Provenance-based Analytics Services for Access Control Policies

Elisa Bertino[#1], Amani Abu Jabal[#2], Seraphin Calo[*3], Christian Makaya[*4], Maroun Touma[*5], Dinesh Verma[*6], Christopher Williams[‡7]

[#] *Dept. of Computer Science, Purdue University, West Lafayette, IN, USA*

{[1]bertino,[2]aabujaba}@purdue.edu

[*] *IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA*

{[3]scalo,[4]cmakaya,[5]touma,[6]dverma}@us.ibm.com

[‡] *The Defence Science and Technology Laboratory, Porton Down, Wiltshire SP4 0JQ, UK*

[7] *cwilliams@dstl.gov.uk*

***Abstract*—Successful collaborations require information and resource sharing and thus adequate access control policy management systems that control sharing among the collaborating entities. Such management systems need to be flexible in order to adapt to different environments and thus be able to support access control policy evolution. However, when dealing with large sets of evolving policies it is critical that policies meet certain "*policy quality requirements*". Specifically, policies of interest must be up-to-date, complete, free of inconsistencies, relevant. In this paper, we propose an approach to analyze policies in order to determine whether policies meet such requirements. Our approach is based on the use of provenance techniques that collect comprehensive data about actions executed by users in the context of workflows, that is, sets of tasks executed according to some ordering by users. Provenance data are used by services that support various types of analysis to determine whether the policies of interest verify the quality requirements.**

Keywords-Access Control Policies, Provenance, Policy Analysis, Policy Quality

I. INTRODUCTION

Collaborative activities of the future will be increasingly carried out by teams including, in addition to humans, cognitive autonomous mobile devices, such as drones, self-driving cars, and robots [12]. Such collaborative activities will also include other forms of autonomous systems, such as self-healing networks (collaboration of networking devices), self-configuring sensor systems (sensors collaborating to provide global environmental pictures) and cognitive radios (dynamically accessing the spectrum and collaborating to maximize spectrum utilization). Those advances are the result of recent technological developments in many different areas, including cognitive computing, autonomous agents, sensors, IoT, embedded systems, and robotics. As with conventional human-only collaborative activities, secure information sharing is a critical requirement for the success of collaborations. Secure information sharing refers to the ability to assure that accesses to information by the collaborating parties comply with organizational policies concerning the sharing of information.

A key mechanism for secure information sharing is represented by access control. Access control policies typically specify which subject (e.g., user, process, and application) can access which resources (e.g., files) for performing which actions (e.g. read, write). Access control has been widely investigated and many access control models have been proposed, including models taking time and location into account [13] [14], and models specific for privacy-sensitive data [15]. Standards for access control models have also been proposed, such as RBAC [2] and XACML [11]. Access control mechanisms are also embedded in many different systems, ranging from operating systems to database management systems. We refer the reader to [1] for a survey of access control models.

A critical issue in the deployment of an access control system is the specification of suitable access control policies. Such policies are the main input for access control decisions as, whenever an access request to a protected resource is issued, the access control mechanism compares the request against the policies to determine whether the access should be allowed or denied. It is thus obvious that a critical requirement is to ensure that such policies are of "good quality". Low quality access control policies would lead to wrong access control decisions that then may lead to situations in which a party, rightfully requiring a data item, is denied access to the data item, or the data is released to unauthorized parties. In other cases, the access control system may not even have a policy concerning certain requests which, depending, on the specific access control mechanism used, may lead to uncertainties concerning the outcome of the request.

The deployment of access control mechanism in the context of collaborations carried out by autonomous cognitive devices increases the difficulty of assuring the quality of policies. Such devices have often to operate in open, uncertain, complex and ambiguous environments which may quickly and dynamically change [10]. In such a context devising in advance all possible access control policies that are needed and making sure that they are of good quality is very difficult, if at all possible. We thus need a different approach by which the policies may have to

dynamically evolve based on situations encountered by the devices.

