



Polisma - A Framework for Learning Attribute-Based Access Control Policies

Amani Abu Jabal¹(✉), Elisa Bertino¹, Jorge Lobo², Mark Law³,
Alessandra Russo³, Seraphin Calo⁴, and Dinesh Verma⁴

¹ Purdue University, West Lafayette, IN, USA
{aabujaba,bertino}@purdue.edu

² ICREA - Universitat Pompeu Fabra, Barcelona, Spain
jorge.lobo@upf.edu

³ Imperial College London, London, UK
{mark.law09,a.russo}@imperial.ac.uk

⁴ IBM TJ Watson Research Center, Yorktown Heights, NY, USA
{scalco,dverma}@us.ibm.com

Abstract. Attribute-based access control (ABAC) is being widely adopted due to its flexibility and universality in capturing authorizations in terms of the properties (attributes) of users and resources. However, specifying ABAC policies is a complex task due to the variety of such attributes. Moreover, migrating an access control system adopting a low-level model to ABAC can be challenging. An approach for generating ABAC policies is to learn them from data, namely from logs of historical access requests and their corresponding decisions. This paper proposes a novel framework for learning ABAC policies from data. The framework, referred to as *Polisma*, combines data mining, statistical, and machine learning techniques, capitalizing on potential context information obtained from external sources (e.g., LDAP directories) to enhance the learning process. The approach is evaluated empirically using two datasets (real and synthetic). Experimental results show that *Polisma* is able to generate ABAC policies that accurately control access requests and outperforms existing approaches.

Keywords: Authorization rules · Policy mining · Policy generalization

1 Introduction

Most modern access control systems are based on the attribute-based access control model (ABAC) [3]. In ABAC, user requests to protected resources are granted or denied based on discretionary attributes of users, resources, and environmental conditions [7]. ABAC has several advantages. It allows one to specify access control policies in terms of domain-meaningful properties of users, resources, and environments. It also simplifies access control policy administration by allowing access decisions to change between requests by simply changing attribute values, without changing the user/resource relationships underlying the

rule sets [7]. As a result, access control decisions automatically adapt to changes in environments, and in user and resource populations. Because of its relevancy for enterprise security, ABAC has been standardized by NIST and an XML-based specification, known as XACML, has been developed by OASIS [20]. There are several XACML enforcement engines, some of which are publicly available (e.g., AuthZForce [18] and Balana [19]). Recently a JSON profile for XACML has been proposed to address the verbosity of the XML notation. However, a major challenge in using an ABAC model is the manual specification of the ABAC policies that represent one of the inputs for the enforcement engine. Such a specification requires detailed knowledge about properties of users, resources, actions, and environments in the domain of interest [13, 22]. One approach to address this challenge is to take advantage of the many data sources that are today available in organizations, such as user directories (e.g., LDAP directories), organizational charts, workflows, security tracking's logs (e.g., SIEM), and use machine learning techniques to automatically learn ABAC policies from data.

Suitable data for learning ABAC policies could be access requests and corresponding access control responses (i.e., *access control decisions*) that are often collected in different ways and forms, depending on the real-world situation. For example, an organization may log past decisions taken by human administrators [17], or may have automated access control mechanisms based on low-level models, e.g., models that do not support ABAC. If interested in adopting a richer access control model, such an organization could, in principle, use these data to automatically generate access control policies, e.g., logs of past decisions taken by the low-level mechanism could be used as labeled examples for a supervised machine learning algorithm¹.

Learned ABAC policies, however, must satisfy a few key requirements. They must be *correct* and *complete*. Informally, an ABAC policy set is correct if it is able to make the correct decision for any access request. It is complete if there are no access requests for which the policy set is not able to make a decision. Such a case may happen when the attributes provided with a request do not satisfy the conditions of any policy in the set.

To meet these requirements, the following issues need to be taken into account when learning ABAC policies:

- *Noisy examples*. The log of examples might contain decisions which are erroneous or inconsistent. The learning process needs to be robust to noise to avoid learning incorrect policies.
- *Overfitting*. This is a problem associated with machine learning [10] which happens when the learned outcomes are good only at explaining data given as examples. In this case, learned ABAC policies would be appropriate only

¹ Learning policies from logs of access requests does not necessarily mean that there is an existing access control system or that policy learning is aimed to reproduce existing policies or validate them. Such logs may consist of examples of access control decisions provided by a human expert, and learning may be used for example in a coalition environment where a coalition member can get logs from another coalition member to learn policies for similar missions.

for access requests observed during the learning process and fail to control any other potential access request, so causing the learned policy set to be incomplete. The learning process needs to generalize from past decisions.

- *Unsafe generalization.* Generalization is critical to address overfitting. But at the same time generalization should be *safe*, that is, it should not result in learning policies that may have unintended consequences, thus leading to learned policies that are unsound. The learning process has to balance the trade-off between overfitting and safe generalization.

This paper investigates the problem of learning ABAC policies, and proposes a learning framework that addresses the above issues. Our framework learns from logs of access requests and corresponding access control decisions and, when available, context information provided by external sources (e.g., LDAP directories). We refer to our framework as *Polisma* to indicate that our ABAC **p**olicy **l**earner uses **m**ining, **s**tatistical, and **m**achine learning techniques. The use of multiple techniques enables extracting different types of knowledge that complement each other to improve the learning process. One technique captures data patterns by considering the frequency and another one exploits statistics and context information. Furthermore, another technique exploits data similarity. The assembly of these techniques in our framework enables better learning for ABAC policies compared to the other state-of-the-art approaches.

Polisma consists of four steps. In the first step, a data mining technique is used to infer associations between users and resources included in the set of decision examples and based on these associations a set of rules is generated. In the second step, each constructed rule is generalized based on statistically significant attributes and context information. In the third step, authorization domains for users and resources (e.g., which resources were accessed using which operations by a specific user) are considered in order to augment the set of generalized rules with “restriction rules” for restricting access of users to resources by taking into account their authorization domain. Policies learned by those three stages are safe generalizations with limited overfitting. To improve the completeness of the learned set, *Polisma* applies a machine learning (ML) classifier on requests not covered by the learned set of policies and uses the result of the classification to label these data and generate additional rules in an “ad-hoc” manner.

2 Background and Problem Description

In what follows, we introduce background definitions for ABAC policies and access request examples, and formulate the problem of learning ABAC policies.

2.1 ABAC Policies

Attribute-based access control (ABAC) policies are specified as Boolean combinations of conditions on attributes of users and protected resources. The following definition is adapted from Xu *et al.* [24]:

Definition 1 (cf. ABAC Model [24]). *An ABAC model consists of the following components:*

- \mathcal{U} , \mathcal{R} , \mathcal{O} , \mathcal{P} refer, respectively, to finite sets of users, resources, operations, and rules.
- A_U refers to a finite set of user attributes. The value of an attribute $a \in A_U$ for a user $u \in \mathcal{U}$ is represented by a function $d_U(u, a)$. The range of d_U for an attribute $a \in A_U$ is denoted by $V_U(a)$.
- A_R refers to a finite set of resource attributes. The value of an attribute $a \in A_R$ for a resource $r \in \mathcal{R}$ is represented by a function $d_R(r, a)$. The range of d_R for an attribute $a \in A_R$ is denote
- A user attribute expression e_U defines a function that maps every attribute $a \in A_U$, to a value in its range or \perp , $e_U(a) \in V_U(a) \cup \{\perp\}$. Specifically, e_U can be expressed as the set of attribute/value pairs $e_U = \{\langle a_i, v_i \rangle \mid a_i \in A_U \wedge f(a_i) = v_i \in V_U(a_i)\}$. A user u_i satisfies e_U (i.e., it belongs to the set defined by e_U) iff for every user attribute a not mapped to \perp , $\langle a, d_U(u_i, a) \rangle \in e_U$. Similarly, a resource s_i can be defined by a resource attribute expression e_R .
- A rule $\rho \in \mathcal{P}$ is a tuple $\langle e_U, e_R, O, d \rangle$ where $\rho.e_U$ is a user attribute expression, $\rho.e_R$ is a resource attribute expression, $O \subseteq \mathcal{O}$ is a set of operations, and d is the decision of the rule ($d \in \{\text{permit}, \text{deny}\}$)².

