ProFact: A Provenance-based Analytics Framework for Access Control Policies

Amani Abu Jabal, Maryam Davari, Elisa Bertino, Christian Makaya, Seraphin Calo, Dinesh Verma, Christopher Williams

Abstract—Policy-based access control systems are crucial for secure information sharing in collaborative applications. However, policy management needs to be flexible in order to adapt to different environments and be able to support policy evolution. However, when dealing with large sets of evolving policies, it is critical that policies meet certain *policy quality requirements*. Policy sets must be complete, free of inconsistencies, and relevant. In this paper, we propose a framework to analyze policies to determine whether they meet such requirements. Our framework uses provenance techniques to collect comprehensive data about actions which were either triggered due to a network context or a user (i.e., a human or a device) action. The framework includes two approaches for policy analysis: structure-based and classification-based. For the structure-based approach, we designed tree structures to organize and assess the policy set efficiently. For the classification-based approach, we employed the classification techniques to learn the characteristics of policies and predict their quality. In addition, the framework includes the policy evolution module which mainly consists of recommendation and re-evaluation services for policy changes which both aim at fulfilling the policy quality requirements. The analysis framework has been implemented and experimental results from the prototype are reported.

Index Terms—Access Control Policies, Provenance, Policy Analysis, Policy Quality, Policy Tree, Structure-based Policy Analysis, Classification-based Policy Analysis.

1 Introduction

Advances of technology in areas such as sensors, IoT, and robotics enable new collaborative applications. Such applications involve not only humans but also autonomous devices (e.g., drones, robots) [18]. A key requirement for such collaborations is represented by secure information sharing and information flow protection. Access control is a primary mechanism for selectively controlling accesses to a set of protected information.

An access control system decides, based on a set of access control policies, whether a subject (e.g., user, process, device, application) can access a specific information resource (e.g., files) for performing a certain action (e.g., read, write). A large number of research efforts have been devoted to defining access control models (see [15] for a survey) including RBAC [47] and XACML [1].

For an access control system to be effective and efficient, it is critical that policies be of "good quality" in order to make sure that the appropriate access control decisions are taken. Towards this, Bertino et al. [12] introduced a set of quality requirements and proposed a framework for policy analysis that assesses the quality of policy sets. The framework uses two data structures for policy assessment based on such quality requirements. The data structures maintain information about both policies and actions executed in the systems (referred to as *transactions*) in *Policy Tree* and *Transaction Tree*, respectively. Executed transactions can be monitored by a provenance system which provides fine-grained historical details about accesses to the protected resources. The framework supports two analysis strategies for analyzing the policies independently from the transactions (referred to as *policy-based analysis*) and the policies with respect to the actual transactions executed by the subjects (referred to as *transaction-based analysis*).

This paper extends our previous work [12] along three major directions. The first is the introduction of a tightly-combined structure referred to as *Policy-and-Transaction Tree* to store both access control policies and their corresponding transactions, along with an additional analysis strategy based on this structure. The second is the introduction of an additional policy analysis method based on classification techniques. The third is the automatic support of policy evolutions based on the results of policy analysis. The overall novel framework, which we refer to *ProFact* (standing for *Provenance-based Analytics Framework for Access Control Policies*), has been implemented and experimental results are reported from the implemented prototype.

Our contributions include:

- The implementation of the data structures discussed in [12] and their corresponding policy analysis approaches.
- The design and implementation of the *Policy-and-Transaction Tree* data structure and its corresponding policy analysis approach. Furthermore, an asymptotic analysis of the policy analysis approaches based on data structures is investigated.

- *Amani Abu Jabal, Maryam Davari, and Elisa Bertino are with the Department of Computer Science, Purdue University, West Lafayette, Indiana, USA, 47907.*
  E-mail: ⟨aabujaba, davari, bertino@purdue.edu⟩
- *Christian Makaya, Seraphin Calo, and Dinesh Verma are with IBM Research, Yorktown Heights, NY, USA, 10598.*
  E-mail: ⟨cmakaya, scalo, dverma@us.ibm.com⟩
- *Christopher Williams is with The Defence Science and Technology Laboratory, Porton Down, Wiltshire SP4 0JQ, UK.*
  E-mail: cwilliams@dstl.gov.uk

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

- The design and implementation of a policy analysis approach that utilizes a classification technique. This approach efficiently detects "low quality" policies based on a pre-trained model which learns the patterns of "low quality" policies obtained from the historical analysis results. The approach employs various types of classifiers to learn the policy patterns. This approach is referred to as *classification-based analysis*. For implementing the classification-based approach, we designed two variants: a single classifier (i.e., kNN, Naïve Bayes, SVM, Random Forest, or Decision Tree) and combined classifier (i.e., combining the results of multiple classifiers using probabilities of prediction votes or a majority of votes).
- An experimental comparison of the different policy analysis approaches.
- The design and implementation of a policy evolution approach. Based on the policy analysis, our evolution tool handles modifications to "low quality" policies and re-analyzes them.

While implementing the framework, we were not able to obtain a large set of access control policies from a real system. Therefore, we generated a synthesized dataset for experimental purposes. Moreover, machine learning algorithms (particularly classification techniques) require sufficient dataset to efficiently learn the characteristics of the dataset. In our framework, we addressed the challenges of classification techniques at two levels: *data level* by sampling more data only in the learning phase, and *approach level* by devising a classification scheme that combines the classification results of multiple well-known classifiers.

The rest of the paper is organized as follows. Section 2 provides background information on access control and data provenance. Section 3 introduces several definitions underlying the policy quality requirements and proposes a new data structure for policy analysis. Section 4 describes our analysis services. Section 5 describes our policy evolution services. Section 6 outlines the infrastructure of the proposed framework. Section 7 presents the experimental results. Section 8 discusses related work. Finally, Section 9 outlines conclusions and future work.

2 Preliminaries

In what follows, based on [12], we introduce background concepts and information needed for the subsequent developments in the paper.

2.1 Role-based Access Control

The role-based access control (RBAC model) definition consists of four basic components [47]: *users*, *roles*, *permissions*, and *sessions*. Moreover, it includes several functions. The user assignment (UA) function specifies which user is assigned which roles, whereas the permission assignment (PA) function specifies the set of permissions assigned to each role. The user function maps each session to a single user, while the role function assigns a session to a set of roles (i.e., the roles that are activated by the corresponding user in that session). The following definition (adapted from Sandhu et al. [47]) formally defines the RBAC model.

Definition 1 (Role-based Access Control Model [13])**.** The model consists of the following components.

- U, R, P, S refer to the set of users, roles, permissions, and sessions, respectively.
- A permission $p_i \in P$ is a tuple of three components consisting of an object $o_j \in O$, an action $a \in A$, and a sign $g \in \{+, -\}$.
- PA is the permission assignment function that assigns permissions to roles (i.e., $PA \subseteq R \times P$ and $PA(r_i) \subseteq P, \forall r_i \in R$).
- UA is the user assignment function that assigns users to roles (i.e., $UA \subseteq U \times R$ and $UA(u_i) \subseteq R, \forall u_i \in U$).
- The $user$ function assigns a session to a single user (i.e., $user : S \longrightarrow U \mid user(s_i) \in U$).
- The $roles$ function assigns a session to the roles associated with the user activated the corresponding session (i.e., $roles \subseteq S \times 2^R \mid roles(s_i) = \{r \mid (user(s_i), r) \subseteq UA\}$).
- RH is the role hierarchy function (i.e., $RH \subseteq R \times R$), which refers to the partially ordered role hierarchy (written $\geq$).

In Definition 1, we associate a "sign" with each permission to support positive and negative authorizations. Negative authorizations are particularly useful when dealing with large sets of protected objects organized according to hierarchies, as well as in our contexts where negative authorizations are also critical in order to provide boundaries to actions that cognitive autonomous devices can execute.

Throughout the discussion, we will refer to an access control policy as acp. The components composing a policy are a role ($acp.r$), and a permission ($acp.p$). The permission of a policy consists of an object ($acp.p.o$), an action ($acp.p.a$), and a sign ($acp.p.sign$).

2.2 Data Provenance

Data provenance is a historical record of a data object, which includes its preceding data objects, activities, and context leading to produce its current state. Several provenance models have been proposed (e.g., Open Provenance Model (OPM) [41], PROV [2], and Secure Interoperable Multi-Granular Provenance (SimP) [3]). In this paper, we used SimP [3]. However, it is possible to utilize other provenance models[1]. As shown in Fig. 1, SimP represents provenance by a set of entities: *data*, *processes*, *operations*, *communications*, *actors*, *environments*, and *access control policies*. A *process* manipulates data objects by performing a sequence of operations to generate other data objects. The *operations* in the same process or in two different processes interact and such interaction is referred to as *communication*. Processes (including its operations) and data are manipulated by *actors* which can be humans or devices. Processes also have a context that affects their execution and output. Such context is represented by the *environment* which refers to a set of parameters, and system configurations. In addition, the

1. Integrating our framework with different provenance models is outside the scope of the paper

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

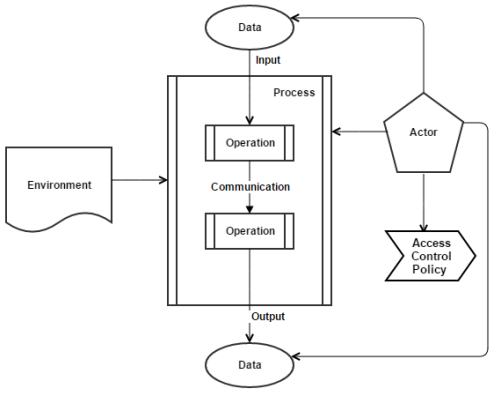


Fig. 1: The SimP provenance model [3]

TABLE 1: SimP Entity Notations

Notation	Description
SimP.Processes	The set of processes captured by SimP provenance
SimP.Policies	The set of access control policies captured by SimP
PR	A process record (i.e., $PR \in SimP.Processes$)
PL	An access control policy record (i.e., $PL \in SimP.Policies$)
PR.Operation	The operation executed by the PR process
PR.Data	The data object manipulated by the PR process
PR.User	The user who executed the PR process
PL.Operation	The operation controlled by the PL policy
PL.Data	The data object controlled by the PL policy
PL.User	The user whose access is controlled by the PL policy

access control policy entity captures the policies of actors at the time of data manipulation by the actors.

