A Provenance-based Access Control Model for Dynamic Separation of Duties

Dang Nguyen
Institute for Cyber Security
University of Texas at San Antonio
ytc141@my.utsa.edu

Jaehong Park
Institute for Cyber Security
University of Texas at San Antonio
jae.park@utsa.edu

Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio
ravi.sandhu@utsa.edu

***Abstract*—Dynamic Separation of Duties (DSOD) is a well-known and important concept in cyber security, which has been extensively studied in the literature. The published literature mostly assumes that necessary information for enabling DSOD constraints is readily available. As such, there has been little discussion on the tasks of capturing, storing, extracting, and utilizing necessary historical information. Since this information is often in the form of system events history, provenance data is naturally suitable as the source for DSOD-related information. Recently the notion of provenance-based access control ($PBAC$) has been formulated and a base $PBAC$ model ($PBAC_B$) together with an underlying provenance data model has been formally specified [19], [22]. Unlike Role-based Access Control where DSOD is modeled as a constraint, $PBAC_B$ directly maintains and utilizes the necessary information for DSOD enforcement. In this paper, we propose an enhanced model, $PBAC_C$, by extending both the provenance data model and the $PBAC_B$ model to enforce various DSOD policy classes identified in the literature, and go beyond these to specify novel DSOD policy classes. A proof-of-concept prototype is implemented and evaluated to demonstrate the feasibility of our approach.**

I. Introduction

Separation of Duties (SOD) has long been studied and accepted as a fundamental approach to prevent fraud and privileges misuse and abuse. Two major variations of SOD, Static SOD (SSOD) and Dynamic SOD (DSOD), are mainly discussed in the context of Role-based Access Control [26]. Static SOD has shown its usefulness and effectiveness in that domain, as evident by the multitude of researches in the literature. However, the approach has a major limitation as it mainly deals with role assignment and henceforth cannot address issues that arise within a dynamic active session. The original concept of DSOD addresses this limitation. Yet, this approach is also limited in its narrow role-centric scope as it is solely concerned with role activation. For expansion, researchers propose variations of DSOD, each of which address a separate issue. The variations include Object-based (ObjDSOD), Operational(OpsDSOD), and to the broader extent, History-based DSOD (HDSOD) [10], [27]. These approaches rely on the history information of system events, which is assumed to be readily available. However, there lacks exact specifications on how such information can be captured and utilized. Provenance data naturally provides such information and its unique characteristics enable even more sophisticated DSOD features that have not been recognized so far in the literature. Therefore, we propose a DSOD approach that utilizes provenance information. The expressive power of provenance utilization can further enable finer-grained DSOD policies and address other DSOD-related issues such as object-based conflicts and workflows.

Specifically, we build our approach on a notion called dependency path pattern that can be used to identify meaningful paths in provenance graphs [19]. This notion is used as a foundation of a base model for Provenance-based Access Control [22]. In this model, the system operation events are captured following the Open Provenance Model (OPM) style [16] provenance graph.[1] Dependency path patterns and associated abstraction names are built upon the base provenance graph that consists of vertices (*subjects, actions, objects*) and connecting causality dependency edges. While the base PBAC model, or $PBAC_B$, is capable of handling some DSOD aspects, it is limited by the types of information that are captured as provenance data. Specifically, the $PBAC_B$ model only captures the basic information of user transactions (User, Action, Objects) in provenance data and ignores other essential contextual information (e.g. timestamps, activated roles, etc.). Such simplification serves well for the purpose of demonstrating the core concepts of the model. However, in abstracting away such contextual information, the $PBAC_B$ model is restricted in its capability to handle other traditional DSOD issues as well as enhanced DSOD features that are specific to provenance-aware systems. To support both previously and newly identified DSOD, this paper extends the $PBAC_B$ model to capture and express the necessary contextual information of transaction events.

We proceed to discuss these issues and concerns in the context of the following online homework grading system (HWGS) examples. Essentially, students can upload a homework to the system, after which they can replace it multiple times before they submit the homework. Once it is submitted, the homework can be reviewed by other students or designated graders until it is graded by the teaching assistant (TA). The Professor holds the highest authority.

In this paper, we make the following novel contributions:

- Identify the potential use of provenance data in addressing traditional DSOD issues.
- Provide an extended model, $PBAC_C$, to the base PBAC model which is capable of addressing the described DSOD issues.

[1] Currently, Prov-DM [2] model extensions are being built upon OPM's foundational constructs. For our purpose, the extended Prov-DM model does not bear essential impact and hence OPM suffices.