The original definition of an ABAC rule does not include the notion of “signed rules” (i.e., rules that specify positive or negative authorizations). By default, $d = \text{permit}$, and an access request $\langle u, r, o \rangle$ for which there exist at least a rule $\rho = \langle e_U, e_R, O, d \rangle$ in \mathcal{P} such that u satisfies e_U (denoted by $u \models e_U$), r satisfies e_R (denoted by $r \models e_R$), and $o \in O$, is permitted. Otherwise, the access request is assumed to be denied. Even though negative authorizations are the default in access control lists, mixed rules are useful when dealing with large sets of resources organized according to hierarchies, and have been widely investigated [21]. They are used in commercial access control systems (e.g., the access control model of SQL Servers provides negative authorizations by means of the DENY authorization command), and they are part of the XACML standard.

2.2 Access Control Decision Examples

An access control decision example (referred to as a *decision example* (l)) is composed of an access request and its corresponding decision. Formally,

Definition 2 (Access Control Decisions and Examples (l)). *An access control decision is a tuple $\langle u, r, o, d \rangle$ where u is the user who initiated the access request, r is the resource target of the access request, o is the operation requested on the resource, and d is the decision taken for the access request. A decision example l is a tuple $\langle u, e_U, r, e_R, o, d \rangle$ where e_U and e_R are a user attribute expression, and a resource attribute expression such that $u \models e_U$, and $r \models e_R$, and the other arguments are interpreted as in an access control decision.*

² Throughout the paper, we will use the dot notation to refer to a component of an entity (e.g., $\rho.d$ refers to the decision of the rule ρ).

There exists an unknown set \mathcal{F} of all access control decisions that should be allowed in the system, but for which we only have partial knowledge through examples. In an example l , the corresponding e_U and e_R can collect a few attributes (e.g., role, age, country of origin, manager, department, experience) depending on the available log information. Note that an access control decision is an example where e_U and e_R are the constant functions that assign to any attribute \perp . We say that an example $\langle u, e_U, r, e_R, o, d \rangle$ belongs to an access control decision set S iff $\langle u, r, o, d \rangle \in S$. Logs of past decisions are used to create an *access control decision example dataset* (see Definition 3). We say that a set of ABAC policies \mathcal{P} controls an access control request $\langle u, r, o \rangle$ iff there is a rule $\rho = \langle e_U, e_R, O, d \rangle \in \mathcal{P}$ that satisfies the access request $\langle u, r, o \rangle$. Similarly, we say that the request $\langle u, r, o \rangle$ is covered by \mathcal{F} iff $\langle u, r, o, d \rangle \in \mathcal{F}$, for some decision d . Therefore, \mathcal{P} can be seen as defining the set of all access decisions for access requests controlled by \mathcal{P} and, hence, be compared directly with \mathcal{F} - and with some overloading in the notation, we want decisions in \mathcal{P} ($\langle u, r, o, d \rangle \in \mathcal{P}$) to be decisions in \mathcal{F} ($\langle u, r, o, d \rangle \in \mathcal{F}$).

Definition 3 (Access Control Decision Example Dataset (\mathcal{D})). *An access control decision example dataset is a finite set of decision examples (i.e., $\mathcal{D} = \{l_1, \dots, l_n\}$).*

\mathcal{D} is expected to be mostly, but not necessarily, a subset of \mathcal{F} . \mathcal{D} is considered to be *noisy* if $\mathcal{D} \not\subseteq \mathcal{F}$.

2.3 Problem Definition

Using a set of decision examples, extracted from a system history of access requests and their authorized/denied decisions, together with some context information (e.g., metadata obtained from LDAP directories), we want to learn ABAC policies (see Definition 4). The context information provides user and resource identifications, as well as user and resource attributes and their functions needed for an ABAC model. Additionally, it might also provide complementary semantic information about the system in the form of a collection of typed binary relations, $\mathcal{T} = \{t_1, \dots, t_n\}$, such that each relation t_i relates pairs of attribute values, i.e., $t_i \subseteq V_X(a_1) \times V_Y(a_2)$, with $X, Y \in \{\mathcal{U}, \mathcal{R}\}$, and $a_1 \in A_X$, and $a_2 \in A_Y$. For example, one of these relations can represent the organizational chart (department names are related in an *is_member* or *is_part* hierarchical relation) of the enterprise or institution where the ABAC access control system is deployed.

Definition 4 (Learning ABAC Policies by Examples and Context (LAPEC)). *Given an access control decision example dataset \mathcal{D} , and context information comprising the sets \mathcal{U} , \mathcal{R} , \mathcal{O} , the sets A_U and A_R , one of which could be empty, the functions assigning values to the attributes of the users and resources, and a possibly empty set of typed binary relations, $\mathcal{T} = \{t_1, \dots, t_n\}$, LAPEC aims at generating a set of ABAC rules (i.e., $\mathcal{P} = \{\rho_1 \dots \rho_m\}$) that are able to control all access requests in \mathcal{F} .*

2.4 Policy Generation Assessment

ABAC policies generated by *LAPEC* are assessed by evaluation of two quality requirements: **correctness**, which refers to the capability of the policies to assign a correct decision to any access request (see Definition 5), and **completeness**, which refers to ensuring that all actions, executed in the domain controlled by an access control system, are covered by the policies (see Definition 6). This assessment can be done against example datasets as an approximation of \mathcal{F} . Since datasets can be noisy, it is possible that two different decision examples for the same request are in the set. Validation will be done only against consistent datasets as we assume that the available set of access control examples is noise-free.

Definition 5 (Correctness). *A set of ABAC policies \mathcal{P} is correct with respect to a consistent set of access control decisions \mathcal{D} if and only if for every request $\langle u, r, o \rangle$ covered by \mathcal{D} , $\langle u, r, o, d \rangle \in \mathcal{P} \rightarrow \langle u, r, o, d \rangle \in \mathcal{D}$.*

Definition 6 (Completeness). *A set of ABAC policies \mathcal{P} is complete with respect to a consistent set of access control decisions \mathcal{D} if and only if, for every request $\langle u, r, o \rangle$, $\langle u, r, o \rangle$ covered by $\mathcal{D} \rightarrow \langle u, r, o \rangle$ is controlled by \mathcal{P} .*

These definitions allow \mathcal{P} to control requests outside \mathcal{D} . The aim is twofold. First, when we learn \mathcal{P} from an example dataset \mathcal{D} , we want \mathcal{P} to be correct and complete with respect to \mathcal{D} . Second, beyond \mathcal{D} , we want to minimize incorrect decisions while maximizing completeness with respect to \mathcal{F} . Outside \mathcal{D} , we evaluate correctness statistically through cross validation with example datasets which are subsets of \mathcal{F} , and calculating Precision, Recall, F1 score, and accuracy of the decisions made by \mathcal{P} . We quantify completeness by the Percentage of Controlled Requests (PCR) which is the ratio between the number of decisions made by \mathcal{P} among the requests covered by an example dataset and the total number of requests covered by the dataset.