In SimP, the access control policy entity includes the following attributes: operation, data, and user. A process is represented by a set of attributes including operations, data, and user.

Table 1 defines notations for SimP entities. Moreover, each process record refers to a task in a workflow. A task represents a transaction executed at run-time. We formally define a transaction as follows:

Definition 2 (Transaction)**.** A transaction $t \in T$ is an action $a \in A$ executed by a user $u \in U$ on an object $o \in O$. Thus, a transaction t is represented by a tuple (u, o, a).

3 Policy Analysis Metrics And Structures

In this section, we introduce the quality requirements that are used as metrics to evaluate policies through the analysis process. We also describe two data structures that are utilized for the policy analysis services.

3.1 Policy Quality Requirements

We illustrate the policy quality requirements by the following running example (adapted from [52]).

Example: *We envision an automated delivery management system for army forces operating in a set of bunkers which are supported by a remote supply depot. Autonomous vehicles (mules) are used to transfer supplies (e.g., meals ready to eat (MREs)) from the depot to the bunkers. Supplies in all sites, including the bunkers and the depot, are managed by robotic devices. Smart refrigerators at bunkers manage the MRE inventory and notify the robot at the depot when additional supplies are needed. At the depot, there are several robots and mules. The mules transfer supplies from the depot to bunkers. There are two types of robot. One is the depot manager which receives notifications from smart refrigerators at bunkers and manages the transfer of the required supplies to bunkers; the other type is the worker that is responsible for loading the mules with supply cartoons. In addition, there is a computer system that maintains a central database for sharing necessary information (e.g., mule location and status, depot supply, robot status).*

In such a delivery management system, we focus on designing an access control system for the robots working at the depot. Because of the two types of robot, we have two corresponding roles: manager and worker. The worker executes the following types of transaction: (i) receive loading requests; (ii) load a mule with the supplies; and (iii) report its status to the computer system. The manager is authorized to perform the same types of transaction as the worker and in addition is authorized to perform the following types of transaction: (i) receive notification from a bunker; (ii) inquire whether a bunker needs supplies; (iii) check availability of supplies; (iv) retrieve the list of available mules and workers; (v) and assign a worker and mule for a delivery request. The access control policies related to these transactions are listed in Table 2. The problem of assuring the quality of a set of access control policies can be restated as the problem of making sure that policies do not have inconsistency, are not redundant, irrelevant, and incomplete with respect to the actions executed by the users. In addition, it is critical to minimize the number of explicit exceptions that must be allowed with respect to the policies. Minimizing the exceptions is critical to reduce the manual administrative

TABLE 2: Example of Access Control Policies for the Depot Manager and Worker Roles for the Robots Working in a Delivery Management System

Policy	Role	Permission		
		Action	Object	Sign
acp_1	Manager	Receive	Notification from bunkers	+
acp_2		Receive	Notification from bunker 10	−
acp_3		Receive	Notification from bunker 5	+
acp_4		Inquire a Bunker	Bunker Status	+
acp_5		Inquire central DB	Supply Status	+
acp_6		Inquire central DB	List of available mules	+
acp_7		Inquire central DB	List of available workers	+
acp_8		Assign loading task	Worker, Mule, Supply	+
acp_9		Receive	Loading task	+
acp_{10}		Load	Supply, mule	+
acp_{11}		Report to central DB	Robot status	+
acp_{12}	Worker	Receive	Loading task	+
acp_{13}		Load	Supply, mule	+
acp_{14}		Report to central DB	Robot status	+
acp_{15}		Report to Manager	Robot status	−

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

activities to be executed in the system. In what follows we introduce several definitions underlying our policy quality notion.

Definition 3 (Inconsistency)**.** Access control policies acp_i, $acp_j \in ACP$ are inconsistent if and only if

- $acp_i.r = acp_j.r \wedge acp_i.p.o = acp_j.p.o \wedge acp_i.p.a = acp_j.p.a$
- $acp_i.p.sign \neq acp_j.p.sign$.

Inconsistency refers to the situation in which for the same access by the same role, one policy allows the access and the other denies it. Policy inconsistency leads to conflicts at the policy enforcement stage that then requires conflict resolution strategies [14] be applied. Minimizing the inconsistencies is thus critical to reduce the need for conflict resolutions activities.

Example: As shown in Table 2, acp_1 specifies that a robot with the manager role has a positive permission to receive notifications from all bunkers. However, based on acp_2 the role manager is forbidden from receiving notifications from the bunker with number 10. Hence, acp_1 is inconsistent with acp_2.

Definition 4 (Policies Exceptions)**.** A transaction $t_i \in T(u \in U, o \in O, a \in A)$ is an exception with respect to an access control policy $acp_j \in ACP$ if and only if

- $t_i.o = acp_j.p.o \wedge t_i.a = acp_j.p.a \wedge acp_j.r \in UA(t_i.u_i) \wedge acp_j.p.sign = \text{`}-\text{'}$
- $\exists PR_k \in SimP.Processes \mid PR_k.Operation = t_i.a \wedge PR_k.Data = t_i.o \wedge PR_k.User = t_i.u$.

A policy exception arises when a transaction is executed that violates a negative authorization. In general, whereas exceptions may arise due for example to unforeseen circumstances; it is important to minimize their occurrences. Exceptions may require explicit ad-hoc and temporary authorizations from human administrators, which can be expensive and not always possible. It is thus therefore critical to analyze exceptions to determine whether policies should be modified so to be able to cover the frequently occurring exceptions.

Example: Consider policy acp_{15} from Table 2. According to such a policy, a worker is not authorized to communicate with the manager about emergency situations. Suppose that one of the worker robots has a malfunction while performing a loading task. The worker robot should notify the manager about the situation to enable the manager to reassign the task to another worker. To solve this situation, the administrator allows the worker robot to notify the manager about the task by adding a temporary access control policy and disabling acp_{15}. However, it is clear that a modification of the policy allowing a robot to notify the manager in the case of malfunctioning would be a more efficient solution.

Definition 5 (Incompleteness)**.** A set of access control policies is incomplete if and only if

- $\exists t_i \in T \mid t_i = (u_i \in U, o_i \in O, a_i \in A$
- $\exists PR_k \in SimP.Processes \mid PR_k.Operation = t_i.a \wedge PR_k.Data = t_i.o \wedge PR_k.User = t_i.u$
- $\nexists acp \in ACP \mid t_i.o = acp.p.o \wedge t_i.a = acp.p.a \wedge acp.r \in UA(t_i.u_i)$.

Incompleteness refers to the situation in which an access request is issued that is not covered by the current policies.

Example: Consider the case in which a worker asks information about the capacity of a mule. In this case, there is no policy either allowing or denying the access to this information.

In cases like the one in the previous example, we say, as in the well-known XACML standard [1], that there is no applicable policy. Making sure that all actions are covered by some policies is critical to enhance the predictability of device behaviors.

Definition 6 (Redundancy)**.** An access control policy $acp_i \in ACP$ is redundant if and only if

- $\exists acp_j \in ACP$
- $acp_i.r = acp_j.r \wedge (acp_i.p.o \subseteq acp_j.p.o \vee acp_j.p.o \subseteq acp_i.p.o) \wedge acp_i.p.a = acp_j.p.a \wedge acp_i.p.sign = acp_j.p.sign$.

Redundancy arises when there is a set of similar policies that control the same situation of interest. Detecting redundancy helps in reducing the size of the policy set. In addition, it enhances security.

Example: Consider the policies in Table 2. Based on acp_1 and acp_3 a robot manager is authorized to receive notifications from Bunker 5. Hence, these two policies will be enforced. However it is clear that policy acp_3 is redundant with respect to acp_1 and as the latter is more general, the former can be removed.

Definition 7 (Irrelevancy)**.** An access control policy acp_i ACP is irrelevant if and only if

- $\nexists PR_k \in SimP.Processes \mid PR_k.Operation = acp_i.p, a \wedge PR_k.Data = acp_i.p.o \wedge acp_i.r \in UA(PR_k.User)$
- $\nexists PL_k \in SimP.Policies \mid PL_k.Operation = acp_i.p, a \wedge PL_k.Data = acp_i.p.o \wedge acp_i.r \in UA(PL_k.User)$.

Irrelevancy refers to the situation in which no access requests are issued to which a given policy is applicable. Removing irrelevant policies enhances security and enhances usability in cases in which human users have to inspect the policies, for example when solving policy conflicts.

Example: Consider the policies in Table 2. According to acp_4, the robot manager is able to inquire a bunker status. Nonetheless, such a policy is irrelevant as bunkers automatically send requests.

Removing irrelevant policies is critical when policies are not used, as irrelevant policies may undermine security. For example, an attacker may try to compromise a user in order to exploit the privileges of this user. Thus, making sure that a user does not have permissions for actions that the user is not expected to execute is critical to minimize such exploitations.