978-1-4673-5839-2/13/$31.00 ©2013 IEEE

	Simple DSOD	ObjDSOD	OpsDSOD	HDSOD	TCE	DSOD in PBAC
Per Role	✓	✓	✓	✓	✓	✓
Per Action		✓	✓	✓	✓	✓
Per Object		✓		✓	✓	✓
Task-aware			✓	✓	✓	✓
Order-aware				✓	✓	✓
Weighted Action-aware					✓	✓
Dependency Path Pattern-aware						✓
Past Attribute-aware						✓

TABLE I. DSOD VARIATIONS AND FEATURES

- Identify new classes of DSOD concerns that have not been discussed elsewhere and are unique to $PBAC_C$ systems.
- Implement a proof-of-concept prototype, which is based on the XACML architecture [17], and provide evaluation results that demonstrate its feasibility in practical system deployment.

II. DYNAMIC SEPARATION OF DUTY: VARIATIONS

Table I shows several classic DSOD variations that are identified in the literature [10], [25], [27] and some of their unique control features. So far most of the published research on DSOD (or more generally SOD) have been in conjunction with RBAC. In particular, SOD is developed around the main concept of dividing a business task into smaller subtasks or other identifiable units such as actions, each of which is assigned to individual roles. These roles are specified into conflicting role sets from which policies can be specified and enforced to achieve SOD.

A definition of simple DSOD is given in [27]. Informally, the concept can be described as follow. Given a set of conflicting roles in a RBAC system, no single user can activate two or more roles from this set at the same time. For example, in a HWGS scenario, no user should be able to activate the roles Student and Reviewer at the same time. This notion of DSOD is limited in that it cannot support per-action control and therefore different variations are introduced [27].

One variation of DSOD is Object-based DSOD (ObjDSOD) [27]. Informally, the concept can be described as follow. Given a set of conflicting roles and a set of conflicting actions allowed in the conflicting roles, while a user may have conflicting roles activated at a given time, a user is not allowed to perform an action allowed in a role on an object if she performed a conflicting action allowed in conflicting roles on the same object. Here, the focus is emphasized on a singular data object upon which conflicting actions are considered. For example, a user should not be allowed to review the homework object that user submitted earlier while the user is allowed to review other homework.

Operational DSOD (OpsDSOD) [27] is another approach that can be described as follow. Given a set of conflicting roles and a set of actions allowed in the conflicting roles, while a user may be assigned to the conflicting roles, the user is not allowed to perform all the actions allowed by the conflicting roles if the union of the actions can complete a particular task. For example, suppose students in a business course are required to participate in an online role-playing project and provide inputs as different roles such as CEO and CTO to finish a given task. While students can acts as multiple roles in the project, one student cannot perform all the actions of roles where the roles are necessary to finish the required task.

History-based DSOD (HDSOD) [27] is defined as the combination of ObjDSOD and OpDSOD to allow finer control that are both object and task aware. More specifically, while operational DSOD prohibits users performing a set of action types that can finish a particular task, it is object-unaware and does not distinguish certain actions that are performed in a task for different objects. History-based DSOD resolves this issue by combining object-based DSOD and operational DSOD. One added characteristics of History-based DSOD is the consideration of order-dependent subtask sequences for identifying SOD conflicts. Order-dependent conflicts arise when the subtasks are allowed or obliged to occur in a certain order. For example, a homework should always be submitted before it can be reviewed.

Proposed by Sandhu [25], Transaction Control Expression (TCE) is another traditional approach toward DSOD issues which possesses its own interesting features. TCE is flexible in the sense that it can cover a wide range of DSOD features exhibited by other DSOD variations. At the same time, it also assumes a readily available system-maintained history to rely on for access decision. TCE also introduces the notion of weighted subtasks. From this point of view, each subtask is assigned an integer value as a weight. The conflict is then determined by whether the total weight of actions exceeds a particular weight threshold. An example of this can be seen in how peer-review processes are introduced in the HWGS. A fellow student can review another student's homework, whereas each review process can be assigned a weight 1. Homework can also be reviewed by designated graders, each of whose reviews can be assigned a weight 2. A TA can only grade a reviewed homework if the combined weight of all review processes exceeds 3, which can happen by various combinations of student and grader reviews.

In Table I, we also identify new features of DSOD that are unique to the $PBAC_C$ environment. Specifically, dependency path-aware refers to cases where the units of conflict are not individual roles, objects, or actions. Instead, conflicts can arise between different dependency paths, which more expressively capture meaningful relations in the system. Another feature, past attribute-aware, refers to how context information of past transactions can be utilized for DSOD constraints. Both of these unique features are further discussed later in the paper.