Definition 7 (Percentage of Controlled Requests (PCR)). *Given a subset of access control decision examples $\mathcal{N} \subseteq \mathcal{F}$, and a policy set \mathcal{P} , the percentage of controlled requests is defined as follows:*

$$PCR = \frac{|\{\langle u, r, o \rangle \mid \langle u, r, o \rangle \text{ is covered by } \mathcal{N} \text{ and } \langle u, r, o \rangle \text{ is controlled by } \mathcal{P}\}|}{|\{\langle u, r, o \rangle \mid \langle u, r, o \rangle \text{ is covered by } \mathcal{N}\}|}$$

3 The Learning Framework

Our framework comprises four steps, see Fig. 1. For a running example see Appendix B.

3.1 Rules Mining

\mathcal{D} provides a source of examples of user accesses to the available resources in an organization. In this step, we use association rule mining (ARM) [1] to analyze the association between users and resources.

Thus, rules having high association metrics (i.e., support and confidence scores) are kept to generate access control rules. ARM is used for capturing the frequency and data patterns and formalize them in rules. *Polisma* uses Apriori [2] (one of the ARM algorithms) to generate rules

that are *correct and complete* with respect to \mathcal{D} . This step uses only the examples in \mathcal{D} to mine a first set of rules, referred to as *ground rules*. These ground rules potentially are overfitted (e.g., ρ_1 in Fig. 8 in Appendix B) or unsafely-generalized (e.g., ρ_2 in Fig. 8 in Appendix B). Overfitted rules impact completeness over \mathcal{F} while unsafely-generalized rules impact correctness. Therefore, *Polisma* post-processes the ground rules in the next step.

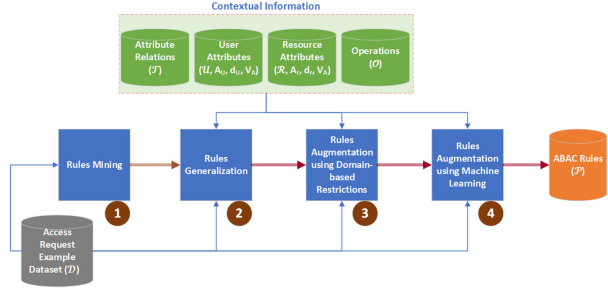


Fig. 1. The architecture of *Polisma*

3.2 Rules Generalization

To address overfitting and unsafe generalization associated with the ground rules, this step utilizes the set of user and resource attributes A_U , A_R provided by either \mathcal{D} , external context information sources, or both. Straightforwardly, e_U (i.e., user expression) or e_R (i.e., resource expression) of a rule can be adapted to expand or reduce the scope of each expression when a rule is overfitted or is unsafely generalized, respectively. In particular, each rule is post-processed based on its corresponding user and resource by analyzing A_U and A_R to statistically “choose” the most appropriate attribute expression that captures the subsets of users and resources having the same permission authorized by the rule according to \mathcal{D} . In this way, this step searches for an attribute expression that minimally generalizes the rules.

Polisma provides two strategies for post-processing ground rules. One strategy, referred to as Brute-force Strategy *BS*, uses only the attributes associated with users and resources appearing in \mathcal{D} . The other strategy assumes that attribute relationship metadata (\mathcal{T}) is also available. Hence, the second strategy, referred to as Structure-based Strategy *SS*, exploits both attributes and their relationships. In what follows, we describe each strategy.

Brute-Force Strategy (BS). To post-process a ground rule, this strategy searches heuristically for attribute expressions that cover the user and the resource of interest. Each attribute expression is weighted statistically to assess its “safety” level for capturing the authorization defined by the ground rule of interest.

The safety level of a user attribute expression e_U for a ground rule ρ is estimated by the proportion of the sizes of two subsets: a) the subset of decision

Algorithm 1. Brute-force Strategy (*BS-U-S*) for Generalizing Rules**Require:** ρ_i : a ground rule, u_i : the user corresponding to ρ_i , \mathcal{D} , A_U .

```

1: Define  $w_{max} = -\infty$ ,  $a_{selected} = \perp$ .
2: for  $a_i \in A_U$  do
3:    $w_i = W_{uav}(u_i, a_i, d_U(u_i, a_i), \mathcal{D})$ 
4:   if  $w_i > w_{max}$  then
5:      $a_{selected} = a_i$ ,  $w_{max} = w_{a_i}$ 
6:   end if
7: end for
8:  $e_U \leftarrow \langle a_{selected}, d_U(u_i, a_{selected}) \rangle$ 
9: Create Rule  $\rho'_i = \langle e_U, \rho_i.e_R, \rho_i.O, \rho_i.d \rangle$ 
10: return  $\rho'_i$ 

```

Algorithm 2. Structure-based Strategy (*SS*) for Generalizing Rules**Require:** ρ_i : a ground rule to be Generalized, u_i : the user corresponding to ρ_i , r_i : the resource corresponding to ρ_i , \mathcal{T} , A_U , A_R .

```

1:  $g_i = G(u_i, r_i, A_U, A_R, \mathcal{T})$ 
2:  $\langle x \in A_U, y \in A_R \rangle = \text{First-Common-Attribute-Value}(g_i)$ 
3:  $e_U \leftarrow \langle x, d_U(u_i, x) \rangle$ ;  $e_R \leftarrow \langle y, d_R(r_i, y) \rangle$ 
4: Create Rule  $\rho'_i = \langle e_U, e_R, \rho_i.O, \rho_i.d \rangle$ 
5: return  $\rho'_i$ 

```

examples in \mathcal{D} such that the access requests issued by users satisfy e_U while the remaining parts of the decision example (i.e., resource, operation, and decision) match the corresponding ones in ρ ; and b) the subset of users who satisfy e_U . The safety level of a resource attribute expression is estimated similarly. Thereafter, the attribute expression with the highest safety level is selected to be used for post-processing the ground rule of interest. Formally:

Definition 8 (User Attribute Expression Weight $W_{uav}(u_i, a_j, d_U(u_i, a_j), \mathcal{D})$). For a ground rule ρ and its corresponding user $u_i \in U$, the weight of a user attribute expression $\langle a_j, d_U(u_i, a_j) \rangle$ is defined by the formula: $W_{uav} = \frac{|U_D|}{|U_C|}$, where $U_D = \{l \in \mathcal{D} \mid d_U(u_i, a_j) = d_U(l.u, a_j) \wedge l.r \in \rho.e_R \wedge l.o \in \rho.O \wedge \rho.d = l.d\}$ and $U_C = \{u_k \in U \mid d_U(u_i, a_j) = d_U(u_k, a_j)\}$.

