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‡
*Department of Computer Science, Purdue University, West Lafayette, Indiana, USA, 47907
{aabujaba, davari, bertino}@purdue.edu
†IBM Research, Yorktown Heights, NY, USA, 10598
{cmakaya, scalo, dverma}@us.ibm.com
‡The Defence Science and Technology Laboratory, Porton Down, Wiltshire SP4 0JQ, UK
{cwilliams}@dstl.gov.uk

***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 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 that 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. Provenance data are used to determine whether the policies meet the quality requirements. 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 supports policy evolution and the assessment of its impact on the policy quality. 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.**

I. 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 or 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 (surveyed by Bertino et al. [14]) including RBAC [51] and XACML [2].

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. [15] 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 system resources. The framework supports two analysis strategies having the goal of analyzing the static set of policies (referred to as *policy-based analysis*) and analyzing the correlated set of transactions executed by subject with their corresponding access policies (referred to as *transaction-based analysis*). However, this framework has not been implemented nor tested.

This paper extends our previous work [15] along two major directions. The first is the introduction of an additional policy analysis method based on classification techniques. The second 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 design and implementation of a policy analysis approach that utilizes a classification technique. This approach efficiently detects the "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., using kNN, Naïve Bayes, SVM, Random Forest, or Decision Tree) and combined classifiers (i.e., combining the results of multiple classifiers using probabilities of prediction votes or a majority of votes).

- An experimental comparison of the two policy analysis approaches.
- The implementation of a policy evolution approach. Based on the policy analysis, our evolution tool handles the modifications to "low quality" policies and reanalyzes them.

While implementing the framework, we were not able to obtain a large set of access control policies in a real system. Alternatively, we generated a synthesized dataset for experimental purposes. Moreover, machine learning algorithm (particularly classification techniques) sufficient dataset to efficiently learning the characteristics of the dataset. In our framework, we addressed the challenges of classification techniques at two levels: data level by sampling more dataset 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, policy lifecycle, and data provenance. Section 3 introduces several definitions underlying the policy quality requirements and proposes two data structures for policy analysis. Section 4 describes our analysis services. Section 5 describes our policy evolution services. Section 6 introduces other policy analytic services. Section 7 presents the experimental results. Section 8 discusses related work. Finally, Section 9 outlines conclusions and future work.

II. Preliminaries

In what follows, based on [15], we introduce background concepts and information needed for the subsequent developments in the paper.

A. Role-based Access Control

The role based access control (RBAC model) consists of four basic components [51]: *users*, *roles*, *permissions*, and *sessions*. A role represents an organizational function within a given domain (e.g., a coalition or an enterprise). Roles are granted permissions required for the execution of their functions. A permission consists of the specification of a protected object and an action, defined on the object, and indicates that the action can be executed on the object. Users (e.g., humans, devices) represent the active entities that execute actions on the protected objects. Users are assigned to roles and thus inherit the permissions of their assigned roles. A session is a sequence of accesses executed by a user under one or more roles. When a user becomes active in the system, it establishes a session and, during this session, it uses one or more roles among the ones it has been assigned to.

The RBAC model definition 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. [51]) formally defines the RBAC model.

Definition 1 (Role-based Access Control Model [12])**.** 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 $\geq$).

Notice that in Definition 1, we associate a "sign" with each permission to support positive and negative authorizations. A positive authorization, denoted by the '+' sign in the permission, is an authorization that allows the role, specified in the authorization, to execute the action on the object specified in the authorization permission. By contrast, a negative authorization, denoted by the '-' sign in the permission, specifies that the role, specified in the authorization, cannot execute the action on the object specified in the authorization permission. Negative authorizations are particularly useful when dealing with large sets of protected objects organized according to hierarchies. In such contexts, negative authorizations combined with authorization propagation along objects hierarchies support the specification of exceptions, by which one can, for example, allow a role to read an entire directory with the exception of a given file in the directory. Authorization propagation and positive and negative authorizations have been widely investigated [47], and also introduced in access control systems of commercial products. An example is the access control model of SQL Server in which authorizations propagate along securable hierarchies and in which negative authorizations can be specified by means of the DENY authorization command [1]. In our contexts, negative authorizations are also critical in order to provide boundaries to actions that cognitive autonomous devices can execute.

