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

Amani Abu Jabal*, Elisa Bertino*, Jorge Lobo†, Mark Law‡, Alessandra Russo‡, Seraphin Calo§, Dinesh Verma§
* *Purdue University*, *West Lafayette*, *IN*, *USA*,
† *ICREA - Universitat Pompeo Fabra Spain*
‡ *Imperial College*, *London*, *UK*
§ *IBM TJ Watson Research Center*, *Yorktown Heights*, *NY*, *USA*
⟨ aabujaba, bertino⟩@purdue.edu, jorge.lobo@upf.edu, ⟨mark.law09, a.russo⟩@imperial.ac.uk, ⟨dverma, scalo⟩@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. *Polisma* capitalizes 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.**

***Index Terms*—Authorization Rules, Policy Mining, Policy Generalization**

I. Introduction

Most modern access control systems are based on the attribute-based access control model (ABAC) [4]. In ABAC, user requests to protected resources are granted or denied, based on discretionary attributes of the users, the resources, and the environmental conditions [6]. ABAC has several advantages. It allows one to specify access control policies in terms of domain-meaningful properties of users, protected objects, 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/object relationships underlying the rule sets [6]. As a result, access control decisions automatically adapt to changes in environments, and in user and object 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 [11]. There are several XACML enforcement engines, some of which are publicly available. 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 [9], [13]. 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.

To learn access control policies, suitable data could be access requests and corresponding access control responses (i.e., *access control decisions*) that are often collected in different way and form, depending on the real-world situation. For example, an organization may log past decisions taken by human administrators, 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 algorithm. But, learned ABAC control policy must also satisfy key main requirements. They must be expressed in terms of Boolean combinations of conditions against attributes of users, resources, and environments. They must also 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 [7] 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 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 policy sets 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*[1] to indicate that our ABAC **po**licy **l**earner uses m**i**ning, **s**tatistical, and **ma**chine learning techniques.

Polisma consists of four steps. In the first step, a data mining technique is used to infer associations between the 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.

The paper is organized as follows. Section II provides background information on ABAC policies, and introduces several definitions underlying the problem of learning ABAC policies from decision examples. Section III describes the proposed learning framework, and Section IV presents experimental results. Section **??** discusses related work, and Section V outlines conclusions and future work.

II. 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.

A. 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.* [15]:

Definition 1 (cf. ABAC Model [15])**.** 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 denoted by $V_R(a)$.
- 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, resources can be selected 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 e_U is a user attribute expression, e_R is a resource attribute expression, $O \subseteq \mathcal{O}$ is a set of operations, and d is the decision of the rule ($d \in \{permit, deny\}$).

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, an access request $\langle u, r, o\rangle$ for which there exist at least one rule $\rho = \langle e_U, e_R, O\rangle$ in $\mathcal{P}$ such that u satisfies e_U, r satisfies e_R, and $o \in O$, is considered to be accepted. Otherwise, the access request is assumed to be denied. However, negative authorizations are useful when dealing with large sets of resources organized according to hierarchies. Signed authorizations have been widely investigated [12], and also introduced in access control systems of commercial products (e.g., the access control model of SQL Servers provides negative authorizations by means of the DENY authorization command [1]), and in the XACML standard. Signed rules can be inconsistent, i.e., a user request can be both permitted and denied for the same access request.

B. 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 (see Definition 2).

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 $u \in e_u$, and $r \in e_r$, and the other arguments are interpreted as in an access control decision.

There exists an unknown set $\mathcal{F}$, of all access control decisions that should be allowed in the system, but for which we

[1]In the Tamil language, "polisma" means "inspector".

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 decision $\langle u, r, o, d\rangle$, denoted as $\langle u, r, o, d\rangle \in \mathcal{P}$ [2], iff there is a rule $\rho \in \mathcal{P}$ that satisfies the access request $\langle u, r, o\rangle$ with decision d. $\mathcal{P}$ can be seen as the set of all access decisions that it can control and hence, be compared directly with $\mathcal{F}$ - decisions controlled by $\mathcal{P}$ are decisions that belong to $\mathcal{F}$.

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

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

C. 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, $\mathcal{T} = \{t_1, \ldots, t_n\}$, 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 to be deployed. More precisely:

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, $\mathcal{T} = \{t_1, \ldots, t_n\}$, $LAPEC$ aims at generating a set of ABAC rules (i.e., $\mathcal{P} = \{\rho_1 \ldots \rho_m\}$) that are able to control all access requests in $\mathcal{F}$.