Different strategies for selecting attribute expressions are possible. They can be based on a single attribute or a combination of attributes, and they can consider either user attributes, resource attributes, or both. Strategies using only user attributes are referred to as *BS-U-S* “for a single attribute” and *BS-U-C* “for a combination of attributes”. Similarly, *BS-R-S* and *BS-R-C* are strategies for a single or a combination of resources attributes. The strategies using the attributes of both users and resources are referred to as *BS-UR-S* and *BS-UR-C*. Due to space limitation, we show only the algorithm using the selection strategy *BS-U-S* (see Algorithm 1).

Structure-Based Strategy (*SS*). The *BS* strategy works without prior knowledge of \mathcal{T} . When \mathcal{T} is available, it can be analyzed to gather information about “hidden” correlations between common attributes of a user and a resource. Such attributes can potentially enhance rule generalization. For example, users working in a department t_i can potentially access the resources owned

by t_i . The binary relations in \mathcal{T} are combined to form a directed graph, referred to as *Attribute-relationship Graph* G (see Definition 9). Traversing this graph starting from the lowest level in the hierarchy to the highest one allows one to find the common attributes values between users and resources. Among the common attribute values, the one with the least hierarchical level, referred to as *first common attribute value*, has heuristically the highest safety level for generalization because the generalization using such attribute value supports the least level of generalization. Subsequently, to post-process a ground rule, this strategy uses \mathcal{T} along with A_U , A_R of both the user and resource of interest to build G . Thereafter, G is used to find the *first common attribute value* between the user and resource of that ground rule to be used for post-processing the ground rule of interest (as described in Algorithm 2).

Definition 9 (Attribute-relationship Graph G). *Given A_U , A_R , ρ and \mathcal{T} , the attribute-relationship graph (G) of ρ is a directed graph composed of vertices V and edges E where*

$$\begin{aligned} V &= \{v \mid \forall u_i \in \rho.e_U, \forall a_i \in A_U, v = d_U(u_i, a_i)\} \\ &\cup \{v \mid \forall r_i \in \rho.e_R, \forall a_i \in A_R, v = d_R(r_i, a_i)\}, \text{ and} \\ E &= \{(v_1, v_2) \mid \exists t_i \in \mathcal{T} \wedge (v_1, v_2) \in t_i \wedge v_1, v_2 \in V\}. \end{aligned}$$

Proposition 1. *Algorithms 1 and 2 output generalized rules that are correct and complete with respect to \mathcal{D} .*

As discussed earlier, the first step generates ground rules that are either overfitted or unsafely-generalized. This second step post-processes unsafely-generalized rules into safely-generalized ones; hence, improving correctness. It may also post-process overfitted rules into safely-generalized ones; hence, improving completeness. However, completeness can be further improved as described in the next subsections.

3.3 Rules Augmentation Using Domain-Based Restrictions

“Safe” generalization of ground rules is one approach towards fulfilling completeness. Another approach is to analyze the authorization domains of users and resources appearing in \mathcal{D} to augment restriction rules, referred to as *domain-based restriction rules*³. Such restrictions potentially increase safety by avoiding erroneous accesses. Basically, the goal of these restriction rules is to augment negative authorization.

One straightforward approach for creating domain-based restriction rules is to analyze the authorization domain for each individual user and resource. However, such an approach leads to the creation of restrictions suffering from overfitting. Alternatively, the analysis can be based on groups of users or resources.

³ This approach also implicitly improves correctness.

Identifying such groups requires pre-processing A_U and A_R . Therefore, this step searches heuristically for preferable attributes to use for partitioning the user and resource sets. The heuristic prediction for preferable attributes is based on selecting an attribute that partitions the corresponding set evenly. Hence, estimating the attribute distribution of users or resources allows one to measure the ability of the attribute of interest for creating even partitions⁴. One method for capturing the attribute distribution is to compute the attribute entropy as defined in Eq. 1. The attribute entropy is maximized when the users are distributed evenly among the user partitions using the attribute of interest (i.e., sizes of the subsets in $G_U^{a_i}$ are almost equal). Thereafter, the attribute with the highest entropy is the preferred one.

Given the preferred attributes, this step constructs user groups as described in Definition⁵ 10. These groups can be analyzed with respect to \mathcal{O} as described in Algorithm 3. Consequently, user groups $G_U^{a_i}$ and resource groups $G_R^{a_i}$ along with \mathcal{O} comprise two types of authorization domains: operations performed by each user group, and users' accesses to each resource group. Subsequently, the algorithm augments restrictions based on these access domains as follows.

- It generates deny restrictions based on user groups (similar to the example in Fig. 11 in Appendix B). These restriction rules deny any access request from a specific user group using a specific operation to access any resource.
- It generates deny restrictions based on both groups of users and resources. These restriction rules deny any access request from a specific user group using a specific operation to access a specific resource group.

On the other hand, another strategy assumes prior knowledge about preferred attributes to use for partitioning the user and resource sets (referred to as Semantic Strategy (SS)).

Definition 10 (Attribute-based User Groups $G_U^{a_i}$). *Given \mathcal{U} and $a_i \in A_U$, \mathcal{U} is divided into is a set of user groups $G_U^{a_i} \{g_1^{a_i}, \dots, g_k^{a_i}\}$ where $(g_i^{a_i} = \{u_1, \dots, u_m\} \mid \forall u', u'' \in g_i, d_U(u', a_i) = d_U(u'', a_i), m \leq |\mathcal{U}| \wedge (g_i \cap g_j = \phi \mid i \neq j) \wedge (k = |V_U(a_i)|))$.*

$$Entropy(G_U^{a_i}, a_i) = \left(\frac{-1}{\ln m} * \sum_{j=1, g_j \in G_U^{a_i}}^m p_j * \ln p_j \right), \text{ where } m = |G_U^{a_i}|, p_j = \frac{|g_j|}{\sum_{l=1, g_l \in G_U^{a_i}}^m |g_l|} \quad (1)$$

Proposition 2. *Step 3 outputs generalized rules that are correct and complete with respect to \mathcal{D} .*

⁴ Even distribution tends to generate smaller groups. Each group potentially has a similar set of permissions. A large group of an uneven partition potentially includes a diverse set of users or resource; hence hindering observing restricted permissions.

⁵ The definition of constructing resource groups is analogous to that of user groups.

3.4 Rules Augmentation Using Machine Learning

The rules generated from the previous steps are generalized using domain knowledge and data statistics extracted from \mathcal{D} . However, these steps do not consider generalization based on the similarity of access requests. Thus, these rules cannot control a new request that is similar to an old one (i.e., a new request has a similar pattern to one of the old requests, but the attribute values of the new request do not match the old ones).

A possible prediction approach is to use an ML classifier that builds a model based on the attributes provided by \mathcal{D} and context information. The generated model implicitly creates patterns of accesses and their decisions, and will be able to predict the decision for any access request based on its similarity to the ones controlled by the rules generated by the previous steps. Thus, this step creates new rules based on these predictions. Once these new rules are created, *Polisma* repeats Step 2 to safely generalize the ML-based augmented rules. Notice that ML classification algorithms might introduce some inaccuracy when performing the prediction. Hence, we utilize this step only for access requests that are not covered by the rules generated by the previous three steps. Thus, the inaccuracy effect associated with the ML classifier is minimized but the *correctness and completeness* are preserved.

Note that *Polisma* is used as a one-time learning. However, if changes arise in terms of regulations, security policies and/or organizational structure of the organization, the learning can be executed again (i.e., on-demand learning).

4 Evaluation

In this section, we summarize the experimental methodology and report the evaluation results of *Polisma*.