We thus focus on an approach (and necessary supporting infrastructure) to describe analytics that allow the quality of policy sets to be assessed, and further support the dynamic modification of policies. A critical issue is represented by the identification of suitable metrics for assessing the quality of policies. Thus, a first contribution of our work is the definition of several quality metrics for policies, and the introduction of data structures to aggregate data required for the evaluation of policies with respect to such metrics. A second issue is the collection of data required by the policy metrics. In particular, some metrics may require analyzing actions executed by the entities in the system of interest. To address such an issue, we propose using a provenance system as logging tool for recording fine-grained details of such actions. Such data are then used by policy analytic services. Such services comprise two strategies having, respectively, the goal of analyzing the static set of policies (referred to as *policy-based analysis*) and analyzing the correlated set of transactions executed by users with their corresponding access policies (referred to as *transaction-based analysis*). In addition, our services also include different types of query to retrieve information about the correlation between policies and the executed transactions as well as aggregated policy analytics results.

The rest of the paper is organized as follows. Section II provides background information on access control, policy lifecycle, and data provenance. Section III introduces several definitions underlying the policy quality requirements and proposes two data structures for policy analysis. Section IV describes our analysis services. Section V introduces our policy analytic service framework. Section VI discusses related work. Finally, Section VII outlines conclusions and future work.

II. PRELIMINARIES

In what follows 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 [2], [3]: *users*, *roles*, *permissions*, and *sessions*. A role represents an organizational function within a given domain, such as 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, whereas the role function establishes a mapping between a session and 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. [3]) formally defines the RBAC model.

Definition 1 (Role-based Access Control Model). 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., $PA(r_i \in R) \subseteq R \times P$). Instances of function PA are referred to as *authorizations.*
- UA is the user assignment function that assigns users to roles (i.e., $UA(u_i \in U) \subseteq U \times R$).
- The *user* function assigns each session to a single user (i.e., $user(s_i \in S) \rightarrow u_i \in U$).
- The *roles* function assigns each session to a set of roles (i.e., $roles(s_i \in S) \subseteq \{r \mid (\text{user}(s_i \in S), r) \subseteq UA\}$).

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 [6], 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 [16]. 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 life cycle (see Fig. 1) 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.

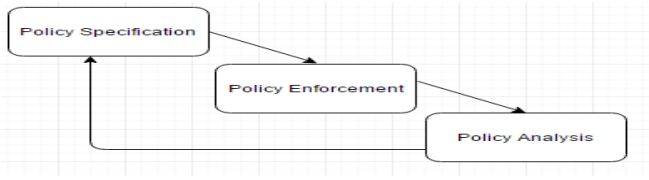

Figure 1. Policy life cycle

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) [7], PROV [8], and Secure Interoperable Multi-Granular Provenance (SimP) [9]). As shown in Fig. 2, 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 1 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).

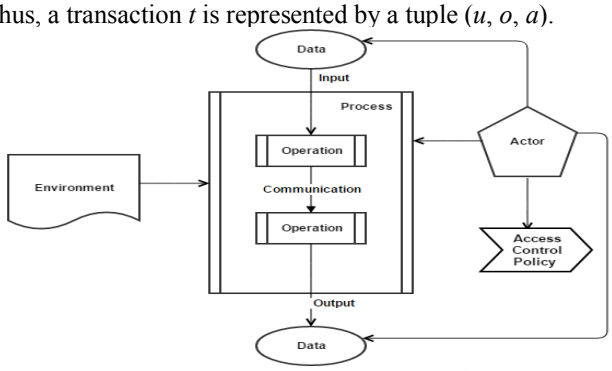

Figure 2. SimP provenance model

TABLE 1. 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* ∈ *SimP.Processes*)
PL	An access control policy record (i.e. *PL* ∈ *SimP. Policies*)
PR.Operation	an operation executed by PR process
PR.Data	Data object manipulated 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

III. POLICY ANALYTICS METRICS AND STRUCTURES

A. Policy Quality Requirements

We illustrate the policy quality requirements by the following running example (adapted from [4]).

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.

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	-

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

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

Inconsistency refers to the situation in which for the same access by a 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 [10] to 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

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

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 [11], 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

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

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 $acp_i \in ACP$ 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 sends request.

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 multi-way tree structure (referred to as *policy tree*) (see Fig. 3 for an example[1]). 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.