III. $PBAC_C$ - EXTENDED PBAC MODEL

The PBAC base model $PBAC_B$, proposed in [22], provides a foundation for enhanced access control mechanisms that utilize provenance information. While the $PBAC_B$ model certainly facilitates access control issues, including dynamic

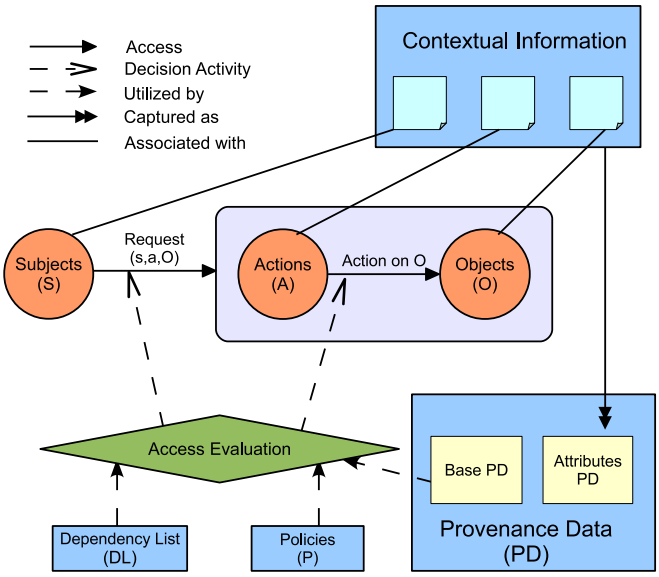

Fig. 1. Components of the $PBAC_C$ Model

separation of duties and workflow control, it is currently restrictive in the type of provenance data the model can capture, store, and extract. As a result, not all DSOD features identified in the previous section can be supported in the current $PBAC_B$ model. To address these issues, we propose an extension model, $PBAC_C$, that utilizes contextual information of subjects, actions and objects (which are captured as attributes and anchored to actions in provenance data).

A. $PBAC_C$ Model Components and Interaction

Figure 1 provides an overview of the extended provenance-based access control model we propose in this paper. We provide more detailed descriptions of various newly added and modified components, which form the essential parts of the extended model, in the subsection. We also provide an abstract discussion on the interaction of these model components in an access control context.

1) Components: Essentially, the $PBAC_C$ model includes the following components:

Subjects (S): represent acting users and interact with the system through initiating access control requests and performing granted requesting actions. While the $PBAC_B$ model does not differentiate between acting users and subjects as the authors assume that it is trivial to obtain the associating acting user from a subject, such simplification causes the model to be not fully sufficient in addressing DSOD concerns. In a RBAC system, subjects correspond to sessions, where each session is initiated by an acting user activating a subset of his assigned roles. For the model to be role-aware, we make the distinction and utilize the notion of subjects in place of acting users.

Actions (A): represent the set of operations on data objects. These actions are supported by a Provenance-aware System (PAS) in the sense the corresponding provenance information can be captured, represented, stored, and extracted.

Objects (O): represents all data pieces that are stored and utilized by the system. Objects are the main target of protection around which access control mechanisms are built.

Requests (R) and Policies (P): represent the initiation for access to resources and the applicable rules that determine the responses to such requests. Since interactions with the system are no longer directly perpetuated by the acting users but by the subjects instead, requests are changed to the form (Subject, ActionType, Objects). Object Roles (OR) are constructs that enable more precise policy rule specification for requests which include multiple input objects as parameters. In addition, provenance information is utilized in assisting this process.

Dependency List (DL): contains all system-specific predefined pairs of Dependency Name (DNAME) and Dependency Path (DPATH). DNAME are semantical abstractions that are assigned to specific patterns of edges, DPATH(s), which hold significance in the context of the specific application domains. These constructs facilitate policy specifications.

Provenance Data (PD): provenance data captures user transactions in a directed acyclic graph structure which allows edge-traversal enabled queries to be performed on a graph node and obtain resulting sets of vertices that provide lineage, versioning, and other provenance-related information. Provenance Data comprises two subtypes:

- Base Provenance Data: represents the causality dependencies between request components that are captured as results of an access request being granted and executed.
- Attribute Provenance Data: represents the contextual information that is associated with the main components of a request (S, A, or O).