4.1 Experimental Methodology

Datasets. To evaluate *Polisma*, we conducted several experiments using two datasets: one is a synthetic dataset (referred to as *project management (PM)* [24]), and the other is a real one (referred to as *Amazon*⁶). The *PM* dataset has been generated by the Reliable Systems Laboratory at Stony Brook University based on a probability distribution in which the ratio is 25 for rules, 25 for resources, 3 for users, and 3 for operations. In addition to decision examples, *PM* is tagged with context information (e.g., attribute relationships and attribute semantics). We used such a synthetic dataset (i.e., *PM*) to show the impact of the availability of such semantic information on the learning process and the quality of the generated rules. Regarding the Amazon dataset, it is a historical dataset collected in 2010 and 2011. This dataset is an anonymized sample of access provisioned within the Amazon company. One characteristic of

⁶ <http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples>.

Table 1. Overview of datasets

	Datasets	
	<i>Project Management (PM)</i>	<i>Amazon (AZ)</i>
# of decision examples	550	1000
# of users	150	480
# of resources	292	271
# of operations	7	1
# of examples with a “Permit” decision	505	981
# of examples with a “Deny” decision	50	19
# of User Attributes	5	12
# of Resource Attributes	6	1

the Amazon dataset is that it is sparse (i.e., less than 10% of the attributes are used for each sample). Furthermore, since the Amazon dataset is large (around 700K decision examples), we selected a subset of the Amazon dataset (referred to as *AZ*). The subset was selected randomly based on a statistical analysis [11] of the size of the Amazon dataset where the size of the selected samples suffices a confidence score of 99% and a margin error score of 4%. Table 1 shows statistics about the *PM* and *AZ* datasets.

Comparative Evaluation. We compared *Polisma* with three approaches. We developed a *Naïve ML approach* which utilizes an ML classifier to generate rules. A classification model is trained using \mathcal{D} . Thereafter, the trained model is used to generate rules without using the generalization strategies which are used in *Polisma*. The rules generated from this approach are evaluated using another set of new decision examples (referred to as \mathcal{N})⁷. Moreover, we compared *Polisma* with two recently published state-of-the-art approaches for learning ABAC policies from logs: a) Xu & Stoller Miner [24], and b) Rhapsody by Cotrini *et al.* [5].

Evaluation and Settings. On each dataset, *Polisma* is evaluated by splitting the dataset into training \mathcal{D} (70%) and testing \mathcal{N} (30%) subsets. We performed a 10-fold cross-validation on \mathcal{D} for each dataset. As the ML classifiers used for the fourth step of *Polisma* and the naïve approach, we used a combined classification technique based on majority voting where the underlying classifiers are Random Forest and kNN⁸. All experiments were performed on a 3.6 GHz Intel Core i7 machine with 12 GB memory running on 64-bit Windows 7.

⁷ Datasets are assumed to be noise-free, that is, $(\mathcal{N} \subset \mathcal{F}) \wedge (\mathcal{D} \subset \mathcal{F}) \wedge (\mathcal{N} \cap \mathcal{D} = \phi)$. Note that \mathcal{F} is the complete set of control decisions which we will never have in a real system.

⁸ Other algorithms can be used. We used Random Forest and kNN classifiers since they showed better results compared to SVM and Adaboost.

Evaluation Metrics. The correctness of the generated ABAC rules is evaluated using the standard metrics for the classification task in the field of machine learning. Basically, predicted decisions for a set of access requests are compared with their ground-truth decisions. Our problem is analogous to a two-class classification task because the decision in an access control system is either “permit” or “deny”. Therefore, for each type of the decisions, a confusion matrix is prepared to enable calculating the values of accuracy, precision, recall, and F1 score as outlined in Eqs. 2-3⁹.

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}, Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

Regarding the completeness of the rules, they are measured using *PCR* (see Definition 7).

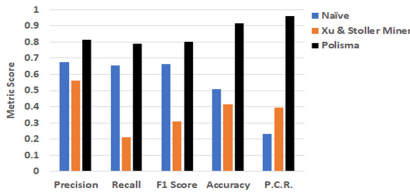


Fig. 2. Comparison of Naive, Xu & Stoller Miner, and *Polisma* Using the *PM* Dataset

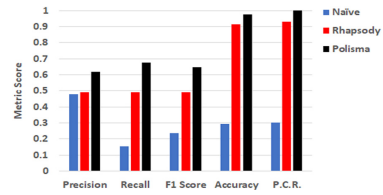
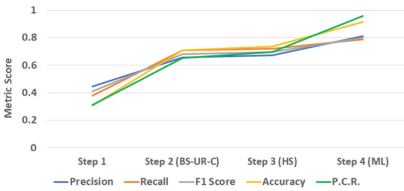
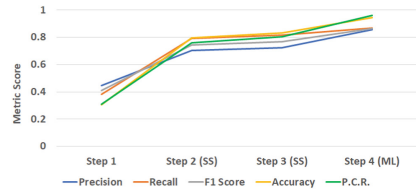


Fig. 3. Comparison of Naive, Rhapsody, and *Polisma* Using the *AZ* Dataset



(a) *Polisma* (BS-HS-ML)



(b) *Polisma* (SS-SS-ML)

Fig. 4. Evaluation of *Polisma* using the *PM* dataset.

4.2 Experimental Results

Polisma vs. Other Learning Approaches. Here, *Polisma* uses the default strategies in each step (i.e., the *BS-UR-C* strategy in the second step and the *HS* strategy in the third step¹⁰) and the impact of the other strategies in *Polisma*

⁹ F1 Score is the harmonic mean of precision and recall.

¹⁰ Since the *AZ* dataset does not contain resource attributes, *BS-R-C* (instead of *BS-UR-C*) is executed in the second step and the execution of the third step is skipped.

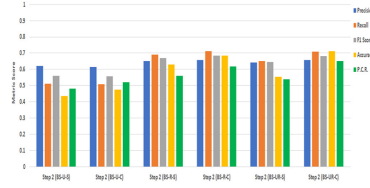


Fig. 5. Comparison between the variants of the brute-force strategy (Step 2) using the *PM* dataset.

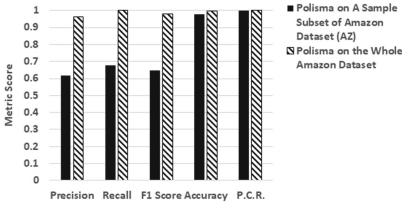


Fig. 6. *Polisma* Evaluation on the Amazon Dataset (a sample subset and the whole set).