B. Policy Life Cycle

We assume an iterative policy lifecycle composed of three stages (specification, enforcement, analysis) which form the basis for the deployment of the policies in the system of interest. In the policy specification stage, the administrator coordinates with the representative system users to determine the policies to be enforced. The policy enforcement stage is the one in which policies are applied to control the actions executed by the system users on the protected objects. As the environments we deal with are characterized by dynamic contexts and situations, it is often the case that policies may have to evolve in order to adapt to changes. Therefore, during the policy enforcement stage, additional information is collected that is used by the next stage in the policy lifecycle, that is, the analysis stage. Such a stage evaluates the quality of the current policy set based on the information collected through the enforcement stage and suggests possible changes to the current set of policies. The evolved policies are then deployed and enforced.

C. 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 [3], and Secure Interoperable Multi-Granular Provenance (SimP) [4]). In this paper we used SimP [4]. 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 *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 I defines notations for SimP processes and policies. 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).

III. 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.

[1]Integrating our framework with different provenance models is out of the paper scope

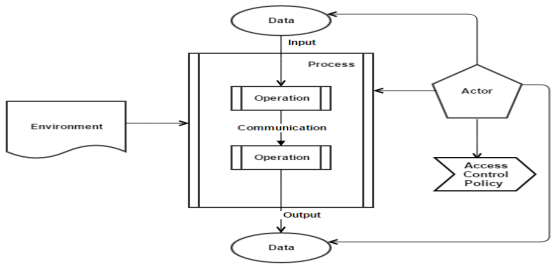


Fig. 1: SimP provenance model [4]

TABLE I: SimP Entity Notations

Notation	Description
SimP.Processes	A set of processes captured by SimP provenance
SimP.Policies	A 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	An operation executed by PR process
PR.Data	an operation executed by PR process
PR.User	User who executed the process
PL.Operation	An operation controlled by PL policy
PL.Data	Data object controlled by PL policy
PL.User	User whose access is controlled by PL policy

A. Policy Quality Requirements

We illustrate the policy quality requirements by the following running example (adapted from [56]).

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;

TABLE II: 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	$-$

(v) and assign a worker and mule for a delivery request. The access control policies related to these transactions are listed in Table II. 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 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 policy enforcement stage that then require conflict resolution strategies [13] be applied. Minimizing the inconsistencies is thus critical to reduce the need for conflict resolutions activities.

Example: As shown in Table II, 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

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

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 II. 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 manger 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 [2], 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 II. 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

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

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 II. 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.

B. 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 executed transactions in the system. Therefore, to support such searches, we introduce the following structures.

1) Policy Tree: The set of access control policies ACP is represented by a balanced multi-way tree structure (referred to as *policy tree*) (see Fig. 2 for an example[2]). A policy tree contains four types of nodes: role, action, object, and sign. The root node consists of role nodes which represent all distinct roles in ACP. Each role node points to a set of action nodes which represent the authorized actions to the role. Furthermore, each action points to a set of objects on which the corresponding role is allowed to perform the corresponding actions. Each object node points to a leaf node which represents the authorization permission sign. The leaf node is augmented with two additional values: *Policy ID* which refers to the identifier of the policy represented by the corresponding path, and *Counter* which represents how many times the policy was enforced. Each node in the tree has unlimited fan-out (i.e., the maximum number of children) so the constructed structure is a wide and shallow tree. Heuristically, for constructing the policy tree, we choose role nodes to be stored in the first level to minimize the number of tree branches; hence searching the tree becomes more efficient. Furthermore, each node in the tree contains a hash table where the main elements in the node are the keys while the key values are the corresponding pointers to the successor nodes. This enhances the efficiency of the searches on the policy tree for a certain access control policy as it avoids the sequential inspection of all child nodes in each node.

[2]The figure is based on the policies with odd identifiers in Table II to simplify the figure.

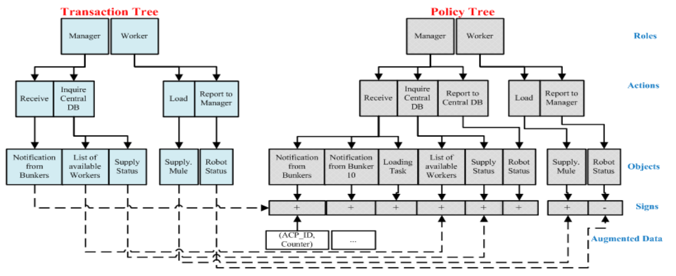


Fig. 2: Policy analysis structures: policy tree (right) and transaction tree (left)

In distributed environments, each party might have its own local policy set; hence a set of policy trees are constructed.