Contextual Information (CI): captures state values of the main components of a request at transaction occurrence time. We identify four different categories of CI:

- Acting Users-related Context: In each application system, an acting user serves as an initiator on access request and associating system events. The context information of acting users can include typical information such as user ID and other forms of attributes.[2]
- Subjects-related Context: While the acting users are responsible for interacting with the system, the subjects are entities that perform actions on behalf of the acting users. In a typical RBAC system, subjects are equivalent to sessions. The contextual information of subjects can include session id, set of activated roles, etc.
- Actions-related Context: The contextual information, which is associated with each action instance that arises from a system event, is unique. The context information that relates actions include temporal aspects, location, weights, etc. Action context information is most suitable for more expressive forms of access control mechanisms.
- Objects-related Context: The object context information can include information such as object size, URI,

[2]We note that while the acting user is not directly captured as a provenance entity, it is trivial to obtain the acting user information from the subjects that the user activates. Therefore, acting user context is effectively still captured in provenance attribute data.

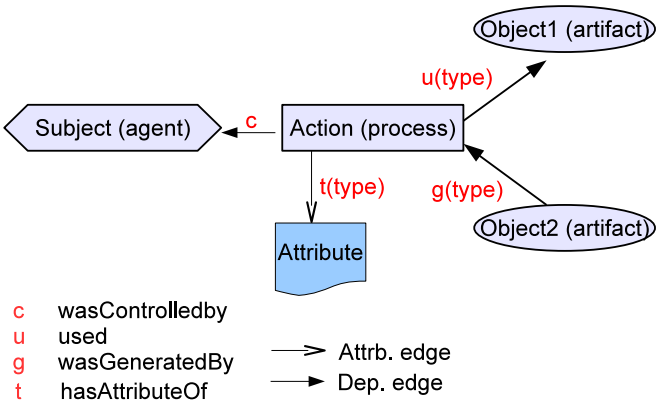

Fig. 2. OPM-style Provenance Graph Notations in $PBAC_C$

etc. Although the object context information can also include information about the origin, list of previous versions, etc., these information does not need to be stored as contextual information as such information is likely to be captured and stored in provenance graph and can be retrieved by accessing the stored provenance data.

2) Interaction: As an access request is generated, the Access Evaluation module extracts the request information to locate the appropriate policies for evaluation. These policies typically make use of the dependency names and associated path expressions, both of which are maintained in the dependency list (DL). The path expressions are constructed using regular expression and carry application-specific semantics. They are used to identify the causality dependencies and contextual information of transaction components (i.e., subject, action, object) that are stored in the provenance storage. Naturally, they can be used to traverse the graph-based provenance data to extract information that is necessary for making access decision. When an access request is granted, the current contextual information is stored as provenance data. This contextual information is uniquely anchored to the action instance of the access transaction in provenance data.

B. Provenance Data in $PBAC_C$ Model

1) Provenance Data Model: Provenance data naturally forms a direct-acyclic graph (DAG). The main entities that are captured in provenance data comprise the graph vertices. The directed edges between adjacent vertices represent the causality dependencies between those vertices. The Open Provenance Model (OPM) [16] is a graph-based model for PD. OPM identifies three main types of vertices as agents, process, and artifacts to refer to subjects, actions, and objects respectively. The vertices are graphically depicted as hexagons, rectangles, and circles, respectively. The dependency edges between these vertices are identified as *wasControlledBy* (or c), *wasGeneratedBy* (or g), *used* (or u), wasDerivedFrom, and wasTriggeredBy. $PBAC_B$ model utilizes mainly u, g, c dependencies for capturing the essential relationships between entities. For more expressive and finer-grained access control, we introduce an additional type of graph edge *hasAttributeOf* (or t) and an additional type of vertices which capture the attribute types and values of the core provenance entities. $PBAC_B$ model allows subtypes of u and g edges. We further allows subtypes of edge t. Variation of edge subtypes is essential for finer-grained and meaningful policy expressions. In figure 2, these subtypes are represented with the parameter, *type*, assigned with specific edge subtypes. The OPM original notations are depicted along with the new notations introduced to capture the concepts presented in the extended model.

2) Capturing User Transactions in Provenance Data: In this section, we describe our perception of how provenance and attribute data of user transactions can be captured in the system. The concept is discussed in the context of the OPM causality dependencies and additional notations that are discussed in the earlier subsection.