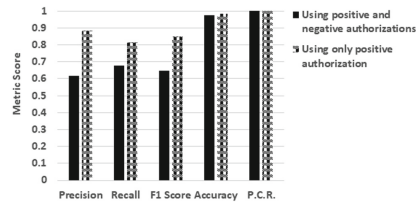


Fig. 7. *Polisma* Evaluation on a sample subset of Amazon Dataset for only positive authorizations.

is evaluated next. The results in Fig. 2 show that *Polisma* achieves better results compared to the naïve and Xu & Stoller Miner using the *PM* dataset. In this comparison, we did not include Rhapsody because the default parameters of Rhapsody leads to generating no rule. With respect to Xu & Stoller Miner, *Polisma*'s F1 score (*PCR*) improves by a factor of 2.6(2.4). This shows the importance of integrating multiple techniques (i.e., data mining, statistical, and ML techniques) in *Polisma* to enhance the policy learning outcome. Moreover, Xu & Stoller Miner requires prior knowledge about the structure of context information while *Polisma* benefits from such information when available. Xu & Stoller Miner [24] generates rules only for positive authorization (i.e., no negative authorization). This might increase the possibility of vulnerabilities and encountering incorrect denials or authorizations. Moreover, Xu & Stoller in their paper [24] used different metrics from the classical metrics for data mining that are used in our paper. In their work [24], they used a similarity measure between the generated policy and the real policy which given based on some distance function. However, such a metric does not necessarily reflect the correctness of the generated policy (i.e., two policies can be very similar, but their decisions are different). In summary, this experiment shows that the level of correctness (as indicated by the precision, recall and F1 score) and completeness (as indicated by the *PCR*) of the policies that are generated by *Polisma* is higher than for the policies generated by Xu & Stoller Miner and the naïve approach due to the reasons explained earlier (i.e., the lack of negative authorization rules), as well as generating generalized rules without considering the safety of generalization. Furthermore, as shown in Fig. 3, *Polisma* also outperforms the naïve

approach and Rhapsody using the *AZ* dataset. Rhapsody [5] only considers positive authorization (similar to the problem discussed above about Xu & Stoller Miner [24]); hence increasing the chances of vulnerabilities. In this comparison, we have excluded Xu & Stoller Miner because of the shortage of available resource attributes information in the *AZ* dataset. Concerning the comparison with the naïve approach, on the *PM* dataset, the F1 score (*PCR*) of *Polisma* improves by a factor of 1.2 (4.1) compared to that of the naïve. On the *AZ* dataset, *Polisma*'s F1 score (*PCR*) improves by a factor of 2.7 (3.3) compared to the naïve. Both *Polisma* and the naïve approach use ML classifiers. However, *Polisma* uses an ML classifier for generating only some of the rules. This implies that among *Polisma* steps which improve the results of the generated rules (i.e., Steps 2 and Step 4), the rule generalization step is essential for enhancing the final outcome. In summary, the reported results show that the correctness level of the rules generated by *Polisma* is better than that of the ones generated by Rhapsody and the naïve approach (as indicated by the difference between the precision, recall, and F1 scores metrics). Meanwhile, the difference between the completeness level (indicated by *PCR*) of the generated rules using *Polisma* and that of Rhapsody is not large. The difference in terms of completeness and correctness is a result of Rhapsody missing the negative authorization rules.

Steps Evaluation. Figure 4 shows the evaluation results for the four steps of *Polisma* in terms of precision, recall, F1 score, accuracy, and *PCR*. In general, the quality of the rules improves gradually for the two datasets, even when using different strategies in each step. The two plots show that the strategies that exploit additional knowledge about user and resource attributes (e.g., \mathcal{T}) produce better results when compared to the ones that require less prior knowledge. Moreover, the quality of the generated rules is significantly enhanced after the second and the fourth steps. This shows the advantage of using safe generalization and similarity-based techniques. On the other hand, the third step shows only a marginal enhancement on the quality of rules. However, the rules generated from this step can be more beneficial when a large portion of the access requests in the logs are not allowed by the access control system.

We also conducted an evaluation on the whole Amazon dataset; the results are shown in Fig. 6. On the whole dataset, *Polisma* achieves significantly improved scores compared to those when using the *AZ* dataset because *Polisma* was able to utilize a larger set of decision examples. Nonetheless, given that the size of *AZ* was small compared to that of the whole Amazon dataset, *Polisma* outcomes using *AZ*¹¹ are promising. In summary, the increase of recall can be interpreted as reducing denials of authorized users (i.e., smaller number of false-negatives). A better precision value is interpreted as being more accurate with the decisions (i.e., smaller number of false-positives). Figures 4a–4b show that most of the reduction of incorrect denials is achieved in Steps 1 and 2, whereas the other steps improve precision which implies more correct decisions (i.e., decreas-

¹¹ We also experimented samples of different sizes (i.e., 2k–5k), the learning results using these sample sizes showed slight improvement of scores.

ing the number of false-positives and increasing the number of true-positives). As Fig. 7 shows, the results improve significantly when considering only positive authorizations. This is due to the fact that the negative examples are few and so they are not sufficient in improving the learning of negative authorizations. As shown in the figure, precision, recall, and F1 score increase indicating that the learned policies are correct. Precision is considered the most effective metric especially for only positive authorizations to indicate the correctness of the generated policies since higher precision implies less incorrect authorizations.

Variants of the Brute-Force Strategy. As the results in Fig. 5 show, using resource attributes for rules generalization is better than using the user attributes. The reason is that the domains of the resource attributes (i.e., $V_R(a_i) \mid a_i \in A_R$) is larger than that of the user attributes (i.e., $V_U(a_j) \mid a_j \in A_U$) in the *PM* dataset. Thus, the selection space of attribute expressions is expanded; hence, it potentially increases the possibility of finding better attribute expression for safe generalization. In particular, using resources attributes achieves 23% and 19% improvement F1 score and *PCR* when compared to that of using users attributes producing a better quality of the generated policies in terms of correctness (as indicated by the values of precision, recall, and F1 score) and completeness (as indicated by the *PCR* value). Similarly, performing the generalization using both user and resource attributes is better than that of using only user attribute or resource attributes because of the larger domains which can be exploited to find the best attribute expression for safe generalization. In particular, *BS-UR-C* shows a significant improvement compared to *BS-U-C* (22% for F1 score (which reflects a better quality of policies in terms of correctness), and 25% for *PCR* (which reflects a better quality of policies in terms of completeness)). In conclusion, the best variant of the *BS* strategy is the one that enables exploring the largest set of possible attribute values to choose the attribute expression which has the highest safety level.

5 Related Work

Policy mining has been widely investigated. However, the focus has been on RBAC role mining that aims at finding roles from existing permission data [16, 23]. More recent work has focused on extending such mining-based approaches to ABAC [5, 8, 14, 15, 24]. Xu and Stoller [24] proposed the first approach for mining ABAC policies from logs. Their approach iterates over a log of decision examples greedily and constructs candidate rules; it then generalizes these rules by utilizing merging and refinement techniques. Medvet *et al.* [14] proposed an evolutionary approach which incrementally learns a single rule per group of decision examples utilizing a divide-and-conquer algorithm. Cotrini *et al.* [5] proposed another rule mining approach, called Rhapsody, based on APRIORI-SD (a machine-learning algorithm for subgroup discovery) [9]. It is incomplete, only mining rules covering a significant number of decision examples. The mined rules are then filtered to remove any refinements. All of those approaches [5, 14, 24] mainly rely on decision

logs assuming that they are sufficient for rule generalization. However, logs, typically being very sparse, might not contain sufficient information for mining rules of good quality. Thus, those approaches may not be always applicable. Moreover, neither of those approaches is able to generate negative authorization rules.

Mocanu *et al.* [15] proposed a deep learning model trained on logs to learn a Restricted Boltzmann Machine (RBM). Then, the learned model is used to generate candidate rules. Their proposed system is still under development and further testing is needed. Karimi and Joshi [8] proposed to apply clustering algorithms over the decision examples to predict rules, but they don't support rules generalization. To the best of our knowledge, *Polisma* is the first generic framework that incorporates context information, when available, along with decisions logs to increase accuracy. Another major difference of *Polisma* with respect to existing approaches is that *Polisma* can use ML techniques, such as statistical ML techniques, for policy mining while at the same time being able to generate ABAC rules expressed in propositional logics.