2) Transaction Tree: The set of transactions that are executed in the system is represented by another balanced multi-way tree (referred to as *transaction tree*) (see Fig. 2 for an example). The structure of this tree is similar to that of the policy tree, but the leaf nodes are a set of objects where each leaf node is augmented with a counter (indicates how many times a transaction recurred) and pointer(s) to leaf node(s) in the policy tree(s). A leaf node in the transaction tree is mapped optimally to one leaf node in the policy tree. However, it might be mapped to more than one leaf node (e.g., in the case of the existence of redundant policies).

The transaction tree is populated with the transactions which are captured by the provenance framework. Hence, the provenance repository is periodically queried about recent transactions in order to maintain the transaction tree. The maintenance of the transaction tree is illustrated in Algorithm 1. The retrieved provenance records include a set of processes where each process record contains a set of operations and each operation executed by user and manipulated a set of data. An operation which is executed by a user and manipulates data is mapped to a transaction t (i.e., user, object, and action). Since SimP captures the role assigned to the user at the operation execution time, SimP identifies the role which was assigned to the user who executed a transaction. A transaction tuple along with its corresponding role is inserted into the transaction tree. Next, we search the policy tree to identify the policy which potentially controls the corresponding transaction. If a policy is found, we connect the leaf node of a transaction path in the transaction tree with the leaf node of the matching policy path in the policy tree and update their counters accordingly.

TABLE III: 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 policy or transaction tree.
f_A	The number of unique actions which represents the maximum fan-out of an action node in the policy or transaction tree.
f_O	The number of unique objects which represents the maximum fan-out of an object node in the policy or transaction tree.

Algorithm 1 Maintain Transaction Tree

```
Require: $TransactionTree$, $PolicyTree$, $SimP$
1: for $\exists PR_k \in$ query($SimP.Processes$) do
2:     Define $t\ as\ Transaction$
3:     $t.a = PR_k.Operation,\ t.o = PR_k.Data,\ t.u = PR_k.User$
4:     $r = PR_k.User.Role$
5:     $leaf_1 =$ Insert($r, t.a, t.o$) $into\ TransactionTree$
6:     $leaf_2 =$ Search($r, t.a, t.o$) $into\ PolicyTree$
7:     $leaf_1.counter = leaf_1.counter + 1$
8:     if $leaf_2 \neq \phi$ then
9:         $link(leaf_1, leaf_2)$
10:        $leaf_2.counter = leaf_2.counter + 1$
11:    end if
12: end for
```

3) Time Complexity of Structure Construction: For discussing the time asymptotic analysis for constructing the policy and transaction trees, we used the notations listed in Table III.

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 $\mathcal{O}(n \log_f n)$. As shown in Algorithm 1, while constructing the transaction tree, 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.

IV. Policy Analysis Services

Our framework supports two types of analysis: structure-based and classification-based. The structure-based analysis requires maintaining two data structures for analyzing all policies and find any "low quality" policy. However, this approach is expensive. Alternatively, the classification-based analysis approach learns the characteristics of policies of each type of "low quality" policies. The classification-based approach is able to quickly predict the quality of a policy, but inaccurate prediction might happen.

A. 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*"). 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.

In the distributed environment that has separate policy sets defined for each party, the transaction-based analysis makes it possible to determine which policies belong to different parties while the policy-based analysis is local to each party in the system. In what follows, we describe our approaches.

1) Policy-based Analysis: In this service, we aim to evaluate the policies in order to detect: inconsistency, redundancy, and irrelevancy. The service utilizes the policy tree structure.

The policy-based analysis is described in Algorithm 2. Lines 2-11 traverse every path in the policy tree from the root (role node) to an object node to validate a set of conditions as follows.

- If a path branches to two different signs, this flags for inconsistency (lines 4-5).
- If the leaf node of a path is augmented with multiple policy IDs, this shows a case of redundancy (lines 7-9).
- If the counter value of a leaf node is zero, this flags for irrelevancy (lines 10-11).

Furthermore, lines 12-19 descends the tree from the root to the action nodes to check if there are object nodes which are composite of each other to assess for inconsistency and redundancy.

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 $\mathcal{O}(n \log_f n)$.