In a provenance-aware system, we assume that all user transactions can be captured by the mechanisms available within the capabilities of the underlying system. Without loss of generality, we assume that the capture of a user transaction can be simplified to a construct of the form (*Subject, Action, InputObject, OutputObject, ContextualInfoSet*). Depend on the use cases, either *InputObject* or *OutputObject* can be optional (but not both). *ContextualInfoSet* is a set of all contextual information related to the transaction which is configured and selected as necessary by the system design. Such a set should contain entries of the form (*actionInstance,attributeType,attributeValue*) where an *actionInstance* represents unique ID of the action performed. For example, *(upload1,weight,3)* captures the *weight* attribute of the *upload1* action with the value of 3. The types of attributes and associating values to be captured are anchored to action instances even though attribute types are associated with either subjects/users, objects or action as discussed earlier. As identified in [16], such raw captured information can be represented as a set of triples. Provenance is unique in its feature of forming a direct-acyclic graph (DAG). This characteristic enables powerful capabilities such as ability to traverse the graph and linking of different data objects and object versions. The OPM model exhibits the DAG-like nature of provenance data through the use of basic dependency edges between the identified vertices. These serve as the foundational blocks upon which we build our model.

As discussed earlier, while we capture the context information of different transaction components (i.e., subjects, actions and objects) when the information is captured as attribute provenance data, the attributes are anchored to the action instances. For example, within a homework grading system scenario, a transaction (Subject1, Grade1, HW1, GradedHW1, ContextualInfoSet-Grade1) can be captured in OPM format as (Grade1, u, HW1), (Grade1, c, Subject1), (GradedHW1, g, Grade1). If the essential ContextualInfoSet-Grade1 consists of the following information (ActingUser:Alice, Subject-Role:TA, Grade-Weight:2, HW1-size: 10MB), they can be captured with the following triples: (Grade1, t(actingUser), Alice), (Grade1, t(activeRole), TA), (Grade1, t(weight), 2), and (Grade1, t(object-size), 10MB).

Anchoring transaction attributes to an action instance is necessary and particularly useful for effective management of

1) S, A, AT, O, OR and ATT are subjects, action instances, action types, objects, object roles, and attributes respectively.
2) G, U, G^{-1}, U^{-1}, T and T^{-1} are sets of type variations of 'WasGeneratedBy' and 'Used' dependencies, matching sets of their inverse dependency types, a set of variations of 'hasAttributeOf' attribute types, and a matching set of its inverse attribute types, respectively.
3) $\{`c', `c^{-1}'\}$ is the set of 'WasControlledBy' dependency type, its inverse dependency type.
4) Provenance data PD forms a directed graph and is formally denoted as a triple $< V, E, L >$:
a) $V = S \cup A \cup O \cup ATT$, a finite set of subjects, action instances, objects, and attributes that have been involved in transactions in the system and are represented as vertices;
b) $L = \{`c'\} \cup U \cup G \cup T \cup \{`c^{-1}'\} \cup U^{-1} \cup G^{-1} \cup T^{-1}$, a finite set of dependency type labels and attribute type labels;
c) $E \subseteq \{(A \times S \times `c') \cup (A \times O \times U) \cup (O \times A \times G) \cup (S \times A \times `c^{-1}') \cup (O \times A \times U^{-1}) \cup (A \times O \times G^{-1}) \cup (A \times ATT \times T) \cup (ATT \times A \times T^{-1})\}$, denoting provenance graph edges, is the set of existing dependencies and attributes types in the provenance data. [3]
5) $DNAME$, disjoint from L, is a finite set of abstracted names for a composition of dependency types and attribute types.
6) Let Σ be an alphabet of terms in $L \cup DNAME$. The set $DPATH$ of regular expressions is inductively defined as follows:
a) $\forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH$;
b) $(P_1|P_2), (P_1.P_2), P_1*, P_1+, P_1? \in DPATH$, where $P_1 \in DPATH$ and $P_2 \in DPATH$.
7) $DPATH_L \subseteq DPATH$, is the set of regular expression using only alphabet of terms in L.
8) $DL : DNAME \rightarrow DPATH$, defines each $dname \in DNAME$ as a path expression. DL is also viewed as a list of pairs of dependency names and corresponding dependency paths.
9) $\lambda : DNAME \rightarrow DPATH_L$, maps each $dname \in DNAME$ to a path expression using only dependency type and attribute type labels $l \in L$ by repeatedly expanding the definitions of any $dname_i \in DNAME$ that occurs in $DL(dname)$.
10) P is a finite set of informal XACML-compatible policies.
11) An access evaluation procedure $AccessEvaluation(s, a, O)$ which utilizes the following functions:
a) $\gamma : AT \rightarrow P$, a mapping of an action type to a policy.
b) $\delta : \{S \cup A \cup O\} \times DPATH_L \rightarrow 2^V$, a function mapping a vertex (except attributes in ATT) and a dependency path to vertices in PD such that $v_2 \in \delta(v_1, dpath)$ iff there exists a path in PD from v_1 to v_2 whose edge labels form a string that satisfies the regular expression $dpath$.
c) F is a set of evaluation functions which can be used to evaluate a set of vertices. These functions are application-specific and are designed to suit policy requirements of the underlying target system.
d) RF is a set of rule-evaluation functions which can be used on the result returned by a function $f \in F$ and returns a Boolean.
e) $RuleCombine : 2^{Boolean} \rightarrow Boolean$, is a function which evaluates a set of Boolean values through conjunction and disjunction.