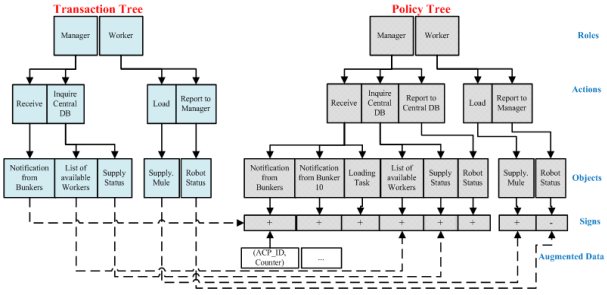

Figure 3. Policy analysis structures: policy tree (right) and transaction tree (left)

Each node in the tree has un-limited fan-out (i.e., 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.

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 multi-way tree (referred to as *transaction tree*) (see Fig. 3 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 was recurred) and pointer(s) to leaf node(s) in the policy tree(s).

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). As SimP identifies the role for 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

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

path in the transaction tree with the leaf node of the matching policy path in the policy tree and update their counters accordingly.

ALGORITHM 1. MAINTAINTRANSACTIONTREE

Input: *TransactionTree*, *PolicyTree*, *SimP*
1. ***For*** $\exists PR_k \in$ ***query***(SimP.*Processes*)
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***($leaf_2 \neq \emptyset$)
9. link($leaf_1$, $leaf_2$)
10. $leaf_2.counter = leaf_2.counter + 1$
11. ***End if***

ALGORITHM 2. POLICY-BASED ANALYSIS

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

IV. POLICY ANALYTICS SERVICES

Our policy analytic services support two types of analysis: policy-based and transaction-based. The search space of the policy-based analytic service is bounded to the access control policy set itself. Hence the policy-based analytic service is not able to assess all quality metrics (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 transaction-based mechanism follows the opposite paradigm (i.e., "*enforce first and analyze later*"). For some situation which 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 a 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.

A. 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 asses for inconsistency and redundancy.

ALGORITHM 3. TRANSACTION-BASED ANALYSIS

Input: *TransactionTree*, *PolicyTree*, *SimP*
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** *TransactionTree*
3. $\mathbb{N} = \{\forall n \mid g$ is pointed by *o*, *g* is leaf node in *PolicyTree*$\}$
4. ***if***($\mathbb{N} \equiv \emptyset$)
5. $\mathcal{L}_{INCOMP} \leftarrow \mathcal{L}_{INCOMP} \cup \{ (u, o, a) \}$
6. ***if*** $(g_i \in \mathbb{N} \wedge g_j \in \mathbb{N} \wedge i \neq j \wedge g_i \neq g_j)$
7. $\mathcal{L}_{INCON} \leftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
8. ***for*** $(g_i \in \mathbb{N})$
9. *ID* = # of Policy IDs augmented with g_i
10. ***if***(*ID* > 1)
11. $\mathcal{L}_{RED} \leftarrow \mathcal{L}_{RED} \cup \{ g_i \}$
12. ***if***($g_i \equiv$ -)
13. $\mathcal{L}_{EXP} \leftarrow \mathcal{L}_{EXP} \cup \{ (u, a, o) \}$
14. **Return** $\mathcal{L}_{INCON}$, $\mathcal{L}_{INCOMP}$, $\mathcal{L}_{RED}$, $\mathcal{L}_{EXP}$

B. 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.

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 un-equal sign, 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 3.

TABLE 3. THE OBJECTIVES OF POLICY ANALYSIS SERVICES

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

V. PROVENANCE-BASED POLICY ANALYTICS FRAMEWORK

We now introduce our policy analytics framework which utilizes provenance metadata to aggregate the analysis results. As shown in Fig. 4, the framework is based on two main phases: data collection and policy analytics. The data collection phase uses a provenance logging component, based on the SimP provenance model [9]. The SimP logging component 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 analytics phase. The analytics phase aggregates the analysis results into a separate repository referred to as *policy analysis repository*. The aggregated analysis results are used for the policy evolution process which updates the access control policy set.

Furthermore, our framework 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 metrics. 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 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 Analytics 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.