2) Transaction-based Analysis: In this service, we aim to evaluate the policy set to detect: inconsistency, redundancy, incompleteness, and exceptions. Unlike the policy-based analysis, this service utilizes the transaction tree primarily and explores the policy tree.

Algorithm 2 Policy-Based Analysis

```
Require: $PolicyTree$
 1: $\mathcal{L}_{INCON} \longleftarrow \{\}, \mathcal{L}_{IRR} \longleftarrow \{\}, \mathcal{L}_{RED} \longleftarrow \{\}$
 2: for $path\ (r,a,o) \in PolicyTree$ do
 3:     $\mathbb{N} = \{\forall g \mid g\ is\ leaf\ node\ for\ (r,a,o)\}$ // list of sign nodes
 4:     if $g_i \in \mathbb{N} \wedge g_j \in \mathbb{N} \wedge i \neq j \wedge g_i \neq g_j$ then
 5:         $\mathcal{L}_{INCON} \longleftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
 6:     end if
 7:     for $g_i \in \mathbb{N}$ do
 8:         $ID = \#of Policy\ IDs\ augmented\ with\ g_i$
 9:         if $ID > 1$ then
10:             $\mathcal{L}_{RED} \longleftarrow \mathcal{L}_{RED} \cup \{g_i\}$
11:         end if
12:         if $g_i.counter = 0$ then
13:             $\mathcal{L}_{IRR} \longleftarrow \mathcal{L}_{IRR} \cup \{g_j\}$
14:         end if
15:     end for
16:     Traverse $each\ path\ (r,a)$ in $PolicyTree$
17:     $\mathbb{N} = \{\forall o \mid o\ is\ child\ node\ for\ (r,a)\}$
        / list of object nodes
18:     if $o_i \in \mathbb{N} \wedge o_j \in \mathbb{N} \wedge i \neq j \wedge o_i \subseteq o_j$ then
19:         $\mathbb{N}_1 = \{\forall g \mid g\ is\ leaf\ node\ for\ (r,a,o_i)\}$ // list of sign nodes
20:         $\mathbb{N}_2 = \{\forall g \mid g\ is\ leaf\ node\ for\ (r,a,o_j)\}$ // list of sign nodes
21:         if $g_i \in \mathbb{N}_1 \wedge g_j \in \mathbb{N}_2 \wedge g_i \neq g_j$ then
22:             $\mathcal{L}_{INCON} \longleftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
23:         end if
24:         if $g_i \in \mathbb{N}_1 \wedge g_j \in \mathbb{N}_2 \wedge g_i = g_j$ then
25:             $\mathcal{L}_{RED} \longleftarrow \mathcal{L}_{RED} \cup \{g_i, g_j\}$
26:         end if
27:     end if
28: end for
29: return $\mathcal{L}_{INCON}, \mathcal{L}_{IRR}, \mathcal{L}_{RED}$
```

The transaction-based analysis is described in Algorithm 3. The algorithm traverses every path in the transaction tree and explores its corresponding policies in the policy tree. While traversing the tree, it validates the following conditions:

- If a transaction path does not point to any policy in the tree, this flags for incompleteness (lines 4-5).
- If a transaction points to two policies which have different signs, this shows a case of inconsistency (line 6-7).
- If a transaction points to one policy path, but the path is augmented with multiple policy IDs, this shows a case of redundancy (lines 9-11).
- If a transaction points to a policy path where the sign of the policy is '-', this flags for exceptions (lines 12-13).

The objectives of our analysis services are summarized in Table IV.

Time Complexity: The transactions-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 transaction-based analysis approach is $\mathcal{O}(n)$.