TABLE II. FORMAL SPECIFICATIONS FOR $PBAC_C$

attributes. The further discussion is made in the next section.

C. $PBAC_C$ Model Specifications

The formal specifications for our extended model is built upon the formal definitions provided in [22]. In Table II, we provide the formal specifications for the extended components identified in the above subsection.

We describe how our provenance data model can be utilized in the $PBAC_C$ model. Specifically, the components (S, A, O, and ATT) are captured as vertices on the provenance graph. Causality dependencies types and subtypes are used to label the directed edges between the vertices of types (S, A, and O). Attribute types and subtypes are also used to label the directed edges between A and ATT vertices. Also, we currently only allow incoming edges to attribute nodes and no outgoing edges from them. The u, g, t edge types (but not c are further differentiated by assigning a subtype. These variations of edges are facilitated to capture different semantics or types of the dependency relationships between corresponding edges In addition, each edge type is accompanied with an inverse edge type which allows traversal toward the future transaction instances in provenance data.

There are two ways in which we can capture the attribute provenance data in the graph-based data model.

- The attribute data of a vertex s, a or o is stored as an attribute vertex connected to the corresponding vertex.
- The attribute data of a vertex s, a or o is stored as an attribute vertex connected to only the vertex a. Specifically, attribute of s and o is connected to the vertex a that is associate with s, o in a same transaction.

[3]In the Table II 4c), note that only certain kinds of edges can exist (no O to O edge for example) and only certain labels can be applied to certain kinds of edges (A to S edge must be labelled 'c' for example). By definition each edge is accompanied by its inverse edge to facilitated traversal in forward and backward direction. If the inverse edges are dropped the graph will be acyclic as in the OPM model.

In our model, we choose the second approach. Contextual information of a transaction is stateful and only meaningful in the context of the associated action instance. Anchoring such stateful information to any other vertices (of types S or O) which can be stateless may result in inconsistent and incorrect policy evaluation. For example, a single subject instance can be involved in multiple different action instances while activating different roles for different actions. In such case, if provenance data stores the list of active roles as an anchored attribute to the corresponding subject, multiple attributes with different lists of active roles may exist. This then creates difficulty in identifying which list is for which transaction. On the other hand, as there could be only one action instance for each transaction, anchoring attributes to a corresponding action will eliminate this problem.

We also note the difference between the query-tracing process between our model and the $PBAC_B$ model. Specifically, $PBAC_B$ only allows the starting node, from which a regular path expression query can be traced, to be exclusively O. In our model, we relax this restriction and allow the starting node to be any of the three vertex types (S, A, and O).

Based on the model components, access requests can be evaluated and decided following the access evaluation algorithm provided in Algorithm 1. Policies can be informally specified using a policy grammar that is modified and incrementally extended from the grammar provided in [22]. A policy specification provides a mechanism for matching a request to a corresponding policy ruleset. Each policy rule utilizes pairs of a dependency name construct, which can be applied with a function γ to arrive at base dependency path patterns, and an associated starting graph node to be applied with a function δ which traces through the provenance graph and obtains a set of resulting nodes. Another function, f, can then be applied on the resulting nodes set to obtain values which can be used to assist the evaluation of a rule decision through a function rf. All rules decisions are then combined in accordance to a *RulesCombining* procedure specified in the policy body.

Algorithm 1 $AccessEvaluation(s, a, O)$

(Rule Collecting Phase)
$at \leftarrow a$'s action type
$p \leftarrow \gamma(at)$
$RULE \leftarrow$ access evaluation rules $Rule$ found in p
(Evaluation Phase)
for all $Rule$ in $RULE$ **do**
  Extract the path rule $(StartingNode, DNAME)$ from $Rule$
  Extract $type$ of $StartingNode$ as S, A, or O
  if $type$ is O **then**
    Extract $ObjRole$ of $StartingNode$
    Determine the object $o \in O$, whose role is $ObjRole$
    Set value of $StartingNode$ to o
  end if
  if $type$ is S **then**
    Set value of $StartingNode$ to s
  end if
  if $type$ is A **then**
    Set value of $StartingNode$ to a
  end if
  Extract dependency path expression $dpath \in DPATH_L$ from $DNAME$ using λ function
  Determine vertices $VSet$ by tracing provenance data PD through the paths expressed in $dpath$ that start from the vertex $StartingNode$ using δ function
  Extract $f \in F$ from $Rule$ and execute it on $VSet$
  Extract $rf \in RF$ from $Rule$ and execute it on $f(VSet)$
  Store $rf(f(VSet))$ in $RuleEval$
end for
Execute $RulesCombining(RuleEval)$ and return the Boolean result