D. Policy Generation Assessment

ABAC policies generated by $LAPEC$ are assessed by verifying the satisfaction 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 some 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.

[2]We acknowledge here some abuse of notation for the $\in \mathcal{P}$ relation.

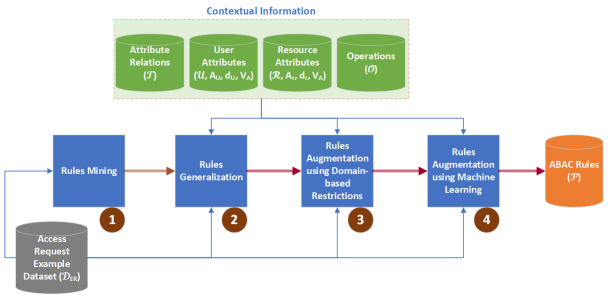

Fig. 1: The Architecture of *Polisma*

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

- $\forall l_i \in \mathcal{P} \rightarrow l_i \in \mathcal{D}$.

Definition 6 (Completeness)**.** A set of ABAC $\mathcal{P}$ is complete with respect to a consistent set of access control decisions $\mathcal{D}$ if and only if

- $\forall l_i \in \mathcal{D} \rightarrow l_i \in \mathcal{P}$.

Precision, recall, F1 score, and accuracy are calculated to measure the correctness of the decisions made by ABAC policies. Completeness is measured by calculating the percentage of access requests that are controlled by the set of policies (referred to as the *percentage of controlled requests* (*PCR*))[3].

III. The Learning Framework

Our framework comprises four steps (see Fig. 1) described in what follows.

A. Rules Mining

$\mathcal{D}$ provides a source of examples concerning user accesses to the available resources in an organization. In this step, we use association rule mining (ARM) [2] to analyze the association between users and resources. Such an association is used to filter out inconsistent decision examples.[4] Thus, rules having high association metrics (i.e., support and confidence scores) are kept to generate rules. $Polisma$ uses Apriori [3] (one of the ARM algorithms) to generate rules using $\mathcal{D}$. It is worth noting that rule mining can be performed by other techniques such as symbolic learning [8] [10].

[3]In this paper, we assume that the access requests that are not controlled by the set of policies are denied by default.

[4]Handling of other types of noise is important but outside the scope of this paper.

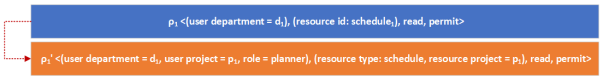

Fig. 2: Ground Rule vs. Safely-generalized Rule: The top rule is an example of rule generated from the first step while the below rule is generated from the second step

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 or unsafely-generalized. Overfitted rules impact completeness while unsafely-generalized rules impact correctness. Therefore, *Polisma* post-processes the ground rules in the next step.

B. 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 $\mathcal{D}$ or external context information sources, or both. In particular, each rule is post-processed 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. Fig. 2 shows an example of a ground rule that is overfitted from the perspective of the resource and unsafely-generalized from the perspective of the user, and an example of a rule generalized from it.

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 ($\mathcal{T}$) are 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.

1) 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 (see Definition 7) is estimated by the proportion of the sizes of two subsets:

a) the subset of decision 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 (see Definition 8). Thereafter, the attribute expression with the highest safety level is selected to be used for post-processing the ground rule of interest.

Definition 7 (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 $U_C = \{u_k \in \mathcal{U} \mid d_U(u_i, a_j) = d_U(u_k, a_j)\}$.

Definition 8 (Resource Attribute Expression Weight $W_{rav}(r_i, a_j, d_R(r_i, a_j), \mathcal{D})$)**.** For a ground rule ρ and its corresponding resource $r_i \in R$, the weight of a resource attribute expression $\langle a_j, d_R(r_i, a_j)\rangle$ is defined by the formula

$$W_{rav} = \frac{|\ R_D\ |}{|\ R_C\ |}$$

where $R_D = \{l \in \mathcal{D} \mid d_R(r_i, a_j) = d_R(l.r, a_j) \wedge l.u \in \rho.e_U \wedge l.o \in \rho.O \wedge \rho.d = l.d\}$ and $R_C = \{r_k \in \mathcal{R} \mid d_R(r_i, a_j) = d_R(r_k, a_j)\}$.

Different strategies for selecting attribute expressions are possible. They can be based on a single or a combination of attributes, and they can consider either user attributes, resource attributes, or both of them. The 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).