3.2 Structures for Policy Analysis

Policy analysis requires scanning the policy set to detect the policies which violate the policy quality requirements. Furthermore, assessing policy quality requires searching two types of data sources: the set of policies and the set of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

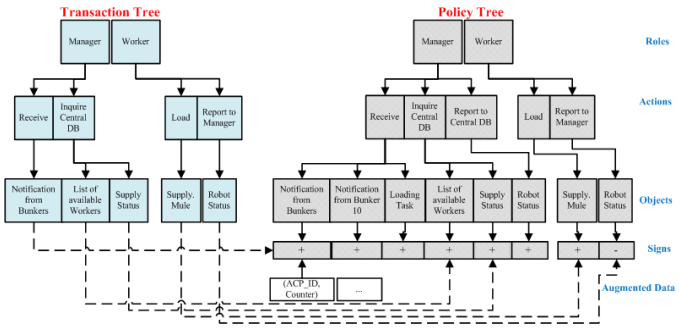


Fig. 2: Policy analysis structures: policy tree (right) and transaction tree (left)

transactions executed in the system. Therefore, in [12], we introduced two data structures: Policy Tree and Transaction Tree (shown in Fig. 2). Both of these structures are multi-way tree structures which consist of different node types including role, action, and object. In addition, the Policy Tree has another node type which is a sign node. In both trees, the first level consists of role nodes representing all roles in the system. The second level of the two trees consists of action nodes, and the third level consists of object nodes (which represent the leaf level in the Transaction Tree). The leaf level in the policy tree consists of sign nodes[2]. Thereafter, we introduce a new data structure to enhance the analysis service.

3.2.1 Policy-and-Transaction Tree (PT-Tree)

Instead of building two individual structures (i.e., policy tree and transaction tree), a hybrid structure (referred to as a *policy-and-transaction tree*) can be built to store both sets of transactions and access control policies in a tightly-combined organization. In order to build such an integrated structure, the structure of policy tree can be adapted to store access control policies as well as their corresponding transactions. When an executed transaction does not have a corresponding policy that controls it, the transaction is added as a policy but the value for the sign node is special[3] and the value for *Policy ID* is empty. Otherwise, the path corresponding to the enforced policy while executing the transaction is updated by incrementing the *Counter* value.

3.2.2 Time Complexity of Structure Construction

For discussing the time asymptotic analysis for constructing the policy, transaction, and PT trees, we used the notations listed in Table 3.

At the running time of a system, every transaction executed by a user is mapped to a record matching an access control policy. Thus, the unique set of transactions m is bounded by the set of access control policies (i.e., $m = \mathcal{O}(n)$). Regarding the structure of the policy tree and transaction tree, the fan-out of a node depends on the node type (i.e., the fan-out of a root node (f_R), action node f_A, and object node f_O vary based on the specifications of an access control system). However, the fan-out of a node in these trees f can be generalized by the maximum of the various node fan-outs (i.e., $f = max(f_R, f_R, f_R)$). Subsequently, the time cost of searching the policy or transaction tree is $\mathcal{O}(\log_f n)$. Inserting a transaction or a policy involves searching the tree structure; hence the time cost of insertion is also $\mathcal{O}(\log_f n)$.

Based on the aforementioned discussion, given n policies, the time cost of constructing the policy tree is $\mathcal{O}(n \log_f n)$. While for constructing the transaction tree [4], the operations of inserting a transaction into the transaction tree and searching for the corresponding policy in the policy tree are performed in sequential; hence these two operations are bounded by $\mathcal{O}(\log_f n)$. Given m transactions, the time cost of constructing the transaction tree is also $\mathcal{O}(m \log_f n)$. Thus, the construction time of a transaction tree is asymptotically larger than the one of a policy tree.

Intuitively, the asymptotic analysis of the PT tree is bounded by the maximum of that for the policy tree and transaction tree (i.e., $\mathcal{O}(max(\text{policy tree, transaction tree}))$. Subsequently, the insertion and construction times of the hybrid structure are $\mathcal{O}(\log_f n)$, $\mathcal{O}(m \log_f n)$, respectively.

TABLE 3: Notations for The Asymptotic Time Analysis

Notation	Description
n	The number of access control policies.
m	The number of transactions.
f_R	The number of unique roles which represents the maximum fan-out of the root node of the tree structures.
f_A	The number of unique actions which represents the maximum fan-out of an action node in the tree structure.
f_O	The number of unique objects which represents the maximum fan-out of an object node in the tree structure.

4 Policy Analysis Services

Our framework supports two types of analysis: structure-based and classification-based. The structure-based analysis aims at analyzing all policies and find any "low quality" policy by using the data structures discussed in Section 3. However, this approach might be expensive (in terms of space and time especially in a real-time environment). Alternatively, the classification-based analysis approach learns the characteristics of each type of "low quality" policies. The classification-based approach is able to quickly predict the quality of a policy, but inaccurate predictions might happen.

4.1 Structure-based Analysis

The search space of the policy-based analytic service is bounded by the access control policy set itself. Hence the policy-based analytic service is not able to assess all quality requirements (e.g., cannot assess incompleteness). Thus, the transaction-based analytic service expands the search space to cover both the executed transactions and their corresponding policies. From another perspective, the policy-based analysis follows the paradigm "*analyze first and enforce later*" while the transaction-based analysis follows the opposite paradigm (i.e.,"*enforce first and analyze later*").

2. For more details, please refer to [12]

3. It implies that there is no corresponding permission neither accepting nor rejecting the access.

4. We refer the reader to Algorithm 1 in [12].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

TABLE 4: The Objectives of Policy Analysis Services

	Policy-based Analysis	Transaction-based Analysis
Inconsistency	✓	✓
Exception	✗	✓
Incompleteness	✗	✓
Redundancy	✓	✓
Irrelevancy	✓	✗

For situations that can be detected by both types of analysis, the policy-based one is preferred because it provides early feedback about the quality of policies. Both policy-based analysis and transaction-based analysis are discussed thoroughly in [12].

4.1.1 Policy-based Analysis

This service utilizes the policy tree structure. Subsequently, it is able to detect inconsistency, redundancy, and irrelevancy. The policy-based approach is not able to detect the exceptions and the incomplete policies as these can only be detected by analyzing the transactions executed in the system as well as their corresponding access control policies.

Time Complexity: The policy-based analysis requires inspecting all access control policies n. Furthermore, for every policy, the approach searches for the other policies that involve similar objects. This leads to the time cost of $\mathcal{O}(n \log_f n)$.

4.1.2 Transaction-based Analysis

Unlike the policy-based analysis, this service utilizes the transaction tree primarily and explores the corresponding policies in the policy tree. Subsequently, this service is able to detect inconsistency, redundancy, incompleteness, and exceptions. Since irrelevant policies do not have corresponding transactions in the transaction tree, such policies can not be reported by the transaction-based approach.

Time Complexity: The transaction-based analysis requires inspecting all unique set of transactions (i.e., $\mathcal{O}(n)$). For every transaction, the approach checks its corresponding policy through the associated pointer linking both trees; hence requiring a constant time. Consequently, the time cost of the transaction-based analysis approach is $\mathcal{O}(n)$.

4.1.3 Policy-and-Transaction-based Analysis (PT-based Analysis)

Neither of the policy-based analysis nor the transaction-based analysis are able to achieve all of the objectives of the policy analysis as shown in Table 4. Nonetheless, performing an analysis utilizing the PT tree can achieve all of the analysis objectives effectively.

Time Complexity: The PT-based analysis requires inspecting all access control policies and their corresponding transactions stored in the hybrid structure and verifying the similarity of each policy with other policies. Thus, the time complexity of the PT-based analysis is bounded by the maximum of that for the Policy-based analysis and Transaction-based analysis. Subsequently, the time cost of the PT-based analysis is $\mathcal{O}(n \log_f n)$.

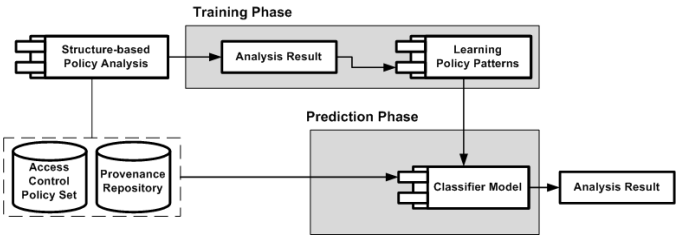


Fig. 3: The Pipeline of Classification-based Policy Analysis

4.2 Classification-based Analysis

The structure-based analysis requires maintaining some data structures. To avoid having to inspect the tree structures periodically, the classification-based analysis (see Fig. 3) aims at learning the patterns of low-quality policies based on the historical results of the structure-based analysis and then generating a classifier. Thereafter, the classifier is used to assess policies and predict whether they will not meet the quality requirements.

4.2.1 Classification of Access Control Policies

The historical results generated from the structure-based policy analysis can be used to generate the patterns of policies that are of "low quality" and thus create categories (i.e., classes) of policies. Each policy quality requirement characterized by a sample set of policies is considered as a class for the corresponding policies. A subset of policies for each category is used to train a classifier for creating a model that summarizes the patterns of policies belonging to each category. In particular, each policy and transaction is represented as a feature vector consisting of role, object, action, sign, and policy quality class[5] (e.g., irrelevant, inconsistent). Then, the classifier utilizes the model for predicting the class of new policies.

There are, however, two main challenges that are associated with the classification-based analysis approach: the imbalanced categories of historical policies and the inevitable classification inaccuracy. Learning the patterns of imbalanced categories potentially leads to a biased pattern learning for the dense categories (i.e., the categories which have many policies); hence increasing the classification inaccuracy to the sparse categories (i.e., the categories which have few numbers of policies). Thus, for obtaining balanced categories of policies, we sample more policies policies using the SMOTE (Synthetic Minority Oversampling Technique) algorithm [20] to over-sample [32] the class with minority samples. Regarding the classification inaccuracy, there is no optimal classifier that definitely guarantees accurate prediction results. Thus, our framework addresses this challenge by adopting the cross-validation mechanism while training a classifier and proposing a classification scheme which combines the classification results obtained from different classifiers to enhance the classification accuracy.

4.2.2 Classification Approaches

Here, we present two classification schemes which we use in our framework.