Algorithm 3 Transaction-Based Analysis

```
Require: $TT$: $TransactionTree$, $PT$: $PolicyTree$
 1: $\mathcal{L}_{INCON} \leftarrow \{\}$, $\mathcal{L}_{INCOMP} \leftarrow \{\}$, $\mathcal{L}_{RED} \leftarrow \{\}$,
    $\mathcal{L}_{EXP} \leftarrow \{\}$
 2: Traverse $each\ path\ (r,a,o)$ in $TT$
 3: $\mathbb{N} = \{\forall g \mid g\ is\ pointed\ by\ o\ , g\ is\ a\ leaf\ node\ \in\ PT\}$
 4: if $\mathbb{N} \equiv \phi$ then
 5:     $\mathcal{L}_{INCOMP} \leftarrow \mathcal{L}_{INCOMP} \cup \{(u,a,o)\}$
 6: end if
 7: if $g_i \in \mathbb{N} \wedge g_j \in \mathbb{N} \wedge i \neq j \wedge g_i \neq g_j$ then
 8:     $\mathcal{L}_{INCON} \leftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
 9: end if
10: for $g_i \in \mathbb{N}$ do
11:     $ID = \#of Policy\ IDs\ augmented\ with\ g_i$
12:     if $ID > 1$ then
13:         $\mathcal{L}_{RED} \leftarrow \mathcal{L}_{RED} \cup \{g_i\}$
14:     end if
15:     if $g_i \equiv -$ then
16:         $\mathcal{L}_{EXP} \leftarrow \mathcal{L}_{EXP} \cup \{(u,a,o)\}$
17:     end if
18: end for
19: return $\mathcal{L}_{INCON}, \mathcal{L}_{INCOMP}, \mathcal{L}_{RED}, \mathcal{L}_{EXP}$
```

TABLE IV: The Objectives of Policy Analysis Services

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

B. Classification-based Analysis

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

1) Background on Classifiers: Here, we review a set of well-known classifiers. First, the k-Nearest Neighbors (kNN) classifier is the simplest one which relies on the kNN search on the training dataset. The class of an object is identified based

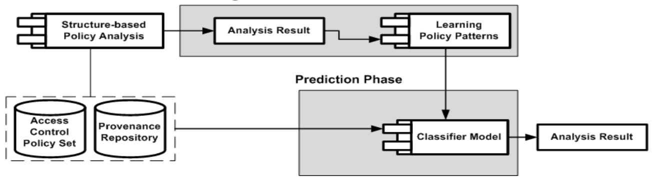


Fig. 3: The Pipeline of Classification-based Policy Analysis

on the majority voting of its kNN. Second, Naïve Bayes is a probabilistic classifier based on Bayes theorem [48] which enables calculating a posterior probability for each class at prediction. Third, Support Vector Machine (SVM) is designed for binary classification. For multi-label classification, SVM is generalized in two schemes: one-versus-all and one-versus-one. Our work considers the one-versus-one scheme in which each pairwise class group is chosen as a two-class SVM each time.

Some classifiers are tree-based such as in the Decision Tree (DT) classifier. A DT organizes a series of test conditions in a tree structure [46] where the internal nodes represent criteria for the attributes of each class, and the leaf nodes are associated with class labels. Random Forest, an extension of DT, includes a set of DTs, and the output of classification is defined by the leaf node that receives the majority of votes [49].

2) 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. The historical analysis results are obtained from the policy-based analysis and transaction-based analysis since neither of them is able to support the analysis with respect to all the quality requirements (as explained in Table IV). 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. Then, the classifier utilizes the model for predicting the class of new policies.

There are however two main challenges that are associated with 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 instantiate a synthesized set of 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.

3) Classification Approaches: Here, we present two classification schemes which we use in our framework.

One Classifier (OC): Among the classifiers discussed earlier, the OC approach adopts one classifier 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

Algorithm 4 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:         $\alpha_{ij}$ = 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
```

Algorithm 5 Policy Evolution Service

```
1: Let $acp_i$ $:=<$ $r, p.a, p.o, p.sign$ $>$ denote a policy to be changed, $M$ denote the policy evolution type $\{Add$, $Update$, $Delete\}$, $D$ denote the set of access control policies, and $T$ denote the policy tree constructed on $D$.
2: Update $D$ given $acp_i$
3: $P$ := Find Affected Policies($T$, $acp_i$, $M$)
4: Update $T$ given $acp_i$
5: Analyze $P$
```

with different classifiers may potentially result with 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 [50] (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 4.

V. Policy Evolution Services

A dynamic system often requires modifying its access control policies. The modifications include adding a new access control policy, and modifying or deleting an existing policy. However, in order to maintain the quality of the policy sets, it is critical that the quality of these policy sets be re-evaluated

Algorithm 6 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 $M \equiv Add$ then
3:     $Path < root \ldots node_j >$ = DepthFirstSearch($T$, $acp_i$)
4:     $P = \{\forall acp_j \mid node_j = acp_j.r \vee node_j = acp_j.p.a \vee node_j = acp_j.p.o \vee node_j = acp_j.p.sign\}$
5: else if $M \equiv Update$ then
6:     if updating $acp_i.r$ then
7:         $x = < acp_i.r >$
8:     else if updating $acp_i.p.a$ then
9:         $x = < acp_i.r, acp_i.p.a >$
10:    else if updating $acp_i.p.o$ then
11:        $x = < acp_i.r, acp_i.p.a, acp_i.p.o >$
12:    else if updating $acp_i.p.sign$ then
13:        $x = < acp_i.r, acp_i.p.a, acp_i.p.o, acp_i.p.sign >$
14:    end if
15:    $Path < root \ldots node_j >$ = DepthFirstSearch($T$, $x$)
16:    $P = \{\forall acp_j \mid node_j = acp_j.r \vee node_j = acp_j.p.a \vee node_j = acp_j.p.o \vee node_j = acp_j.p.sign\}$
17: else if $M \equiv Delete$ then
18:    $P = \{\forall acp_j \mid acp_j.r = acp_i.r\}$
19: end if
20: Return $P$
```