2) Structure-based Strategy (SS).: The BS strategy works without prior knowledge of $\mathcal{T}$. When $\mathcal{T}$ is available, it can

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.

```
Define w_max = -∞, a_selected = ⊥.
for a_i ∈ A_U do
    w_i = W_uav(u_i, a_i, d_U(u_i, a_i), D)
    if w_i > w_max then
        a_selected = a_i, w_max = w_ai
    end if
end for
e_U ← ⟨ a_selected, d_U(u_i, a_selected) ⟩
Create Rule ρ'_i = ⟨e_U, ρ_i.e_R, ρ_i.O, ρ_i.d⟩
return ρ'_i
```

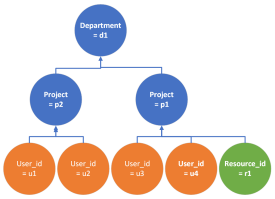

Fig. 3: Example of Attribute-relationship Graph G

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: $\langle x \in A_U, y \in A_R \rangle$ = First-Common-Attribute-Value(g_i)
3: $e_U \leftarrow \langle\ x,\ d_U(u_i,\ x)\ \rangle$
4: $e_R \leftarrow \langle\ y,\ d_R(r_i,\ y)\ \rangle$
5: Create Rule $\rho_i' = \langle e_U, e_R, \rho_i.O, \rho_i.d \rangle$
6: **return** ρ_i'

be analyzed to gather information about "hidden" correlations between common attributes of a user and a resource. Such attributes can potentially benefit 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 least level in the hierarchy to the highest one enables finding 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 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).

Example: *Assume that a ground rule ρ = ⟨(user department = d1), (resource id = r1), read, permit ⟩ is given. Also, Polisma is given the following information:*

- A_U = {*user id, project, department*} *and* A_R = {*resource id, project, department*},
- *The list of users U' satisfying $\rho.e_U$ have two values for the project attribute (i.e., p1, p2) while the resource R' satisfying $\rho.e_R$ belong to the project p1 and the department d1, and*
- $\mathcal{T} = (p1, d1)$.

G is constructed as shown in Fig. 3. Using G, the two common attributes for U' and R' are p1 and d1 and the first common attribute is p1. Therefore, the generated rule is ρ = ⟨(user department = d1, user project = p1), (resource project = p1), read, permit ⟩.

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

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

As discussed earlier, the first step generates ground rules that are either overfitted or unsafely-generalized. This step post-processes unsafely-generalized rules into safely-generalized ones; hence, improving correctness, and 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.

C. Rules Augmentation using Domain-based Restrictions

"Safe" generalization of ground rules is one approach towards fulfilling the 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*). Such restrictions potentially increase safety by avoiding erroneous accesses.

Example: *Given that from the examples we gather that a) the range of the role attribute in A_U is {employee, contractor, auditor}, b) $\mathcal{O}$ = {read, write, approve, setStatus, setCost}, and c) users of the auditor role use the list of actions {read}, the following restriction rule can be generated ρ = ⟨(user role = auditor), (resource: *), {write, approve, setStatus, setCost}, deny ⟩.*

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. Identifying such groups requires prior knowledge or a pre-processing of A_U and A_R. Therefore, this step includes two strategies. One strategy assumes prior knowledge about preferable attributes to partition the user and resource sets. The other strategy searches heuristically for such attributes. We describe these strategies in what follows.

1) Semantic Strategy (SS).: A_U and A_R might be tagged with some semantic metadata. Such semantics can be descriptive metadata about the distinctive attributes (referred to as *preferable attributes*) which can be used for grouping users and resources. For example, the "role" attribute of users and the "type" attribute of resources can be reasonably used for creating appropriate groups because these two attributes are often used for defining access control policies.

Given the preferable attributes, the semantic strategy constructs user and resource groups as described in Definitions 10 and 11, respectively. 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 three types of authorization domains: operations performed by each user group (lines 6-7), operations performed on each resource group (lines 8-9), and users' accesses to each resource group (lines 8,10). Subsequently, the algorithm augments restrictions based on these access domains (lines 12-14).