5. The classification of policy quality is a multi-class classification (i.e., irrelevant, inconsistent, redundant, exception, incomplete, and good-quality).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

Algorithm 1 Classification-based Analysis - Combined Classifiers Approach

1: Let D denote the training set of Access Control Policies, k denote the number of base classifiers, T be the test set of policies, m denote the mode of combined classifier approach, and n denote the number of classes.
2: **for** $i \in \{1, \ldots, k\}$ **do**
3: Build a base classifier C_i from D
4: **for** each class $j \in \{1, \ldots, n\}$ **do**
5: α_{ij} = Accuracy(C_i, $\forall$ policy $x \in j$)
6: **end for**
7: **end for**
8: **for** each policy $x \in T$ **do**
9: **if** m: Majority voting based **then**
10: $C^*(x)$ = Vote($C_1, C_2, \ldots, C_K$)
11: **else if** m: Probability-based **then**
12: $C^*(x) = C_i(x) \mid \alpha_i = \max_{i=1}^{k} \alpha_i$
13: **end if**
14: **end for**

One Classifier (OC)

The OC approach adopts one of the state-of-the-art classifiers (i.e., k-Nearest Neighbors (kNN), Naïve Bayes, Support Vector Machine (SVM), Decision Tree, or Random Forest) at a time. In the experiment section, we discuss the impact of the choice of the classifier on the policy classification.

Combined Classifiers (CC)

Since classifiers inevitably suffer from inherent classification inaccuracy, classifying a policy with different classifiers may potentially result in various categories for a given policy. Consequently, combining the results of various classifiers potentially increases the certainty of the classification result.

We investigate two methods for combining the classifiers: majority voting [46] (referenced as *Majority-based Combined Classifiers* (MCC)) and maximum probability [24] (referenced as *Probability-based Combined Classifiers* (PCC)). The majority voting method builds a consensus of opinion among classifiers. In particular, the method selects the predicted class that is supported by the majority of the classifiers. If there is no majority voting (i.e., a strong disagreement among classifiers concerning the predicted class), the method randomly selects one of the classes predicted from one of the classifiers. Meanwhile, PCC utilizes the probabilities associated with the predicted class using every classifier and chooses the class from the classifier whose probability is the highest. The two methods of the combined classification scheme are formalized in Algorithm 1.

5 Policy Evolution Services

A dynamic system often requires modifying its access control policies. Especially, when the policy analysis process identifies some low-quality policies. The analysis results can be then be used to evolve the system. However, modifying the policies manually is not reliable and is expensive in terms of human administrative time and efforts. Hence, we propose a set of policy evolution algorithms that aim at automatically evolving the policies in order to enhance them. The policy evolution module (see Fig. 4) comprises two services, i.e., recommendation and re-evaluation services, that we describe in what follows.

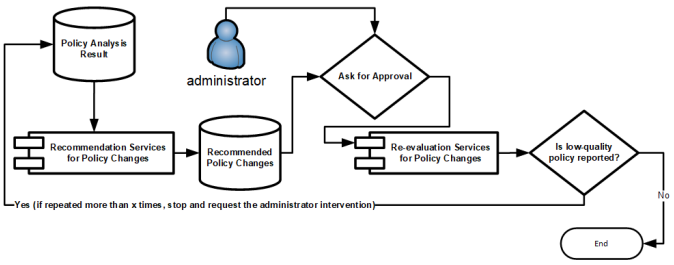


Fig. 4: Policy Evolution Services

5.1 Recommendation Services for Policy Changes

The recommendation services receive as input the results of the policy analysis module and return a set of recommended primitive policy changes. In particular, there is a recommendation service for each type of low-quality policies (see Algorithms 2-4). Each service inspects the set of access control policies and transactions thoroughly to evaluate the causes for low-quality policies and suggests accordingly one or multiple primitive changes (see Table 5 for the list of primitive changes).

The recommendation service for redundant policies is described in Algorithm 2. Given two redundant policies, the service basically checks whether the two policies are identical (lines 2-3) or similar (lines 4-33). Checking the similarity of two policies implies verifying the similarity of their corresponding three components (i.e., role, object, and action) [6] as follows:

- If the corresponding roles are related by the parent-child relationship, the two policies are similar (lines 5-22). Hence, the service recommends either deleting the policy having the child role or changing the role of the parent policy through analyzing the executed transactions associated with them.
- If the object of one policy is a child of the object of the other policy, the two policies are similar (lines 23-27). Hence, the service recommends deleting the policy corresponding to the child object.
- If the action of one policy is partially implied by the action of the other policy, the two policies are considered similar (lines 28-32). Hence, the service recommends deleting the policy corresponding to the partial action.

Regarding inconsistent policies, their recommendation service is described in Algorithm 3. Given two inconsistent policies, the service checks the similarity of the corresponding three components. In particular, if the corresponding roles are related by the parent-child relationship, the inconsistent policies are considered similar. Subsequently, the service recommends creating a new role for the users belonging to the parent role (except the users belonging to the child role of interest), and then assigning the policy associated with the parent to the new role (lines 4-10). Also, the service

6. The sign is not checked because the two policies are reported as redundant and thus they have the same sign.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

TABLE 5: Primitive Changes on Policies

Primitive	Description
$DEL\_POLICY(ACP_i)$	Delete the policy ACP_i
$CHG\_POLICY_r(ACP_i, x)$	Change policy role: $ACP_i.r = x$
$CHG\_POLICY_a(ACP_i, x)$	Change policy action: $ACP_i.p.a = x$
$CHG\_POLICY_o(ACP_i, x)$	Change policy object: $ACP_i.p.o = x$
$CHG\_POLICY_s(ACP_i, x)$	Change policy sign: $ACP_i.p.sign = x$
$ADD\_POLICY(ACP_i)$	Add the policy ACP_i ⟨role: $ACP_i.r$, permission: ($ACP_i.p.a$, $ACP_i.p.o$, $ACP_i.p.sign$)⟩
$ADD\_ROLE(r_i, r_j)$	Add the role r_i to the parent role r_j
$DEL\_ROLE(r_i)$	Delete the role r_i

recommends similar changes when the inconsistent policies are considered similar with respect to the objects (lines 11-16) and actions (lines 17-23).

The recommendation service for exceptional transactions is described in Algorithm 4. Given a policy and its corresponding exceptional transaction, the service checks whether the policy has been enforced. If not, the service recommends negating the sign of the policy. Otherwise, the service checks the following conditions when the exceptional transaction has been repeated a number of times higher than a certain threshold (i.e., σ):

- If the corresponding roles of the exceptional transaction and the policy are related by a parent-child relationship, the service recommends creating a new role for the users belonging to the parent role (except the users belonging to the child role of interest), and then assigning the policy associated with the parent to the new role (lines 8-11). Also, the service recommends adding a new policy to permit the exceptional transaction for the child role (lines 12-14).
- If the corresponding objects (lines 18-26) or actions (lines 27-35) of the exceptional transaction and the policy are related by a parent-child relationship, the service recommends similar changes to the ones recommended when the roles of the exceptional transaction and its associated policy are related by a parent-child relationship.

Due to space limitations, we omit the algorithms for the recommendation services in two cases: irrelevancy and incompleteness. Those algorithms are quite simple. In the case of irrelevant policies, the corresponding service recommends deleting it, whereas in the case of incomplete policies the service recommends adding new policies to cover the missing scenarios.

Validating the parent-child relations, among the corresponding roles, objects, and actions of the two policies, requires inspecting some auxiliary structures. In particular, the roles are validated using the role hierarchies captured by the RBAC model while the objects are inspected using metadata defining object hierarchies in the system.

5.2 Re-evaluation Services for Policy Changes

The second set of services is to perform the changes recommended by the recommendation services, and then re-evaluate the policies which are affected by these recommended changes to assure that the quality of the policy set has been improved as intended. Evaluating the policy set after its evolution is a critical step. However, re-evaluating the quality of the entire policy set can be expensive. Thus, the re-evaluation services particularly consider only the policies that are directly affected by the evolution.

Towards such goal, the re-evaluation services investigate each of the primitives changes to track their effects on the set of policies and narrows the scope of the re-analysis to be performed only for them. However, some of these primitives, such as ADD_ROLE and DEL_ROLE, have no direct impact on the policy set. Nonetheless, such primitives are associated with other primitives (i.e., CHG_POLICY_r and DEL_POLICY, respectively) which directly affect the policy set.

When adding a new policy or deleting an existing policy acp_i (i.e., the primitive ADD_POLICY and DEL_POLICY, respectively), the policy tree (or PT tree) is first searched using a depth-first-search to find a path whose nodes match the components composing acp_i. In particular, the set of policies that are affected by this change primitive includes any policy acp_j that is specified by the components: a role ($acp_j.r$) matched to $acp_i.r$ (or $acp_j.r$ and $acp_j.r$ are related by a parent-child relation), an action $acp_j.p.a$ matched to (or part of) $acp_i.p.a$, and an object $acp_j.p.o$ matched to $acp_i.p.o$ (or $acp_j.o$ and $acp_j.o$ are related by a parent-child relation).

Meanwhile, when changing a role of a policy (i.e., the primitive CHG_POLICY_r), the set of policies that are affected by this change primitive includes any policy acp_j that is specified by the components: a role $acp_j.r$ matched to (or share a parent-child relation with) either of the old or new values of $acp_i.r$, an action $acp_j.p.a$ matched to (or part of) either of the old or new values of $acp_i.p.a$, and an object $acp_j.p.o$ matched to (or share a parent-child relation with) either of the old or new values of $acp_i.p.o$. Similarly, finding the policies affected by the execution of the primitives CHG_POLICY_a and CHG_POLICY_o follows the same logic. The procedure of finding the set of affected polices for all primitive changes is outlined in Algorithm 5. Thereafter, the identified affected policy set is re-evaluated either by partially traversing the policy tree (or PT tree) or performing the classification-based analysis on the batch of affected policies.