All the specific evaluation functions associated with a particular instance of request are obtained from the rule specification in the policy body. For example, consider the sample policy 4 provided in the next section. In this policy, the function f is specified as the sum function and the rf function is specified as "greater-than-or-equal" 3.

IV. DSOD IN $PBAC_C$ MODEL

In this section, we illustrate how the $PBAC_C$ model can be utilized to support traditional DSOD policies. We also identify and discuss several limitations of current approaches as well as additional unique DSOD and some access control features that can be supported with the $PBAC_C$ model.

A. Traditional DSOD in $PBAC_C$ Model

We provide a set of informal policies to express the various DSOD constraints in a provenance-aware system. In the following sample policies, we are using several sample *DNAME*s which we assume are defined accordingly in the dependency list *DL* of the application system. These *DNAME*s can be broken down to a dependency path of basic edge labels (i.e., u, g, c variations), as defined in the model specification, and then utilized in regular path queries. Furthermore, we note that these informal policies can be easily translated to equivalent XACML policies such as the representation of the sample policy 2 as shown in Listing 1.

Sample Policy 1: represents a *simple DSOD* concern in a single session only for simplicity.[4] In particular, a subject should not be active with the role "Reviewer" to activate the role "Student".

[4]Our approach can also address simple DSOD for multi-sessions where active roles of all active sessions of the same user need to be considered by identifying all the active subjects of the same user in provenance data. However, we do not discuss further due to the space limitation.

allow(sub,activate,Student) $\Rightarrow$ *Reviewer* $\notin$ δ *(sub, performedActionsOf:hasAttributeOf(activeRoles)).*

In this example, the DNAME $performedActionsOf$ is used to determine all the actions performed by the subject so the resulting action vertices can be used to identify all the active roles of the subject with $hasAttributeOf(activeRoles)$. Then the policy evaluates whether the "$Reviewer$" role is among the active roles found by the δ function.

Unlike other DSOD variations identified in this paper, simple DSOD does not require any historical information as it only utilize information of currently active roles of the user of a subject. However, it can be still achieved by using provenance data only though it may not be ideal to exercise. To achieve this, as shown above, we first need to find all the actions performed by the subject then further find all the active roles of the identified actions.

Sample Policy 2: represents an *ObjDSOD* concern that requires the requesting subject on replacing a homework object to be activated by the same acting user who activated the subject on uploading it.

allow(sub,replace,o) $\Rightarrow$
δ*(sub,hasPerformedActions:hasAttributeOf(actingUser))*
$\in$ δ*(o,wasUploadedBy)* $\wedge$ *count*(δ*(o,wasSubmittedVof)*) = 0.

In this scenario, to allow a subject *sub* to perform *(action:replace)* on the object *(o)*, the acting user of *sub* should have performed *(action:upload)* on, either through the same or different subjects. The acting user of *sub* can be found by tracing provenance graph staring from o following the path identified as DNAMEs $hasPerformedActions$: $hasAttributeOf(actingUser)$. The user who uploaded o can be obtained by tracing through the provenance graph starting from o and following path as defined with the DNAME *wasUploadedBy*. In addition, the DNAME *wasSubmittedVof* is used in a separate rule to specify an additional rule that the homework object should not have been submitted as well.

Sample Policy 3: represents a *history-based DSOD* ($HDSOD$) concern by requiring that a request to review a homework object can only be allowed after the object is submitted and before it is graded.

allow(s,review,o) $\Rightarrow$ *count*(δ*(o,wasSubmittedVof)*) $\neq$ 0 $\wedge$ *count*(δ*(o,*$wasGradedOof^{-1}$*)*) = 0.

Sample Policy 4: represents a *weighted-action DSOD* concern where the weight of each review process is summed up and the total is used to regulate the access request to grade an object.

allow(sub,grade,o) $\Rightarrow$
sum(δ*(o,previousReviewProcesses:hasAttributeOf(Weight))*)
$>=$ *3*.