6 Conclusion and Future Work

In this paper, we have proposed *Polisma*, a framework for learning ABAC policies from examples and context information. *Polisma* comprises four steps using various techniques, namely data mining, statistical, and machine learning. Our evaluations, carried out on a real-world decision log and a synthetic log, show that *Polisma* is able to learn policies that are both complete and correct. The rules generated by *Polisma* using the synthetic dataset achieve an F1 score of 0.80 and *PCR* of 0.95; when using the real dataset, the generated rules achieve an F1 score of 0.98 and *PCR* of 1.0. Furthermore, by using the semantic information available with the synthetic dataset, *Polisma* improves the F1 score to reach 0.86. As part of future work, we plan to extend our framework by integrating an inductive learner [12] and a probabilistic learner [6]. We also plan to extend our framework to support policy transfer and policy learning explainability. In addition, we plan to extend *Polisma* with conflict resolution techniques to deal with input noisy data. Another direction is to integrate in *Polisma* different ML algorithms, such as neural networks. Preliminary experiments about using neural networks for learning access control policies have been reported [4]. However, those results show that neural networks are not able to accurately learn negative authorizations and thus work is required to enhance them.

Acknowledgment. This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Jorge Lobo was also supported

by the Spanish Ministry of Economy and Competitiveness under Grant Numbers TIN201681032P, MDM20150502, and the U.S. Army Research Office under Agreement Number W911NF1910432.

A Additional Algorithms

Algorithm 3 outlines the third step of *Polisma* for augmenting rules using domain-based restrictions.

Algorithm 3. Rules Augmentation using Domain-based Restrictions

Require: $\mathcal{D}, \mathcal{U}, \mathcal{R}, x$: a preferable attribute of users, y : a preferable attribute of resources, \mathcal{O} .
1: $G_U = \mathcal{G}_U^{a_i}(\mathcal{U}, x)$; $G_R = \mathcal{G}_R^{a_i}(\mathcal{R}, y)$
2: $\forall g_i \in G_U, O_{g_i}^u \rightarrow \{\}$; $\forall g_i \in G_R, O_{g_i}^r \rightarrow \{\}$; $\forall g_i \in G_R, U_{g_i}^r \rightarrow \{\}$
3: **for** $l_i \in \mathcal{D}$ **do**
4: $g' \leftarrow g_i \in G_U \mid l_i.u \in g_i \wedge l_i.d = \text{Permit}; O_{g'}^u \rightarrow O_{g'}^u \cup l_i.o$
5: $O_{g''}^r \rightarrow O_{g''}^r \cup l_i.o$; $U_{g''}^r \rightarrow U_{g''}^r \cup l_i.u$
6: **end for**
7: $\mathcal{P}' \rightarrow \{\}$
8: **for** $g_i \in G_U$ **do**
9: $\rho_i = \langle \langle x, d_U(u_i, x) \rangle, *, o_i, \text{Deny} \rangle \mid u_i \in g_i \wedge o_i \in (\mathcal{O} \setminus O_{g_i}^u)$; $\mathcal{P}' \rightarrow \mathcal{P}' \cup \rho_i$
10: **end for**
11: **for** $g_i \in G_R$ **do**
12: $\rho_i = \langle \langle x, d_U(u_i, x) \rangle, \langle y, d_R(r_i, y) \rangle, *, \text{Deny} \rangle \mid r_i \in g_i \wedge u_i \in (\mathcal{U} \setminus U_{g_i}^r)$; $\mathcal{P}' \rightarrow \mathcal{P}' \cup \rho_i$
13: **end for**
14: **return** \mathcal{P}'

B Running Example of Policy Learning Using *Polisma*

Consider a system including users and resources both associated with projects. User attributes, resource attributes, operations, and possible values for two selected attributes are shown in Table 2. Assume that a log of access control decision examples is given.

Table 2. Details about A Project Management System

User Attributes (A_U)	{id, role, department, project, technical area}
Resource Attributes (A_R)	{id, type, department, project, technical area}
Operations List (\mathcal{O})	{request, read, write, setStatus, setSchedule, approve, setCost}
$V_U(\text{role})$	{planner, contractor, auditor, accountant, department manager, project leader}
$V_R(\text{type})$	{budget, schedule, and task}



 Overfitted Rule	$\rho_1: \langle \text{user (ID: planner-1)}, \text{resource (ID: schedule-1-0)}, \text{read, permit} \rangle$
 Unsafe Generalized Rule	$\rho_2: \langle \text{user (ID: acct-4)}, \text{resource (type: Task)}, \text{setCost, permit} \rangle$

Fig. 8. Examples of ground rules generated from rule mining based on the specifications of the running example

B.1 Rules Generalization

Brute-Force Strategy (BS). Assume that *BS-UR-C* is used to generalize ρ_2 defined in Fig. 8. A_U is $\{id, role, department, project, technical\ area\}$ and A_R is $\{id, type, department, project, technical\ area\}$. Moreover, ρ_2 is able to control the access for the user whose ID is “acct-4” when accessing a resource whose type is task. The attribute values of the user and resources controlled by ρ_2 are analyzed. To generalize ρ_2 using *BS-UR-C*, each attribute value is weighted as shown in Fig. 9. For weighting each user/resource attribute value, the proportion of the sizes of two user/resource subsets is calculated according to Definition 8.

In particular, for the value of the “department” attribute corresponding to the user referred by ρ_2 (i.e., “d1”) (Fig. 9b), two user subsets are first found: a) the subset of the users belonging to department “d1”; and b) the subset of the users belonging to department “d1” and having a permission to perform the “setCost” operation on a resource of type “task” based on \mathcal{D} . Then, the ratio of the sizes of these subsets is considered as the weight for the attribute value “d1”. The weights for the other user and resource attributes values are calculated similarly. Thereafter, the user attribute value and resource attribute value with the highest weights are chosen to perform the generalization. Assume that the value of the “department” user attribute is associated with the highest weight and the “project” resource attribute is associated with the highest weight. ρ_2 is generalized as:

$\rho'_2 = \langle \text{user}(\text{department: d1}), \text{resource}(\text{type: task, project: d1-p1}), \text{setCost, permit} \rangle$

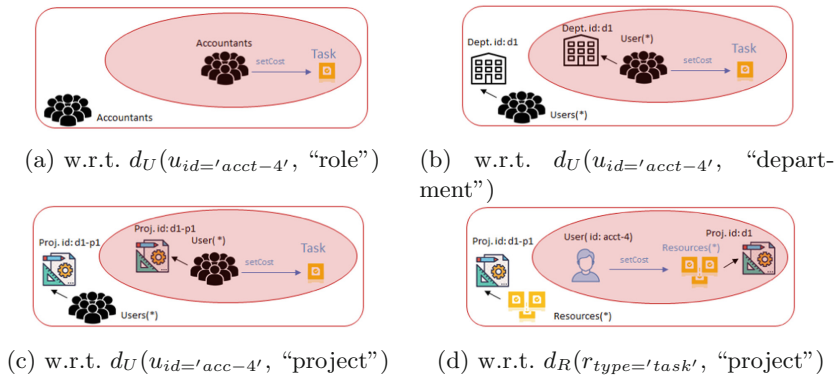


Fig. 9. Generalization of ρ_2 defined in Fig. 8 using the Brute Force Strategy (*BS-UR-C*)