The query services utilize both provenance and policy analysis repositories. Since the provenance repository is based on the SimP model, the repository can be queried using the SimP query language (QL-SimP) [18].

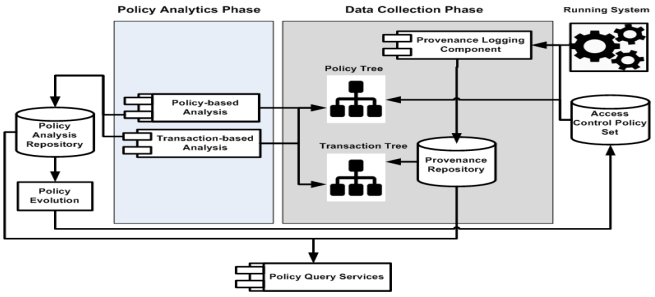

Figure 4. Provenance-based analytics framework

VI. RELATED WORK

Due to the relevance of access control policies, policy analysis has been widely investigated. Past research on policy analysis has focused on two perspectives: policy similarity assessment and policy quality metrics.

Policy similarity refers to the technique for characterizing the relationships between policies and the actions authorized by them. Existing techniques for policy similarity are mostly based on graphs, model checking or SAT-solver. For example, Koch et al. [19] proposed a graph-based similarity technique to represent policy changes; hence it is only useful during the specification phase of the policy life cycle. Fisler et al. [20] developed a software tool, called Margrave, which translates policies into multi-terminal binary decision diagrams (MTBDD). The MTBDD-based similarity technique has limitations when dealing with continuous values and dynamic policies. The EXAM framework [21] employs a similarity technique and a combination of MTBDD and SAT-solver.

Past work on policy quality metrics has investigated some of the quality metrics that we have introduced. Mankai et al. [22] proposed a method to detect inconsistent XACML policies using the Alloy logic language. Shaikh et al. [23] [24] investigated inconsistency and incompleteness in access control policies using data classification techniques. Moffett et al. [25] proposed a framework to detect the inconsistency between policies in distributed systems. Bandara et al. [27] introduced a formal characterization of policy analysis based on policy behavior and system execution traces.

To the best of our knowledge, our approach is first addressing policy quality metrics by using a provenance-based framework. Because of the use of provenance, our framework is able to assess policies with respect to new types of quality metrics (i.e., incompleteness, exceptions, and irrelevancy).

VII. CONLUSION AND FUTURE WORK

In this paper, we have proposed a set of metrics 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.

As part of future work, we will implement our policy analytics framework in the Purdue Computational Research Infrastructure for Science (CRIS) [17]. We will also extend the provenance system to capture spatial context information as this is critical for mobile cognitive devices. We will also expand our approach to support attribute-based access control policies as these policies allow one to include context-based information in policy decisions. At a broader level, we plan to extend our policy model, lifecycle, and analytics to support the notion of generative policies model [26] by which devices are given abstract policies and can then autonomously refine and adapt them by using their own analytic services. We also plan to design strategies supporting rapid re-analysis of policies based on changes, such as introduction of new roles, new protected resources, rather than using a full tree breadth and depth analysis.

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] E. Bertino, G. Ghinita, and A. Kamra. "Access control for databases: concepts and systems." Foundations and Trends in Databases 3, no. 1–2 (2011): 1-148.

[2] D. F. Ferraiolo, and D. R. Kuhn. "Role-based access controls." arXiv preprint arXiv:0903.2171.

[3] R.S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. "Role-based access control models." Computer 29, no. 2 (1996): 38-47.

[4] M. Touma, E. Bertino, B. Rivera, D. Verma, and S. Calo. "Framework for behavioral analytics in anomaly identification." Proccedings of SPIE 2017.

[5] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. "Role-based access control models." Computer 29, no. 2 (1996): 38-47.

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

[7] L. Moreau, J. Freire, J. JFutrelle, R.E. McGrath, J. Myers, P. Paulson. "The Open Provenance Model (v1.00)." Tech. Rep., University of Southampton, URL http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf, 2007.

[8] http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[9] A. Abu Jabal, and E. Bertino. "SimP: Secure interoperable multi-granular provenance framework." Proceedings of the 2016 IEEE 12th International Conference on e-Science, Baltimore, MD, USA, October 23 – 27, 2016.