6 The ProFact Framework

We now introduce our policy analysis framework which utilizes provenance metadata to enable efficient policy analysis. As shown in Fig. 5, the framework is based on three main phases: data collection, policy analysis, and policy evolution. The data collection phase uses a provenance logging component which captures all transactions executed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

Algorithm 2 Recom. Algorithm for Redundant Policies

1: Given two policies ACP_i and ACP_j are redundant.
2: **if** ACP_i = ACP_j **then**
3: Recommend $DEL\_POLICY(ACP_i)$
4: **else**
5: **if** $ACP_i.r \subseteq ACP_j.r$ **then**
6: t_i = Query_Transactions(ACP_i)
7: t_j = Query_Transactions(ACP_j)
8: **if** $\mid t_j - t_i \mid = 0$ **then**
9: Recommend $DEL\_POLICY(ACP_i)$
10: **else if** $\mid t_j - t_i \mid > \mid t_i \mid$ **then**
11: Recommend $DEL\_POLICY(ACP_i)$
12: **if** |Query_Policies($acp_i.r$) = 1 **then**
13: Recommend $DEL\_ROLE(acp_i.r)$
14: **end if**
15: **else**
16: Recommend $ADD\_ROLE(r_k, acp_j.r)$
17: Assign users(t_j - t_i) to r_k
18: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
19: **end if**
20: **else**
21: Similar Logic to Lines 6-18 but swapping i and j.
22: **end if**
23: **if** $ACP_i.p.o \subseteq ACP_j.p.o$ **then**
24: Recommend $DEL\_POLICY(ACP_i)$
25: **else**
26: Recommend $DEL\_POLICY(ACP_j)$
27: **end if**
28: **if** $ACP_i.p.a \subseteq ACP_j.p.a$ **then**
29: Recommend $DEL\_POLICY(ACP_i)$
30: **else**
31: Recommend $DEL\_POLICY(ACP_j)$
32: **end if**
33: **end if**

Algorithm 3 Recom. Algorithm for Inconsistent Policies

1: Given two policies ACP_i and ACP_j are inconsistent
2: t_i = Query_Transactions(ACP_i)
3: t_j = Query_Transactions(ACP_j)
4: **if** $ACP_i.r \subseteq ACP_j.r$ **then**
5: Recommend $ADD\_ROLE(r_k, acp_j.r)$
6: Assign users(t_j - t_i) to r_k
7: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
8: **else**
9: Similar Logic to Lines 5-7 but swapping i and j.
10: **end if**
11: **if** $ACP_i.p.o \subseteq ACP_j.p.o$ **then**
12: o_k = ($\forall\ o_w \in ACP_j.p.o$) - $ACP_i.p.o$
13: Recommend $CHG\_POLICY_o(ACP_j, o_k)$
14: **else**
15: Similar Logic to Lines 12-13 but swapping i and j.
16: **end if**
17: **if** $ACP_i.p.a \subseteq ACP_j.p.a$ **then**
18: Recommend $ADD\_ROLE(r_k, acp_j.r)$
19: a_k = ($\forall\ a_w \in ACP_j.p.a$) - $ACP_i.p.a$
20: Recommend $CHG\_POLICY_a(ACP_j, a_k)$
21: **else**
22: Similar Logic to Lines 18-20 but swapping i and j.
23: **end if**

in the system in addition to all changes made on the access control policy set and stores them in the provenance repository. To support the data collection phase, the framework maintains a tree structure (i.e., PT tree) which abstracts the necessary information for the analysis phase. The analysis phase includes two types of analysis approaches: structure-based and classification-based. The results of the analysis approaches are aggregated into a separate repository referred to as *policy analysis repository*. The evolution phase comprises two main services: policy change recommendation and policy re-evaluation. The results of the re-analysis are also aggregated into the policy analysis repository.

Furthermore, ProFact includes an orthogonal component for query services that supports the following query types:

- **Queries on the Quality of Policies:** Such queries retrieve the policies which do not satisfy our quality requirements. Querying on quality might be general (e.g., find all policies which are inconsistent, find all irrelevant policies) or specific to policy attributes (e.g., find all inconsistencies related to a specific object or find roles with respect to which policies are incomplete).
- **Queries on Policies:** These queries allow one to retrieve basic information on policies (e.g., policies for a role, how many times a policy was enforced) and advanced information on policies (e.g., the history of a policy, how the policy was evolved).
- **Queries on Transactions:** These queries allow one to retrieve information about the executed transactions. Examples include: find the transactions executed by a certain user (through different roles), and find the transactions that accessed specific objects.
- **Queries on Policy Analysis Statistics:** These queries utilize the policy analysis repository to retrieve aggregated analysis results. Examples include: retrieve the most common exceptions and the frequency of an exception.
- **Queries on Policy Evolution Statistics:** These queries allow one to estimate the percentage of policies affected by a specific type of policy changes or find the policy components mostly affected by a specific type of policy changes; these queries are based on the historical runs of the evolution service. Examples include: find the object which was mostly affected by policy deletion, and what is the primitive change that affected the largest number of policies.

7 Experiments

The goal of the experiments is to evaluate the two types of policy analysis approaches included in ProFact: structure-based and classification-based.

7.1 Dataset and Settings

Due to the lack of a large-scale real dataset compromising both access control policies and executed transactions for a real system[7], we created three synthetic datasets (referenced

7. A large-scale dataset is required to generate machine learning models for the classification-based analysis approach.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

Algorithm 4 Recom. Algorithm for Exceptional Transactions

1: Given a transaction t_i $\langle r, o, a\rangle$ that violates an existing policy ACP_j
2: n = Count_Transactions(t_i)
3: t_j = Query_Transactions(ACP_j)
4: **if** $\mid t_j \mid = 0$ **then**
5: $s = \neg\, ACP_j.p.sign$
6: $CHG\_POLICY_s(ACP_j, s)$
7: **else if** $n > \sigma$ **then**
8: **if** $t_i.r \subseteq ACP_j.r$ **then**
9: Recommend $ADD\_ROLE(r_k, acp_j.r)$
10: Assign users(t_j - t_i) to r_k
11: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
12: $s = \neg\, ACP_j.p.sign$
13: Set ACP_i = ⟨role: $t_i.r$, permission: ($t_i.a$, $t_i.o$, s⟩
14: Recommend $ADD\_POLICY(ACP_i)$
15: **else**
16: Similar Logic to Lines 9-14.
17: **end if**
18: **if** $t_i.o \subseteq ACP_j.p.o$ **then**
19: o_k = ($\forall\ o_w \in ACP_j.p.o$) - $t_i.o$
20: Recommend $CHG\_POLICY_o(ACP_j, o_k)$
21: $s = \neg\, ACP_j.p.sign$
22: Set ACP_i = ⟨role: $t_i.r$, permission: ($t_i.a$, $t_i.o$, s⟩
23: Recommend $ADD\_POLICY(ACP_i)$
24: **else**
25: Similar Logic to Lines 19-23.
26: **end if**
27: **if** $t_i.p.a \subseteq ACP_j.p.a$ **then**
28: a_k = ($\forall\ a_w \in ACP_j.p.a$) - $t_i.a$
29: Recommend $CHG\_POLICY_a(ACP_j, a_k)$
30: $s = \neg\, ACP_j.p.sign$
31: Set ACP_i = ⟨role: $t_i.r$, permission: ($t_i.a$, $t_i.o$, s⟩
32: Recommend $ADD\_POLICY(ACP_i)$
33: **else**
34: Similar Logic to Lines 28-32.
35: **end if**
36: **end if**

Algorithm 5 Find Affected Policies

1: Let $acp_i :=< r, p.a, p.o, p.sign >$ denote a policy to be changed, M denote the policy evolution type $\{Add, Update, Delete\}$, and T denote the policy tree constructed on D.
2: **if** $M \equiv Add \vee M \equiv Delete$ **then**
3: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
4: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
5: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
6: $P = \{\forall acp_j \mid (acp_j.r \in R) \wedge (acp_j.p.a \in A) \wedge (acp_j.p.o \in O) \wedge (i \neq j)\}$
7: **else if** $M \equiv Update$ **then**
8: **if** updating $acp_i.r$ **then**
9: $R = \{\forall r \mid (r \subseteq acp_i.r_{old}) \vee (acp_i.r_{old} \subseteq r) \vee (r \subseteq acp_i.r_{new}) \vee (acp_i.r_{new} \subseteq r)\}$
10: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
11: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
12: **else if** updating $acp_i.p.a$ **then**
13: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
14: $A = \{\forall a \mid (a \subseteq acp_i.p.a_{old}) \vee (acp_i.p.a_{old} \subseteq a) \vee (a \subseteq acp_i.p.a_{new}) \vee (acp_i.p.a_{new} \subseteq a)\}$
15: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
16: **else if** updating $acp_i.p.o$ **then**
17: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
18: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
19: $O = \{\forall o \mid (o \subseteq acp_i.p.o_{old}) \vee (acp_i.p.o_{old} \subseteq o) \vee (o \subseteq acp_i.p.o_{new}) \vee (acp_i.p.o_{new} \subseteq o)\}$
20: **else if** updating $acp_i.p.sign$ **then**
21: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
22: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
23: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
24: **end if**
25: $P = \{\forall acp_j \mid (acp_j.r \in R) \wedge (acp_j.p.a \in A) \wedge (acp_j.p.o \in O) \wedge (i \neq j)\}$
26: **end if**
27: **Return** P

as *DS1*, *DS2*, and *DS3*) using a random dataset generator (as explained later) to evaluate the policy analysis approaches. Table 6 shows the size of each dataset in terms of the number of roles, protected objects, access control policies, and transactions.