whenever the sets are modified. It is clear, however, that re-evaluating the quality of entire policy sets can be very expensive. Thus, the policy evolution service aims at re-evaluating only the policies affected by the modifications. Towards this, when updating the policy set, the policy evolution service first identifies the policies affected by the current modification. It then re-evaluates such policies either by traversing the policy tree partially or performing the classification-based analysis on the batch of affected policies.

After a new policy acp_i, whose components are $acp_i.r$; $acp_i.p.a$; $acp_i.p.o$; and $acp_i.p.sign$, is added to the policy set, the policy tree is first searched using a depth-first-search to find a path (i.e., starting from the root to a node j) whose nodes match the components composing acp_i until a non-matching node is encountered. For complementing the found path, new nodes representing the missing components of acp_i are connected to the ending node of the path. After that, all policies which share the identified path (i.e., policies reachable from the internal node j) are analyzed.

Modifying a policy acp_i involves changing the values of one or more components composing acp_i. To analyze the policies using the policy tree, we first identify the modified component x corresponding to the highest level in the tree hierarchy among all modified components. After that, the policy tree is searched using depth-first-search to find a path whose nodes matching the components composing acp_i until finding a node matching x. Subsequently, all policies branched from the internal node x are analyzed.

Among the policy modifications (i.e., add, update, or delete), deleting a policy acp_i results in the largest number of affected policies. Hence, the required evaluation is the most expensive one. To analyze policies using the policy tree, all policies branching at the role matching $acp_i.r$ should be re-evaluated. In the worst case, when all policies in the policy set are related to only one role, all policies in the policy tree will be re-assessed using the quality requirements. Policy evolution services are defined in Algorithms 5 and 6.

VI. The ProFact Framework

We now introduce our policy analysis framework which utilizes provenance metadata to aggregate the analysis results. As shown in Fig. 4, 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 actions executed 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 the two tree structures (i.e., policy and transaction trees) which abstract the necessary information for the analysis phase. The analysis phase includes two types of analysis approaches: structure-bases 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 a pipeline of four steps: updating a policy in the policy set, identifying the policies correlated with the updated policy, updating the policy tree, and then performing one of the policy analysis approaches only on the affected policies. The results of the re-analysis are also aggregated into the *policy analysis repository*.

Furthermore, ProFact includes a query services component supporting 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 policies affected by a specific type of policy change or find the policy

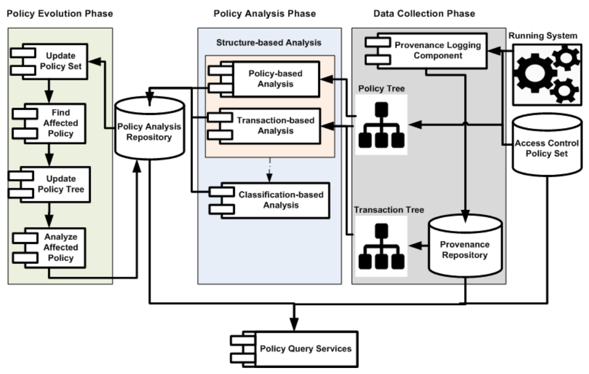


Fig. 4: The Infrastructure of the ProFact Framework

TABLE V: 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

components most affected by a specific type of policy change; these queries are based on the historical runs of the evolution service. In particular, searching through the history of policy changes. Examples include: find the object which was most affected by policy deletion, and what is the change operation that affected the largest number of policies.

