Applications Conference*, pages 3–12. IEEE, 2009.

[22] Vanessa Frias-Martinez, Salvatore J Stolfo, and Angelos D Keromytis. Behavior-based network access control: A proof-of-concept. In *International Conference on Information Security*, pages 175–190. Springer, 2008.

[23] Cyril Goutte, Peter Toft, Egill Rostrup, Finn Å Nielsen, and Lars Kai Hansen. On clustering fmri time series. *NeuroImage*, 9(3):298–310, 1999.

[24] Qi Guo, Jaideep Vaidya, and Vijayalakshmi 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.

[25] Michael A Harrison, Walter L Ruzzo, and Jeffrey D Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[26] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 255–272, Baltimore, MD, 2018. USENIX Association.

[27] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Anomaly discovery and resolution in web access control policies. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 165–174, 2011.

[28] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[29] Padmavathi Iyer and Amirreza 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.

[30] Amani Abu Jabal, Elisa Bertino, Jorge Lobo, Mark Law, Alessandra Russo, Seraphin Calo, and Dinesh Verma. Polisma-a framework for learning attribute-based access control policies. In *European Symposium on Research in Computer Security*, pages 523–544. Springer, 2020.

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

[32] Jing Jin, Gail-Joon Ahn, Hongxin Hu, Michael J Covington, and Xinwen 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.

[33] Leila Karimi, Maryam Aldairi, James Joshi, and Mai Abdelhakim. An automatic attribute based access control policy extraction from access logs. *arXiv preprint arXiv:2003.07270*, 2020.

[34] Leila Karimi and James 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.

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

[36] Martin Kuhlmann, Dalia Shohat, and Gerhard 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.

[37] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pages 817–824, 2008.

[38] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[39] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[40] Paul Marinescu, Chad Parry, Marjori Pomarole, Yuan Tian, Patrick Tague, and Ioannis Papagiannis. Ivd: Automatic learning and enforcement of authorization rules in online social networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1094–1109. IEEE, 2017.

[41] Eric Medvet, Alberto Bartoli, Barbara Carminati, and Elena Ferrari. Evolutionary inference of attribute-based access control policies. In *EMO (1)*, pages 351–365, 2015.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[43] Decebal Mocanu, Fatih Turkmen, Antonio Liotta, et al. Towards abac policy mining from logs with deep learning. In *Proceedings of the 18th International Multiconference, ser. Intelligent Systems*, 2015.

[44] Jonathan D Moffett and Morris S Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing and Electronic Commerce*, 4(1):1–22, 1994.

[45] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin Calo, and Jorge Lobo. Mining roles with semantic meanings. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 21–30. ACM, 2008.

[46] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin Calo, and Jorge Lobo. Mining roles with multiple objectives. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):36, 2010.

[47] Montanez, Ken. Amazon access samples. UCI Machine Learning Repository: Amazon Access Samples Data Set.

[48] Qun Ni, Jorge Lobo, Seraphin Calo, Pankaj Rohatgi, and Elisa 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.

[49] C. J. van Rijsbergen. *Information retrieval. 2.ed.* Butterworths, 1979.

[50] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

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

[52] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

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

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

[55] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[56] Daniel B Suits. Use of dummy variables in regression equations. *Journal of the American Statistical Association*, 52(280):548–551, 1957.

[57] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 1998.

[58] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[59] Hassan Takabi and James BD 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.

[60] Robert L Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[61] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi 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.

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

[63] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[64] Wikipedia contributors. Accuracy paradox-wikipedia, the free encyclopedia, 2018. [Online; accessed 30-September-2019].

[65] Zhongyuan Xu and Scott 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.

[66] Zhongyuan Xu and Scott 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.

[67] Zhongyuan Xu and Scott 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.

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

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