Random Dataset Generator: The dataset generator first generates randomly the basic entities of a dataset (i.e., users, roles, objects, and actions[8]. Thereafter, the generator creates hierarchical relations (i.e., parent-child relations) among the created roles and objects. For example, for generating the role hierarchies, a quarter of the roles are initially selected as parent roles and each of the remaining roles is assigned as a child role to one of the parent roles and this child role is appended to the set of the parent roles. The object hierarchies are similarly generated. Using a maximum value for role assignment (i.e., m), each user is assigned randomly at maximum to a set of m roles. Subsequently, every user is assigned to multiple roles and every role includes multiple users (given the fact that the number of roles is much smaller than the number of users). Using all combinations (referenced as $\mathcal{X}$) of roles, object, and actions, we randomly select a subset $\mathcal{Y} \subset \mathcal{X}$ to generate a set of transactions $\mathcal{Y}$. Similarly, we randomly select a subset $\mathcal{Z} \subset \mathcal{X}$ to generate a set of access control policies $\mathcal{Z}$ and the sign of each policy is randomly chosen as positive or negative[9]. Finally, the generator randomly assigns a frequency counter to each transaction. As a result, the generated dataset intentionally includes all types of low-quality policies. The subset of transactions $\mathcal{Y} - (\mathcal{Z} \cap \mathcal{Y})$ indicates missing policies (i.e., incompleteness). Meanwhile, the subset of policies $\mathcal{Z} - (\mathcal{Y} \cap \mathcal{Z})$ is irrelevant. The subset of policies which have corresponding transactions $(\mathcal{Y} \cap \mathcal{Z})$ can potentially include inconsistencies, redundancy, and exceptions. Table 7 shows the distribution of the low-quality policies among the three generated datasets.

Classification approach settings: For the classifiers adopted in our classification-based analysis, we used the Java Weka library [29]. All classifiers are trained on 70%

8. The generator randomly creates at maximum n instances of an entity where n is specified by the user of the generator.

9. The generator verifies that $\mathcal{Y} \cap \mathcal{Z} \neq \phi$ to guarantee having all low-quality policy types.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing

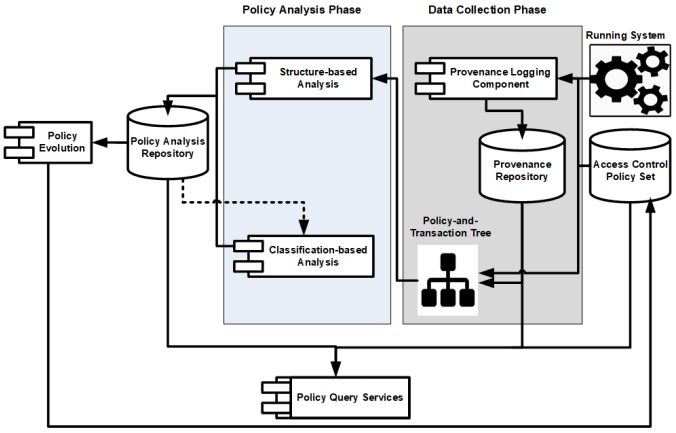


Fig. 5: The Infrastructure of the ProFact Framework

TABLE 6: Access Control Policy and Transaction Datasets

	# Roles	# Objects	# Policies	# Transactions
Dataset 1 (*DS1*)	50	2,000	46,800	124,800
Dataset 2 (*DS2*)	75	2,500	427,500	641,250
Dataset 3 (*DS3*)	100	3,000	877,200	1,152,000

of the dataset using 10-fold cross-validation. Regarding the kNN classifier, we used the instance-based learner algorithm. For the decision tree classifier, we used the C4.5 Decision algorithm. Overall, we used the Weka default values for the parameters of the machine learning techniques (e.g., for kNN classifier, k = 1).

We implemented the prototype infrastructure of ProFact in Java 1.7. All experiments were performed on high-performance computing clusters at Purdue Research Center. The analysis prototype was run on a cluster of one node with 16 cores and 64GB memory.

7.2 Pre-processing Time for the Analysis Approaches

We collected the total time for building the underlying structures for the structure-based approach[10]. Fig. 6a shows the construction time (in base-10 logarithmic scale) for the three variants (i.e., policy-based, transaction-based, and PT-based) of the structure-based approach using the three datasets. In general, the construction of the structure for the transaction-based analysis takes longer time than that of the policy-based analysis because of two reasons: a) the number of transactions in a system usually is larger than the number of access control policies, and b) adding a transaction to the transaction tree involves finding its corresponding policy in the policy tree to link them together. Moreover, the construction time for PT-tree is almost similar to that of the transaction tree.

10. We discarded the training time for every classifier of the classification-based analysis approach because training a classifier is an off-line step. Meanwhile, the construction of the tree structures can not be considered off-line because they should be maintained up to date.

TABLE 7: The Distribution of Low-Quality Policies among Datasets

	Datasets		
	DS1	*DS2*	*DS3*
Inconsistency	162	368	478
Exceptions	2,836	5,762	12,062
Incompleteness	1,672	3,052	8,755
Redundancy	787	962	1,507
Irrelevancy	1,398	1,940	3,896

7.3 Analysis Results

7.3.1 Structure-based Analysis

Performance: Fig. 6b shows the analysis time for detecting inconsistent and redundant policies[11] using the policy-based, transaction-based, and PT-based approaches across the three datasets. In general, the transaction-based approach has the best performance compared to the policy-based approach. In particular, the transaction-based approach has a speedup factor of 4x, 5x, and 3x with respect to the policy-based approach using *DS1*, *DS2*, and *DS3*, respectively. All approaches are based on the tree traversal but with different analysis mechanisms. The transaction-based approach only visits every path in the tree and utilizes the associated link to fetch the corresponding policy in the policy tree. Meanwhile, the policy-based approach searches for all policies that involve similar components while inspecting each policy. The transaction-based approach invested the time spent on constructing the transaction tree and linking it with the policy tree to achieve better performance at the analysis phase. Since the PT-based approach uses the hybrid tree structure that stores both transactions and policies, inspecting all policies and corresponding transactions thoroughly through such a structure leads to extra time for the analysis compared to both the policy-based and transaction-based approaches.

Efficiency: Both policy-based and transaction-based approaches were able to detect all inconsistent and redundant policies. Since irrelevant policies do not have corresponding transactions in the transaction tree, the irrelevant policies were reported by only the policy-based approach. Nonetheless, the policy-based approach was not able to detect the exceptions and the incomplete policies as these can only be detected by analyzing the transactions executed in the system as well as their corresponding access control policies. Thus, such cases are detected by the transaction-based approach. However, the PT-based approach was able to detect all of these cases.

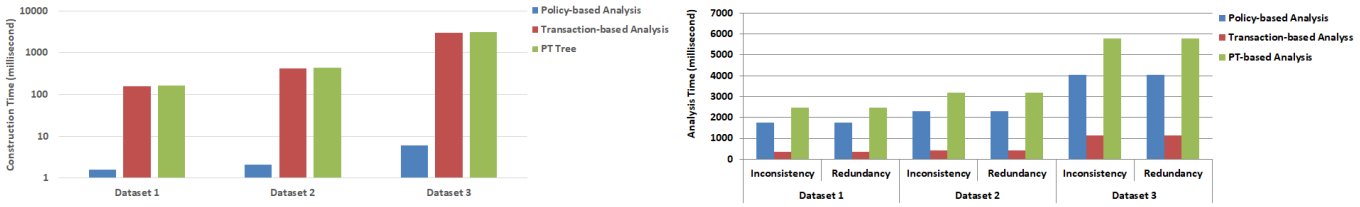
7.3.2 Classification-based Analysis

Performance: Fig. 7a shows the average analysis time per policy (in base-10 logarithmic scale) using both analysis approaches (structure-based and classification-based)[12] across the three datasets. In general, all schemes of the classification-based analysis approach (i.e., OC and CC) outperform the structure-based analysis approach. The analysis time of the classification-based approach using the OC scheme varies based on the adopted classifier. In particular, OC with the Naïve Bayes and Decision Tree classification algorithms have the best performance. Among all classifiers

11. We discarded the other types of low-quality policies and considered only the types of policies that can be detected by the three variants of the structure-based approach.

12. The analysis time of the structure-based analysis represents the PT-based one since it is able to detect all types of low-quality policies.

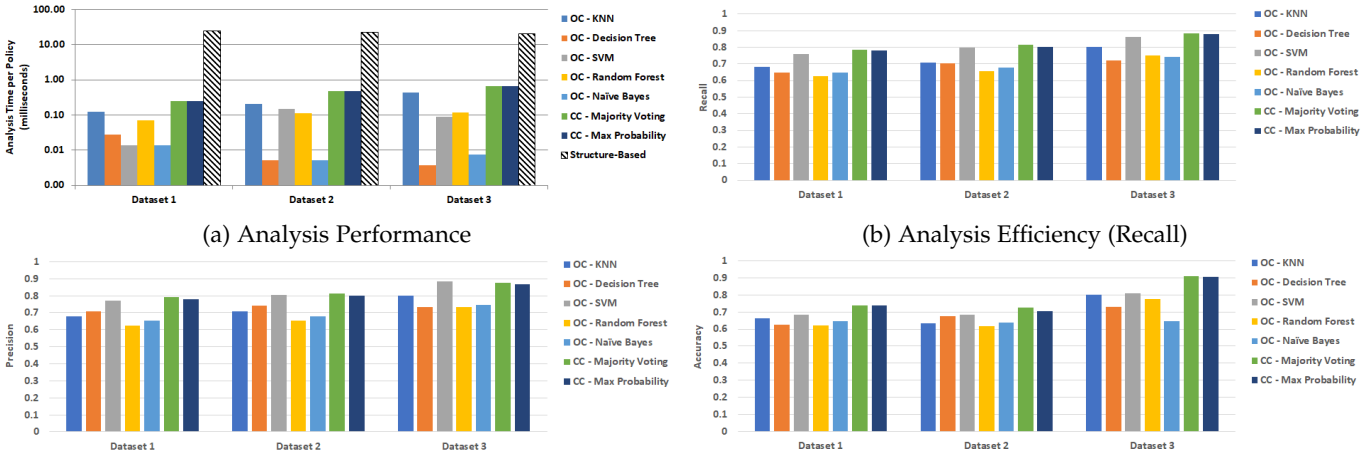
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2900641, IEEE Transactions on Services Computing



(a) Construction Time

(b) Analysis Time

Fig. 6: Structure-based Approaches



(a) Analysis Performance

(b) Analysis Efficiency (Recall)

(c) Analysis Efficiency (Precision)

(d) Analysis Efficiency (Accuracy)

Fig. 7: Classification-based Approaches

used with OC, the kNN and Random Forest classifiers have the worst performance since the kNN classifier requires retrieving all closest objects among the training dataset and Random Forest investigates multiple internal decision trees to conclude the classification result. Intuitively, CC is slower than OC because CC performs the analysis through all classifiers adopted by OC to obtain the final classification results. However, both approaches based on the CC scheme (i.e., Majority Voting and Max Probability) outperform the structure-based approach. In particular, the speedup factor of CC is 31x with respect to the structure-based approach.