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 $G_U^{a_i}$

Algorithm 3 Rules Augmentation using Domain-based Restrictions

Require: $\mathcal{D}$, $\mathcal{U}$, $\mathcal{R}$, x: preferable attribute of users, y: 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: $\forall\ g_i \in G_U,\ O^u_{g_i} \rightarrow \{\}$
3: $\forall\ g_i \in G_R,\ O^r_{g_i} \rightarrow \{\}$
4: $\forall\ g_i \in G_R,\ U^r_{g_i} \rightarrow \{\}$
5: **for** $l_i \in \mathcal{D}$ **do**
6: $\quad g' \leftarrow g_i \in G_U \mid l_i.u \in g_i \wedge l_i.d = Permit$
7: $\quad O^u_{g'} \rightarrow O^u_{g'} \cup l_i.o$
8: $\quad g'' \leftarrow g_i \in G_R \mid l_i.r \in g_i \wedge l_i.d = Permit$
9: $\quad O^r_{g''} \rightarrow O^r_{g''} \cup l_i.o$
10: $\quad U^r_{g''} \rightarrow U^r_{g''} \cup l_i.u$
11: **end for**
12: $\forall\ g_i \in G_U$, Create Rule $\rho_i = \langle\langle x, d_U(u_i, x)\rangle, *, o_i, Deny\rangle \mid u_i \in g_i \wedge o_i \in (\mathcal{O} \setminus O^u_{g_i})$
13: $\forall\ g_i \in G_R$, Create Rule $\rho_i = \langle *, \langle y, d_R(r_i, y)\rangle, o_i, Deny\rangle \mid r_i \in g_i \wedge o_i \in (\mathcal{O} \setminus O^r_{g_i})$
14: $\forall\ g_i \in G_R$, Create Rule $\rho_i = \langle\langle x, d_U(u_i, x)\rangle, \langle y, d_R(r_i, y)\rangle, *, Deny\rangle \mid r_i \in g_i \wedge u_i \in (\mathcal{U} \setminus U^r_{g_i})$

$\{g_1^{a_i}, \ldots, g_k^{a_i}\}$ where

$$(g_i^{a_i} = \{u_1, \ldots, u_m\} \mid \forall u', u'' \in g_i, d_U(u', a_i) = d_U(u'', a_i),$$
$$m \leq \mid \mathcal{U} \mid) \wedge (g_i \cap g_j = \phi \mid i \neq j) \wedge (k = \mid V_U(a_i) \mid).$$

Definition 11 (Attribute-based Resource Groups $G_R^{a_i}$)**.** Given $\mathcal{R}$ and $a_i \in A_R$, $\mathcal{R}$ is divided into is a set of resource groups $G_R^{a_i}$ $\{g_1^{a_i}, \ldots, g_k^{a_i}\}$ where

$$(g_i^{a_i} = \{r_1, \ldots, r_m\} \mid \forall r', r'' \in g_i, d_R(r', a_i) = d_R(r'', a_i)$$
$$, m \leq \mid \mathcal{R} \mid) \wedge (g_i \cap g_j = \phi \mid i \neq j) \wedge (k = \mid V_R(a_i) \mid).$$

2) Heuristic Strategy (HS).: When the preferable attributes are not given, this strategy first predicts heuristically the best attribute for user and resource partitions. Thereafter, the remaining steps of the strategy follow Algorithm 3.

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 enables measuring the ability of the attribute of interest for creating even partitions[5]. One method for capturing the attribute distribution is to compute the attribute entropy as defined in Equation 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 preferable one.

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

[5]Even distribution tends to generate smaller groups in which each group potentially have 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

D. 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). For example, assume that $\mathcal{D}$ includes a decision example (l_i) for u_i accessing r_i where both of them belong to a department x. The generated rule based on l_i cannot control a new request (e.g., u_j requests access to r_j where both of them belong to another department y) which is similar to l_i. Therefore, a prediction-based approach is required.

A possible prediction approach is to use an ML classifier that builds a training model based on the attributes provided by $\mathcal{D}$ and context information. The generated model implicitly creates patterns of accesses and their decisions. Thus, the model will be capable of predicting the decision of any access request based on its similarity to the ones controlled by the rules generated from the previous steps. Thereafter, this step creates new rules based on these predictions and generalizes them. 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.

IV. Evaluation

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

A. Experimental Methodology

Datasets. To evaluate the performance of *Polisma*, we conducted several experiments using two datasets: one is a synthetic dataset (referred to as *project management* (PM) [14]), and the other is a real one (referred to as *Amazon* (AZ)[6]) (see Table I for statistics about these datasets). In addition to decision examples, PM is tagged with context information (e.g., attribute relationships and attribute semantics).