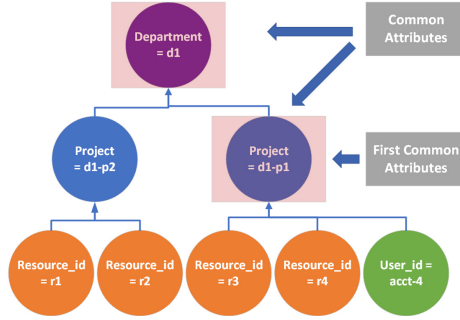


Fig. 10. Generalization of p_2 defined in Fig. 8 using Structure-based Strategy: An example of Attribute-relationship Graph

Structure-Based Strategy (SS). Assume that SS is used to generalize p_2 , defined in Fig. 8. Also, suppose that Polisma is given the following information:

- The subset of resources R' satisfying $p_2.e_R$ has two values for the project attribute (i.e., “d1-p1”, “d1-p2”).
- The user “acct-4” belongs to the project “d1-p1”.
- R' and “acct-4” belong to the department “d1”.
- $\mathcal{T} = \{(\text{“d1-p1”}, \text{“d1”}), (\text{“d1-p2”}, \text{“d1”})\}$.

G is constructed as shown in Fig. 10. Using G , the two common attributes for “acct-4” and R' are “d1-p1” and “d1” and the first common attribute is “d1-p1”. Therefore, p_2 is generalized as follows:

$\rho_2'' = \langle \text{user}(\text{project: d1-p1}), \text{resource}(\text{type: task, project: d1-p1}), \text{setCost}, \text{permit} \rangle$

B.2 Rules Augmentation Using Domain-Based Restrictions

Assume that we decide to analyze the authorization domain by grouping users based on the “role” user attribute. As shown in the top part of Fig. 11, the authorization domains of the user groups having distinct values for the “role” attribute are identified using the access requests examples of \mathcal{D} . These authorization domains allow one to recognize the set of operations authorized per user group. Thereafter, a set of negative authorizations are generated to restrict users having a specific role from performing specific operations on resources.

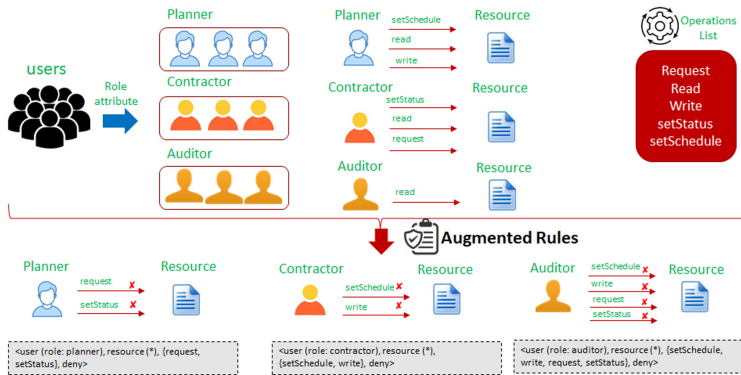


Fig. 11. Rules augmentation using domain-based restrictions

B.3 Rules Augmentation Using Machine Learning

Assume that \mathcal{D} includes a decision example (l_i) for a user (id: “pl-1”) accessing a resource (id: “sc-1”) where both of them belong to a department “d1”. Assuming Polisma generated a rule based on l_i as follows: $\rho_i = \langle \text{user}(\text{role: planner, department: d1}), \text{resource}(\text{type: schedule, department: d1}), \text{read, permit} \rangle$. Such a rule cannot control a new request by a user (id: “pl-5”) for accessing a resource (id: “sc-5”) where both of them belong to another department “d5”. Such a is similar to l_i . Therefore, a prediction-based approach is required to enable generating another rule.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: ACM SIGMOD Record, vol. 22, pp. 207–216. ACM (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. VLDB **1215**, 487–499 (1994)
3. Bertino, E., Ghinita, G., Kamra, A.: Access control for databases: concepts and systems. Found. Trends® Databases **3**(1–2), 1–148 (2011)
4. Cappelletti, L., Valtolina, S., Valentini, G., Mesiti, M., Bertino, E.: On the quality of classification models for inferring ABAC policies from access logs. In: Big Data, pp. 4000–4007. IEEE (2019)
5. Cotrini, C., Weghorn, T., Basin, D.: Mining ABAC rules from sparse logs. In: EuroS&P, pp. 31–46. IEEE (2018)
6. De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. In: IJCAI (2015)
7. Hu, V., et al.: Guide to attribute based access control (ABAC) definition and considerations (2017). <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf>
8. Karimi, L., Joshi, J.: An unsupervised learning based approach for mining attribute based access control policies. In: Big Data, pp. 1427–1436. IEEE (2018)

9. Kavšek, B., Lavrač, N.: APRIORI-SD: adapting association rule learning to sub-group discovery. *Appl. Artif. Intell.* **20**(7), 543–583 (2006)
10. Kohavi, R., Sommerfield, D.: Feature subset selection using the wrapper method: overfitting and dynamic search space topology. In: *KDD*, pp. 192–197 (1995)
11. Krejcie, R.V., Morgan, D.W.: Determining sample size for research activities. *Educ. Psychol. Measur.* **30**(3), 607–610 (1970)
12. Law, M., Russo, A., Elisa, B., Krysia, B., Jorge, L.: Representing and learning grammars in answer set programming. In: *AAAI* (2019)
13. Maxion, R.A., Reeder, R.W.: Improving user-interface dependability through mitigation of human error. *Int. J. Hum.-Comput. Stud.* **63**(1–2), 25–50 (2005)
14. Medvet, E., Bartoli, A., Carminati, B., Ferrari, E.: Evolutionary inference of attribute-based access control policies. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C.C. (eds.) *EMO 2015. LNCS*, vol. 9018, pp. 351–365. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15934-8_24
15. Mocanu, D., Turkmen, F., Liotta, A.: Towards ABAC policy mining from logs with deep learning. In: *IS*, pp. 124–128 (2015)
16. Molloy, I., et al.: Mining roles with semantic meanings. In: *SACMAT*, pp. 21–30. ACM (2008)
17. Ni, Q., Lobo, J., Calo, S., Rohatgi, P., Bertino, E.: Automating role-based provisioning by learning from examples. In: *SACMAT*, pp. 75–84. ACM (2009)
18. AuthZForce. <https://authzforce.ow2.org/>
19. Balana. <https://github.com/wso2/balana>
20. OASIS eXtensible Access Control Markup Language (XACML) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
21. Rabitti, F., Bertino, E., Kim, W., Woelk, D.: A model of authorization for next-generation database systems. *TODS* **16**(1), 88–131 (1991)
22. Sadeh, N., et al.: Understanding and capturing people’s privacy policies in a mobile social networking application. *Pers. Ubiquitous Comput.* **13**(6), 401–412 (2009)
23. Xu, Z., Stoller, S.D.: Algorithms for mining meaningful roles. In: *SACMAT*, pp. 57–66. ACM (2012)
24. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies from logs. In: Atluri, V., Pernul, G. (eds.) *DBSec 2014. LNCS*, vol. 8566, pp. 276–291. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43936-4_18