Efficiency: To evaluate the efficiency of the classification-based analysis approach, we report the recall, precision, and accuracy values which are formulated in Eqs. 1 - 3. For calculating recall, precision, and accuracy, we used the output of the structure-based analysis results as the ground truth.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Figs. 7b, 7c, and 7d show the recall, precision, and accuracy of both classification schemes (OC and CC) across the three datasets, respectively. In general, using *DS3*, all classification approaches achieve recall and precision values above 70%. Meanwhile, Naïve Bayes was the only one that does not achieve an accuracy value above 70%. Moreover, the efficiency of OC varies based on the trained model and the chosen classifier. For example, Random Forest has the worst recall, and accuracy values using *DS1* and *DS2*, but it has been improved in *DS3*. Consequently, the OC scheme is not reliable for our framework. On the other hand, the CC scheme achieves the best recall, precision, and accuracy compared with OC using all datasets. In particular, the CC scheme achieved at maximum recall, precision, and accuracy of 88%, 87%, and 91%, respectively. This supports our claim that using the combined scheme CC for classification is preferable than using the OC scheme. The CC scheme with Majority Voting and Max Probability techniques achieved almost similar efficiency. However, Majority Voting was the best with a minor difference.

8 Related Work

The area of policy analysis has been widely investigated. Approaches to policy analysis use various methods and are characterized by different goals as we discuss below.

8.1 Goals for Policy Analysis

The goals of policy analysis mainly fall into two directions: assessing the fulfillment of a set of quality requirements, and designing and organizing a set of policies.

Policy Quality Assessment: Past work on policy quality requirements have focused on some of the quality requirements that we have introduced. Among these requirements, consistency was the most investigated one. Gupta et al. [28]

and Cau et al. [19] focused on detecting policies inconsistencies in the RBAC domain. Meanwhile, Mankai et al. [38] and Turkmen et al. [53] proposed methods to detect inconsistency among XACML policies. Regarding the other requirements, redundancy has been evaluated using various approaches such as the ones proposed by Ngo et al. [42], Hadj et al. [5] and Pina et al. [44], while incompleteness is assessed using other research approaches (e.g., [36], [37], [50]). To the best of our knowledge, our approach is the first to assess access control policies with respect to new types of quality requirements (i.e., exceptions and irrelevancy). Because of incorporating provenance metadata, our framework is able to aggregate information about system behavior at execution time, and thus it is able to address all quality requirements.

Policy Design and Organization: For properly reorganizing and evolving policy sets, it is often important to assess the similarity of different policies and the impact of policy modifications. Policy similarity refers to the technique for characterizing the relationships between policies and the actions authorized by them. Several researchers focused on policy similarity including Kolovski et al. [33], Lin et al. [34], [35], Craven et al. [21], [22], and Mazzoleni et al. [39], [40]. Meanwhile, the change impact analysis on the policy set evaluates the changes among two versions of a policy by providing a set of counterexamples that illustrate semantic differences between the two policies. Several research efforts have been devoted to investigating the change impact analysis (e.g., [25], [26], [45], [53]). Our framework includes the policy evolution services which includes analyzing the impact of changing one of the policies on the quality of the correlated policies of the changed policy.

8.2 Methods for Policy Analysis

Various methods have been proposed for policy analysis including formal methods, model checking, data mining, and structure-based (surveyed in [4]). Some policy analysis approaches utilizing formal methods techniques such as reasoning [30] [10], and argumentation [16], [17], [43]. Moreover, several approaches and frameworks for policy analysis have been developed using model checking techniques such as SAT solver [31], [34], or SMT solver [7], [53], and binary decision diagrams [19], [27], [34], [44]. Regarding data mining methods, Shaikh et al. [48], [49], [50] and Aqib et al. [9] proposed several approaches for policy analysis using the decision tree classifier, while Bauer et al. [11] proposed an approach to reorganize the RBAC policies using association rule mining method. Furthermore, structure-based methods were adopted for analyzing access control policies. For example, Xu et al. [55], Staniford et al. [51], Alves et al. [8] used adapted versions of the graph data structure. Meanwhile, the tree structure was used for firewall policy analysis [6], [23]. To the best of our knowledge, our approach is the first to utilize tree-based structures for analyzing access control policies. In addition, our framework proposes another analysis approach based on various classification methods, other than decision trees.

9 Conclusion and Future Work

In this paper, we have proposed a set of requirements to evaluate the quality of access control policies. We have also shown the use of provenance for capturing fine-grained metadata essential for evaluating the quality of policies. Our framework (ProFact) supports various types of query services which convey detailed information about the system environment in the context of transactions and access control policies. ProFact supports two approaches for policy analysis: structure-based and classification-based. Regarding the structure-based approach, our experiments show that transaction-based analysis is faster than policy-based analysis with a maximum speedup factor of 5X. Regarding the classification-based analysis approach, kNN and SVM achieved the best efficiency obtaining a maximum recall of 80% and 86%, respectively. By adopting the combined classifiers, the efficiency improved reaching a recall of 88%. Moreover, classification-based approach outperformed the structure-based approach with a maximum speedup factor of 31x. If the system is not in a real-time environment, the system can adopt the structure-based analysis approach to obtain a 100% recall

As part of future work, we will extend the provenance system to capture spatial context information as this is critical for mobile systems. Furthermore, we will expand our approach to support attribute-based access control policies as these policies allow one to include context-based information in the policy decisions. At a broader level, we plan to extend our policy model, lifecycle, and analytics to support the notion of generative policies model [54] by which devices are given abstract policies and can then autonomously refine and adapt them by using their own analytic services.

Acknowledgments

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.

References

[1] "Extensible access control markup language (xacml) version 3.0," [Last Accessed: Jan. 27, 2018]. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[2] "Prov-overview." [Online]. Available: http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[3] A. Abu Jabal and E. Bertino, "SimP: Secure interoperable multi-granular provenance framework," in *Proceedings on the 2016 IEEE 12th International Conference on e-Science (e-Science)*. IEEE, 2016, pp. 270–275.

[4] A. Abu Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, A. Russo, and C. Williams, "Methods and tools for policy analysis," *ACM Computing Surveys (CSUR)*, 2018.

[5] M. Ait El Hadj, M. Ayache, Y. Benkaouz, A. Khoumsi, and M. Erradi, "Clustering-based approach for anomaly detection in xacml policies," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE), Madrid, Spain, July 24-26, 2017*, 2017, pp. 548–553.

[6] E. S. Al-Shaer and H. H. Hamed, "Modeling and management of firewall policies," *IEEE Transactions on Network and Service Management*, vol. 1, no. 1, pp. 2–10, 2004.

[7] F. Alberti, A. Armando, and S. Ranise, "Efficient symbolic automated analysis of administrative attribute-based rbac-policies," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 165–175.

[8] S. Alves and M. Fernández, "A graph-based framework for the analysis of access control policies," *Theoretical Computer Science*, 2016.

[9] M. Aqib and R. A. Shaikh, "Policy validation tool for access control policies," *Journal of Internet Technology*, 2018.

[10] A. K. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Proceedings of 2003 IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, June 4-6, 2003*. IEEE, 2003, pp. 26–39.

[11] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 2, 2011.

[12] E. Bertino, A. Abu Jabal, S. Calo, C. Makaya, M. Touma, D. Verma, and C. Williams, "Provenance-based analytics services for access control policies," in *Proceedings of the 2017 IEEE World Congress on Services (SERVICES), Honolulu, HI, USA, June 25-30, 2017*. IEEE, 2017, pp. 94–101.

[13] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 191–233, 2001.

[14] E. Bertino, S. Calo, M. Touma, D. Verma, C. Williams, and B. Rivera, "A cognitive policy framework for next-generation distributed federated systems: concepts and research directions," in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1876–1886.

[15] 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.

[16] G. Boella, J. Hulstijn, and L. van der Torre, "Argument games for interactive access control," in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, Compiegne, France, 19-22 September 2005*. IEEE, 2005, pp. 751–754.

[17] G. Boella, J. Hulstijn, and L. Van Der Torre, "Argumentation for access control," *AI* IA 2005: Advances in Artificial Intelligence*, pp. 86–97, 2005.

[18] S. B. Calo, D. C. Verma, and E. Bertino, "Distributed intelligence: Trends in the management of complex systems," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. ACM, 2017, pp. 1–7.

[19] A. Cau, H. Janicke, and B. Moszkowski, "Verification and enforcement of access control policies," *Formal Methods in System Design*, vol. 43, no. 3, pp. 450–492, 2013.

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[21] R. Craven, J. Lobo, E. Lupu, J. Ma, A. Russo, M. Sloman, and A. Bandara, "A formal framework for policy analysis," *Imperial College London, Tech. Rep*, 2008.

[22] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, and A. Bandara, "Expressive policy analysis with enhanced system dynamicity," in *Proceedings of the 4th ACML Symposium on Information, Computer, and Communications Security (ASIACCS), Sydney, Australia, March 10-12, 2009*. ACM, 2009, pp. 239–250.

[23] S. Davy, B. Jennings, and J. Strassner, "Efficient policy conflict analysis for autonomic network management," in *Proceedings of the IEEE 5th Workshop on Engineering of Autonomic and Autonomous Systems (EASE)*. IEEE, 2008, pp. 16–24.

[24] R. P. Duin and D. M. Tax, "Experiments with classifier combining rules," in *MCS*. Springer, 2000, pp. 16–29.