Naïve ML Approach. For comparison purposes, we

[6]http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples

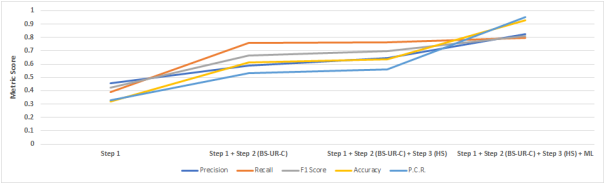

Fig. 4: Evaluation of *Polisma* using the brute-force strategy for Step 2 and the heuristics strategy for Step 3 (i.e., *Polisma-BS-HS-ML*) using the PM dataset.

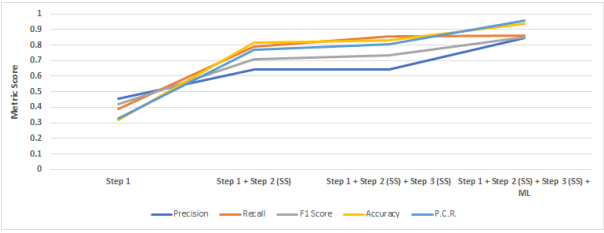

Fig. 5: Evaluation of *Polisma* using the structure-based strategy for Step 2 and the semantic strategy for Step 3 (i.e., *Polisma-SS-SS-ML*) using the PM dataset.

developed an approach which utilizes an ML classifier to create rules. In particular, 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]

Evaluation and Settings. *Polisma* is evaluated in two ways. First, to evaluate how rules are generalized or constrained throughout *Polisma* steps, *Polisma* uses a subset of $\mathcal{D}$ (i.e., $\mathcal{D}'$ where $\mathcal{D}' \subseteq \mathcal{D}$) to generate rules. The rules generated after each step are evaluated using the remaining subset of $\mathcal{D}$. We used 90% of $\mathcal{D}$ for learning ABAC rules and 10% of $\mathcal{D}$ for evaluation. Second, to compare *Polisma* with the naïve approach, we evaluated the generated rules based on $\mathcal{D}$ using $\mathcal{N}$ (for the PM dataset $\mid \mathcal{N} \mid = 200$ while for the AZ dataset $\mid \mathcal{N} \mid = 500$). Since both evaluations can be affected by the examples in $\mathcal{D}'$, for each dataset *Polisma* is evaluated 10 times while varying the random selection of $\mathcal{D}'$. We report the average of the evaluation metrics (precision, recall, F1 score, accuracy, and *PCR*). As ML classifiers used for the fourth step of Polisma and the naive approach, we used a combined classification technique based on majority voting where the underlying classifiers are Random Forest and kNN.

B. Experimental Results

1) Steps Evaluation.: Figs. 4, 5, and 6 show 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

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

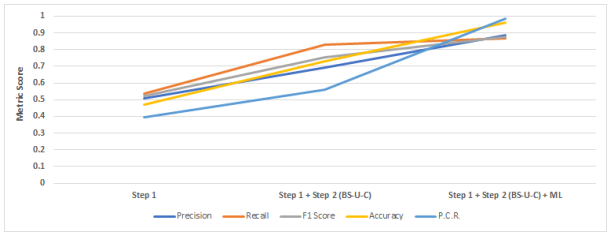

Fig. 6: Evaluation of *Polisma* using the brute-force strategy for Step 2 (i.e., *Polisma-BS-ML*) using the AZ dataset.

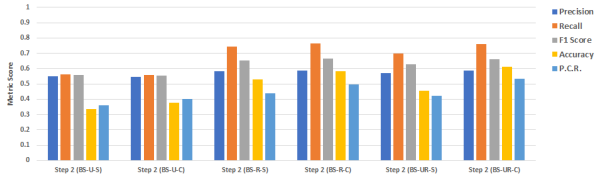

Fig. 7: Comparison between the variants of the brute-force strategy (second step) using the PM dataset.

when using different strategies in each step. As the results in Figs. 4 and 5 show, the strategies that exploit additional knowledge about user and resource attributes (e.g., $\mathcal{T}$) have better results 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 which 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 many of the issued access requests are not allowed by the authorizations in the access control system. In the case of the AZ dataset (Fig. 6), because some of the required context information is not available, only the BS strategy has been used in the experiments. Nonetheless, the policies learned by *Polisma* with BS have good results in terms of correctness and completeness. In particular, the F1 score reached up to 0.86, while the *PCR* reached up to 0.98.