[10] E. Bertino, S. Calo, M. Touma, D. Verma, C. Williams, B.Rivera. "A cognitive policy framework for next-generation distributed federated systems: concepts and research directions." Proceedings of the 2017 IEEE International Conference on Distributed Computing Systems, (ICDCS'17), Atlanta, GA, USA, June 5 – 8, 2017.

[11] Extensible access control markup language (XACML) version 2.0 (2005).

[12] S. Calo. D. Verma, E. Bertino "Distributed Intelligence – Trends in the Management of Complex Systems." Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2017, Indianapolis, IN, USA, June 21-23, 2017.

[13] E. Bertino, P. A. Bonatti, E. Ferrari. "TRBAC: A temporal role-based access control model. " ACM Trans. Inf. Syst. Secur. 4(3): 191-233 (2001).

[14] M. L. Damiani, E. Bertino, B. Catania, P. Perlasca. "GEO-RBAC: A spatially aware RBAC." ACM Trans. Inf. Syst. Secur. 10(1): 2 (2007).

[15] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, A. Trombetta. "Privacy-aware role-based access control."ACM Trans. Inf. Syst. Secur. 13(3): 24:1-24:31 (2010).

[16] https://msdn.microsoft.com/en-us/library/bb669084(v=vs.110).aspx

[17] E. Dragut, P. Baker, J. Xu, M. Sarfraz, E. Bertino, A. Madhkour, R. Agarwal, A. Mahmood, and S. Han. "CRIS—Computational research infrastructure for science." Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration (IRI), San Francisco, CA, USA, Augst 14-16 2013.

[18] A. Abu Jabal, and E. Bertino. "QL-SimP: Query Language for Secure Interoperable Multi-Granular Provenance Framework." Proceedings of the 2016 IEEE 2nd Inernational Conference on Collaboration and Internet Computing, Pittsburgh, PA, USA, November 1-3 2016.

[19] M. Koch, L. Mancini, and F. Parisi-Presicce. "On the specification and evolution of access control policies." Proceedings of the 6th ACM symposium on Access control models and technologies, Chantilly, VA, USA — May 03 - 04, 2001.

[20] K. Fisler, S. Krishnamurthi, L. Meyerovich, and M. Tschantz. "Verification and change-impact analysis of access-control policies." Proceedings of the IEEE 27th International Conference on Software Engineering, St. Louis, MO, USA, May 15-21, 2005.

[21] 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 9, no. 4 (2010): 253-273.

[22] M. Mankai, and L. Logrippo. "Access control policies: Modeling and validation." Proceedings of the 5th NOTERE Conference, Gatineau, Canada, August 29-September 1, 2005.

[23] R. Shaikh, K. Adi, L. Logrippo, and S. Mankovski. "Inconsistency detection method for access control policies." Proceedings of the 2010 IEEE 6th International Conference on Information Assurance and Security (IAS), Atlanta, Georgia, USA, August 23-25, 2010.

[24] R. Shaikh, K. Adi, L. Logrippo, and S. Mankovski. "Detecting incompleteness in access control policies using data classification schemes." Proceedings of the 2010 IEEE 5th International Conference on Digital Information Management (ICDIM), Thunder Bay, Ontario, Canada, July 5-8, 2010.

[25] J. Moffett, and M. Sloman. "Policy conflict analysis in distributed system management." Journal of Organizational Computing and Electronic Commerce 4, no. 1 (1994): 1-22.

[26] D. Verma, S. Calo, et al. "Generative Polcy Model for Autonomic Management", Proceedings of 1st International Workshop on Distributed Analytics InfraStructure and Algorithms for Multi-Organization Federations 2017, 3rd IEEE Smart World Congress 2017, August 4-8, 2017, San Francisco, CA, USA., in press.

[27] A. Bandara, S. Calo, J. Lobo, E. Lupu, A. Russo, and M. Sloman. "Toward a Formal Characterization of Policy Specification & Analysis." Proceedings of the Annual Conference of ITA (ACITA), College Park, MD, USA, September, 25-27, 2007.