[25] Y. Elrakaiby, T. Mouelhi, and Y. Le Traon, "Testing obligation policy enforcement using mutation analysis," in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), Montreal, QC, Canada, April 17-21, 2012*. IEEE, 2012, pp. 673–680.

[26] S. R. Fatih Turkmen, Jerry den Hartog and N. Zannone, "Analysis of xacml policies with smt," in *Proceedings of the Principles of Security and Trust - 4th International Conference (POST), Held as Part of the European Joint Conferences on Theory and Practice of Software, London, UK, April 11-18, 2015*, vol. 9036, Lecture Notes in Computer Science. Springer, Heidelberg, 2015, pp. 115–134.

[27] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th International Conference on Software Engineering, 2005 (ICSE)*. IEEE, 2005, pp. 196–205.

[28] P. Gupta, S. D. Stoller, and Z. Xu, "Abductive analysis of administrative policies in rule-based access control," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 412–424, 2014.

[29] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[30] J. Y. Halpern and V. Weissman, "Using first-order logic to reason about policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 4, p. 21, 2008.

[31] G. Hughes and T. Bultan, "Automated verification of access control policies using a sat solver," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, no. 6, pp. 503–520, 2008.

[32] N. Japkowicz, "The class imbalance problem: Significance and strategies," in *Proc. of the Intl Conf. on Artificial Intelligence*, 2000.

[33] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies," in *Proceedings of the 16th International Conference on World Wide Web (WWW), Banff, Alberta, Canada, May 8-12, 2007*. ACM, 2007, pp. 677–686.

[34] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo, "Exam: a comprehensive environment for the analysis of access control policies," *International Journal of Information Security*, vol. 9, no. 4, pp. 253–273, 2010.

[35] D. Lin, P. Rao, E. Bertino, and J. Lobo, "An approach to evaluate policy similarity," in *Proceedings of the 2007 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2007.

[36] J. Ma, D. Zhang, G. Xu, and Y. Yang, "Model checking based security policy verification and validation," in *Proceedings of the 2010 Second International Workshop on Intelligent Systems and Applications (ISA)*. IEEE, 2010, pp. 1–4.

[37] X. Ma, R. Li, Z. Lu, and W. Wang, "Mining constraints in role-based access control," *Mathematical and Computer Modelling*, vol. 55, no. 1, pp. 87–96, 2012.

[38] M. Mankai and L. Logrippo, "Access control policies: Modeling and validation," in *5th NOTERE Conference (Nouvelles Technologies de la Répartition)*, 2005, pp. 85–91.

[39] P. Mazzoleni, E. Bertino, B. Crispo, and S. Sivasubramanian, "Xacml policy integration algorithms: not to be confused with xacml policy combination algorithms!" in *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), Lake Tahoe, California, USA, June 7-9, 2006*. ACM, 2006, pp. 219–227.

[40] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino, "Xacml policy integration algorithms," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 1, p. 4, 2008.

[41] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The open provenance model (v1.00)." [Online]. Available: http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf

[42] C. Ngo, Y. Demchenko, and C. de Laat, "Decision diagrams for xacml policy evaluation and management," *Computers & Security*, vol. 49, pp. 1–16, 2015.

[43] L. Perrussel, S. Doutre, J.-M. Thévenin, and P. McBurney, "A persuasion dialog for gaining access to information," in *Proceedings of the 4th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS), Honolulu, HI, USA, May 15, 2007*. Springer, 2007, pp. 63–79.

[44] S. Pina Ros, M. Lischka, and F. Gómez Mármol, "Graph-based xacml evaluation," in *Proceedings of the 2012 17th ACM Symposium on Access Control Models and Technologies (SACMAT), Newark, NJ, USA, June 20-22, 2012*. ACM, 2012, pp. 83–92.

[45] D. J. Power, M. Slaymaker, and A. Simpson, "Conformance checking of dynamic access control policies," in *Proceedings of the 13th International Conference on Formal Engineering Methods (ICFEM), Durham, UK, October 26-28, 2011*. Springer, 2011, pp. 227–242.

[46] D. Ruta and B. Gabrys, "Classifier selection for majority voting," *Information fusion*, vol. 6, no. 1, pp. 63–81, 2005.

[47] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[48] R. A. Shaikh, K. Adi, and L. Logrippo, "A data classification method for inconsistency and incompleteness detection in access control policy sets," *International Journal of Information Security*, vol. 16, no. 1, pp. 91–113, 2017.

[49] R. A. Shaikh, K. Adi, L. Logrippo, and S. Mankovski, "Inconsistency detection method for access control policies," in *Proceedings of the 2010 Sixth International Conference on Information Assurance and Security (IAS)*. IEEE, 2010, pp. 204–209.

[50] A. Shaikh Riaz, K. Adi, L. Logrippo, and S. Mankovski, "Detecting incompleteness in access control policies using data classification schemes," in *Proceedings of the 2010 Fifth International Conference on Digital Information Management (ICDIM)*. IEEE, 2010, pp. 417–422.

[51] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "Grids-a graph based intrusion detection system for large networks," in *Proceedings of the 19th national information systems security conference*, vol. 1. Baltimore, 1996, pp. 361–370.

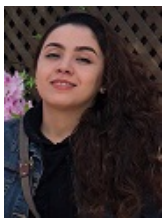
[52] M. Touma, E. Bertino, B. Rivera, D. Verma, and S. Calo, "Framework for behavioral analytics in anomaly identification," in *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII*, vol. 10190. International Society for Optics and Photonics, 2017, p. 101900H.

[53] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone, "Analysis of xacml policies with smt," in *Proceedings of the 2015 4th International Conference on Principles of Security and Trust (POST), London, UK, April 11-18, 2015*, 2015, pp. 115–134.

[54] D. Verma, S. Calo *et al.*, "Generative policy model for autonomic management," in *Proceedings of the 1st International Workshop on Distributed Analytics InfraStructure and Algorithms for Multi-Organization Federations*. IEEE, 2017.

[55] W. Xu, X. Zhang, and G.-J. Ahn, "Towards system integrity protection with graph-based policy analysis," in *Proceedings of the Data and Applications Security XXIII, 23rd Annual (IFIP) (WG) 11.3 Working Conference, Montreal, Canada, July 12-15, 2009*. Springer, 2009, pp. 65–80.

Amani Abu Jabal is working toward the Ph.D. degree in the Department of Computer Science, Purdue University. She received a BS degree in computer science from Jordan University of Science and Technology in 2007, and then an MS degree from Purdue University in 2013. Her research interests include data provenance, access control, and social network mining.



Maryam Davari is working toward the Ph.D. degree in the Department of Computer Science, Purdue University. She received an MS degree in computer science from Queen's University, Canada in 2017. She is a member of the Center for Education and Research in Information Assurance and Security (CERIAS). Her research interest includes access control and information security.

Elisa Bertino is a professor of computer science with Purdue University, and serves as director of the CyberSpace Security Lab (Cyber2SLab). Prior to joining Purdue in 2004, she was a professor and department head in the Department of Computer Science and Communication at the University of Milan. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, CA, in the Microelectronics and Computer Technology Corporation, at Rutgers University, and at Telcordia Technologies. Her recent research focuses on database security, digital identity management, policy systems, and security for web services. She received the IEEE Computer Society 2002 Technical Achievement Award, the IEEE Computer Society 2005 Kanai Award, and the ACM SIGSAC Outstanding Contributions Award. She served as EiC of the IEEE Transactions on Dependable and Secure Computing. She is a fellow of the ACM, of the IEEE, and the AAAS.

Christian Makaya is currently a researcher at IBM T.J. Watson Research Center, NY, USA. Prior to joining IBM, he was a senior research scientist at Telcordia Technologies, NJ, USA and a visiting researcher at Ericsson Research, Montreal, Canada. His work has been a catalyst behind several new initiatives and technologies resulting in the delivery of high-value capabilities to products and services. For his technical contributions, he has been recognized by several high-prestige internal awards at IBM and Telcordia. Dr. Makaya leads several technical research activities in the areas of distributed systems and analytics with the mission of delivering deep technical breakthroughs. The focus of his current research interests is on distributed AI and ML, edge computing, Internet of Things (IoT), network functions virtualization (NFV), policy-based management systems, and cyber-security. Dr. Makaya has authored numerous technical papers in peer-reviewed journals and conferences, and filled several patents. He received his Ph.D. in Computer Engineering from Polytechnique Montreal (2007). Dr. Makaya is an active member and volunteer of IEEE and serves on the Industry Outreach Board of IEEE Communication Society (ComSoc). He served as the co-chair of IEEE Young Professionals for the IEEE Princeton/Central Jersey Section.

Seraphin Calo received the MS, MA, and Ph.D. degrees in electrical engineering from Princeton University. He is a research staff member at IBM Research and currently manages the Network Science group within that organization. He is a senior member of the IEEE.

Dinesh Verma is a Research Staff Member and Department Group Manager at the IBM T. J. Watson Research Center, NY. He manages the IT and Wireless Convergence team at the center. He is the program manager for the International Technology Alliance. He received the BS degree from the Indian Institute of Technology, Kharagpur, India, in 1987 and his Ph.D. degree in 1991 from the University of California, Berkeley, in the Tenet Networking Group headed by Prof. Domenico Ferrari. He joined IBM Research in 1992 and worked on network control protocols and algorithms. He is a Fellow of the IEEE and has authored five books on the topic of networking.

Christopher Williams graduated from the University of Oxford with a First in Engineering Science, and subsequently gained his Ph.D. from Bristol University on the topic of chaotic waveforms for communications. Alongside periods in the industry (Research Manager for Fujitsu) and academia (Research Fellow at Bristol University) much of his career has been in Government defence research (Dstl and predecessors). Areas of expertise include novel waveforms, communications signal processing, dynamic spectrum access, risk based decision making, complexity, agile systems and requirements engineering. He is the chair of the 5-eyes TTCP Communications and Networking panel.