2) Variants of the Brute-force Strategy.: As the results in Fig. 7 show, using resource attributes for rules generalization is better than using the user attributes. The reason is that the number of attributes for resources is higher than that of users in the PM dataset. Thus, the selection space of attribute expressions is expanded; hence it potentially increases the possibility of finding the best attribute expression for safe generalization. In particular, using resources attributes achieves 20% and 24% improvement compared to that of using users attributes in term of F1 score and *PCR*, respectively. Similarly, performing the generalization using both user and resource attributes is better that of using only user attribute or resource attributes. In particular, BS-UR-C showed a significant improvement compared to BS-U-C (20% for F1 score, and 32% for *PCR*). 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.

TABLE I: Overview of Datasets

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

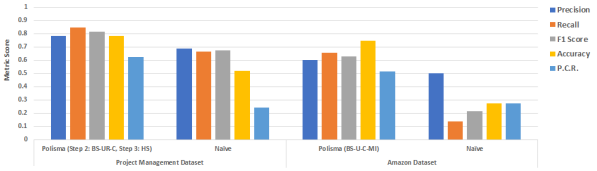

Fig. 8: Comparison between *Polisma* and the Naïve approach using the PM and AZ datasets.

3) Naïve Approach vs. Polisma.: The results in Fig. 8 show that *Polisma* achieved better results compared to the naïve approach. For example, on the PM dataset, the F1 score (*PCR*) of *Polisma* improves by a factor of 1.2 (2.6) compared to that of the naïve. On the AZ dataset, *Polisma*'s F1 score (*PCR*) improves by a factor of 2.9 (1.9) compared to the naïve. Both *Polisma* and the naïve approach use an ML classifier. However, *Polisma* uses the ML classifier for generating only some of the rules. This implies two observations: a) using multiple techniques (i.e., data mining, statistical, and ML techniques) enables *Polisma* to learn policies better; b) among the two 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 policy learning process.

V. 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 employing various techniques, namely data mining, statistical, and machine learning. Our evaluations, carried out on a real-world decision log (referred to as AZ) and on a synthetic one (referred to as PM), show that *Polisma* is able to learn policies that are both complete and correct. The rules generated by *Polisma* using the PM dataset achieved an F1 score of 0.85 and *PCR* of 0.95; also, when using the AZ dataset, the generated rules achieved an F1 score of 0.86 and *PCR* of 0.98. As part of future work, we plan to extend our framework by integrating an inductive learner [8] and a probabilistic learner [5]. Furthermore, we plan to extend our framework to support policy transferability and policy learning explainability.

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 partially supported by the Spanish Ministry of Economy and Competitiveness under Grant Numbers: TIN201681032P, MDM20150502.

References

[1] (2017) Authorization and permissions in SQL Server. [Last Accessed: April. 27, 2019]. [Online]. Available: https://msdn.microsoft.com/en-us/library/bb669084(v=vs.110).aspx

[2] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[3] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *VLDB*, vol. 1215, 1994, pp. 487–499.

[4] E. Bertino, G. Ghinita, and A. Kamra, "Access control for databases: Concepts and systems," *Foundations and Trends® in Databases*, vol. 3, no. 1–2, pp. 1–148, 2011.

[5] L. De Raedt, A. Dries, I. Thon, G. Van den Broeck, and M. Verbeke, "Inducing probabilistic relational rules from probabilistic examples," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[6] V. Hu, D. Ferraiolo, K. Rick, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (abac) definition and considerations, 2017. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf

[7] R. Kohavi and D. Sommerfield, "Feature subset selection using the wrapper method: Overfitting and dynamic search space topology." in *KDD*, 1995, pp. 192–197.

[8] M. Law, A. Russo, B. Elisa, B. Krysia, and L. Jorge, "Representing and learning grammars in answer set programming," in *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

[9] R. A. Maxion and R. W. Reeder, "Improving user-interface dependability through mitigation of human error," *International Journal of Human-Computer Studies*, vol. 63, no. 1-2, pp. 25–50, 2005.

[10] S. Muggleton and J. Firth, "Relational rule induction with cprogol 4.4: A tutorial introduction," in *Relational data mining*. Springer, 2001, pp. 160–188.

[11] OASIS eXtensible Access Control Markup Language (XACML) TC. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[12] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems," *TODS*, vol. 16, no. 1, pp. 88–131, 1991.

[13] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and capturing people's privacy policies in a mobile social networking application," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 401–412, 2009.

[14] Z. Xu and S. D. Stoller, "Mining attribute-based access control policies from logs," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2014, pp. 276–291.

[15] ——, "Mining attribute-based access control policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 533–545, 2015.