VII. Experiments

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

A. Dataset and Settings

To evaluate the policy analysis approaches, we conducted several experiments using three synthetic datasets generated by a random dataset generator (referenced as *DS1*, *DS2*, and *DS3*). Table V shows the size of each dataset in terms of the number of roles, protected objects, access control policies, and transactions.

We implemented the prototype infrastructure of ProFact in Java 1.7. For the classifiers adopted in our classification-based analysis, we used the Java Weka library [29]. All classifiers are trained on 70% of the dataset using 10-fold cross validation. 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.

B. Pre-processing Time for the Analysis Approaches

For each approach, we collected the total time for building the underlying structures or model. For the structure-based approach, the construction time refers to the time for organizing the access control policy sets and transaction sets in the policy and transaction trees, respectively. For the classification-based approach, we report the time taken for learning the characteristics of a training dataset and generating a trained model. Fig. 5 shows the construction time (in base-10 logarithmic scale) for the two variants (i.e., policy-based and transaction-based) of the structure-based approach using the three datasets. In general, the construction of the structures for the transaction-based analysis takes longer time than for 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. On the other hand, for the classification-based approach, the training time for the underlying classifiers varies dramatically (see Fig. 6[3]). The time required for building the kNN and Naïve Bayes classifiers is negligible since the kNN classification algorithm does not require building a trained model and the Naïve Bayes classification algorithm requires only aggregating numbers (e.g., count and posterior probability) for each category. Compared with kNN and Naïve Bayes, the time required for the Decision tree and Random Forest is higher since generating trained models for these classifiers involves building a conditional tree based on the properties of the training dataset. Among all classifiers, SVM requires the longest training time. For training an SVM classifier in the case of a binary classification (i.e., only two classes), SVM finds mathematically a decision boundary between the data distributed in the two classes. In multi-class classification, the SVM algorithm is adapted by utilizing further optimization algorithms to find the proper decision boundaries among all classes; hence increasing the training time.

C. Analysis Results

1) Structure-based Analysis: ***Performance:*** Fig. 7 shows the analysis time for detecting inconsistent and redundant policies[4] using both policy-based and transaction-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. Both 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 objects 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.

Efficiency: Both policy-based and transaction-based approaches were able to detect all inconsistent and redundant policies. Since irrelevant policies do not have corresponding

[3]We omitted the training time for $CC - Majority\ Voting$ and $CC - Max\ Probability$ to simplify the figure. The training time of CC is the total of training time of individual classifiers.

[4]We discarded the other types of "low quality" policies and considered only the types of policies that can be detected by the two variants of the structured-based approach

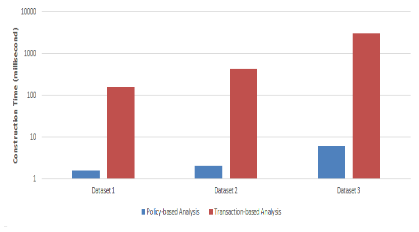


Fig. 5: Construction Time of Structure-based Approaches

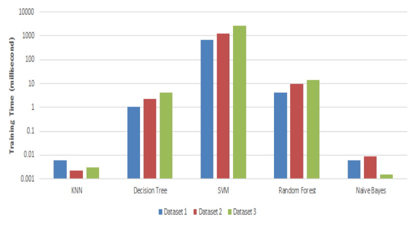


Fig. 6: Training Time of Classification-based Approaches

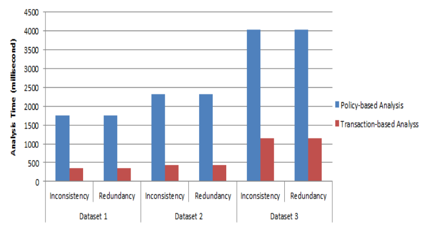


Fig. 7: Analysis Time of Structure-based Approaches

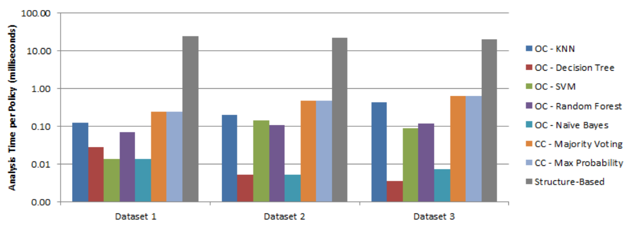


Fig. 8: Analysis Performance of Classification-based Approaches

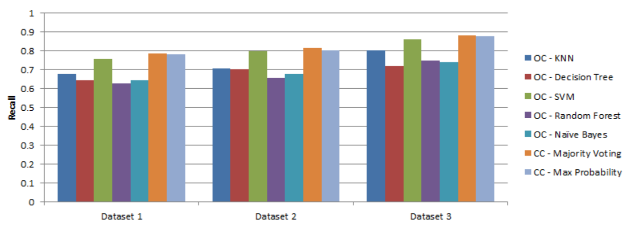


Fig. 9: Analysis Efficiency of Classification-based Approaches

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.

2) Classification-based Analysis: ***Performance:*** Fig. 8 shows the average analysis time per policy (in base-10 logarithmic scale) using both analysis approaches (structured-based and classification-based) [5] 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 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 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 the 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 values which indicate the ratio of the correctly classified policies among the policies classified to a certain category (as formulated in Eq. (1)).

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \qquad (1)$$

For calculating recall, we used the output of the structure-based analysis results as the ground truth. Fig. 9 shows the recall of both classification schemes (OC and CC) across the three datasets. In general, all classification approaches achieve recall values above 60%. The recall varies among the OC approaches depending on the adopted classifier. In particular, the efficiency of OC varies based on the trained model and the chosen classifier. For example, Decision Tree was the worst using *DS3*, but its recall improved in the other datasets. Consequently, using the OC scheme is not suitable for our framework. On the other hand, the CC scheme achieves the best recall using all datasets. 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.

VIII. 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.

A. 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

[5]The analysis time of the structure-based analysis represents the maximum time between the policy-based or transaction-based approaches since none of these approaches is able to detect all types of low-quality policies

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. [57] proposed methods to detect inconsistency among XACML policies. Regarding 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], [54]). 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]. 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], [57]). Our framework includes the policy evolution services which analyzes the impact of changing one of the policies on the quality of the correlated policies of the changed policy. However, for analyzing policies, our framework utilizes the matching metric which is a simple similarity policy for identifying the correlated policies. Our framework can be enhanced by utilizing other similarity metrics.

B. Methods for Policy Analysis

Various methods have been proposed for policy analysis including formal methods, model checking, data mining, and structure-based. 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], [57], and binary decision diagrams [19], [27], [34], [44]. Regarding data mining methods, Shaikh et al. [52]–[54] and Aqib et al. [9] proposed several approaches for policy analysis using 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. [59], Staniford et al. [55], 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.

IX. 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 supports various types of query services which convey detailed information about the system environment in the context of transactions and access control policies. It 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.

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 [58] by which devices are given abstract policies and can then autonomously refine and adapt them by using their own analytic services.

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.

References

[1] "Authorization and permissions in sql server." [Online]. Available: https://msdn.microsoft.com/en-us/library/bb669084(v=vs.110).aspx

[2] "Extensible access control markup language (xacml) version 3.0." [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

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

[4] A. AbuJabal 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.

[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, 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.

[13] 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 *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1876–1886.

[14] 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.

[15] E. Bertino, A. A. 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.

[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 *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. 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] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[47] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems," *ACM Transactions on Database Systems (TODS)*, vol. 16, no. 1, pp. 88–131, 1991.

[48] I. Rish, “An empirical study of the naive bayes classifier,” in *IJCAI (EMP AI Workshop)*, vol. 3, no. 22. IBM, 2001, pp. 41–46.

[49] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, “Rotation forest: A new classifier ensemble method,” *PAMI*, vol. 28, no. 10, pp. 1619–1630, 2006.

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

[51] 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.

[52] 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.

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

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

[55] 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.

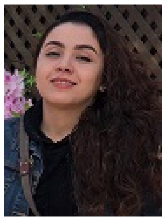
[56] M. Touma, E. Bertino, B. Rivera, D. Verma, and S. Calo, “Framework for behavioral analytics in anomaly identification,” in *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII*, vol. 10190. International Society for Optics and Photonics, 2017, p. 101900H.

[57] 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.

[58] 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.

[59] 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.

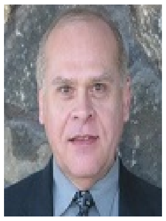
Amani Abu Jabal is working toward the Ph.D. degree in the Department of Computer Science, Purdue University. She received the BS degree in computer science from Jordan University of Science and Technology in 2007 and then the MS degree from Purdue University in 2013. Her research interests include data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