DNAME is utilized in all policies for specifying the policy. While careful selection of such named abstraction can convey the semantics of the provenance graph path, it is the DPATH expression that describes a precise path and is used for actual tracing process. We also note that while a DPATH expression can be paired with any types of starting vertex (either $subject$, $object$, or $action$), for efficiency and effectiveness, one type of starting vertex is preferable to another in different cases.

In our HWGS example, we can utilize DPATH(s) with $object$ starting vertices for addressing ObjDSOD and $subject$ starting vertices for simple DSOD, OpsDSOD and HDSOD issues.

B. Unique DSOD Features in $PBAC_C$ Model

The utilization of provenance data in the form of DPATH(s) supports the traditional DSOD variations effectively. Furthermore, it also supports novel features that have not been discussed much, if at all, in the literature. We briefly introduced these features in Section II. In this subsection, we elaborate on these unique features $PBAC_C$ brings about.

Feature 1: Awareness of Past-Action attribute. Context information of the system components provides insightful information on the current state of the system. The information is useful not only for access control mechanisms in the current state, but can also play important part in future access control cases. $PBAC_C$ provides convenient mechanisms for storing and carrying the state attributes of a current action instance into the future, at which point they are past attributes of the current action instance. We demonstrate this novelty through the weight-action example.

When an action transaction is executed in the system, it is assigned a weight by the system policy and setting. That weight attribute is stored in the provenance data in association with that particular action instance. At some later point in the future, the system may assign a different weight value to that action type. An access control policy based on weighted actions can base policy rules on either the current weight attribute state or the ones captured in the past action transitions. $PBAC_C$ facilitates this process and can support both type of mechanisms.

Feature 2: Dependency Path Pattern-based DSOD. Most of traditional DSOD approaches base the control on some fixed and restrictive form of control unit, especially actions. With regular expression-based DPATH(s), we achieve much more expressiveness in the specification of the control unit. In particular, we can express a wide variety of path patterns which includes sequences, repetitions, existences, and any combinations of actions. Additionally, the intrinsic characteristics of provenance data enables versioning of a data object and linkages of distinct but related data objects. This enables the grouping of objects or object versions with meaningful semantics, on which DSOD constraints (e.x., conflicting actions on current and past object versions instead of a particular object), can be specified.

The identified features above allows a broader family of DSOD policies and constraints to be specified. This goes hand in hand with the complexity a provenance-aware environment and all its huge amount of data and the associating complexity can produce. It seems natural that as the amount of data and complexity rises, so do concerns of DSOD involved in such environment. Utilizing provenance constructs such as dependency path patterns and associating dependency names can lay a strong foundation for addressing this phenomenon.

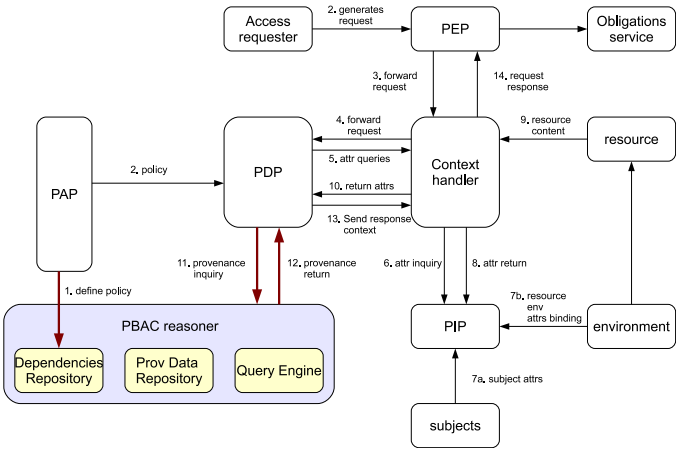

Fig. 3. An extended XACML architecture

V. Prototype: Architecture, Implementation, and Evaluation

A. Extended XACML Architecture

1) Existing Components: The main components of the XACML architecture include the Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Administration Point (PAP), and Policy Information Point (PIP). The PEP is responsible for receiving and enforcing access requests from a front end interface. The PDP receives the transferred request from the PEP and is responsible for evaluating the request. It does so by obtaining the appropriate policy sets and rules from the PAP. It also obtains relevant additional information from the PIP, which is responsible for looking up information involving the subjects, objects and the system environment. Obligations also play important roles but the concept is outside the scope of our approach and henceforth not considered in details.

2) Policy Language: The XACML policy language essentially comprises policy targets and policy rules. The policy targets include subjects, actions, and resources. The policy rules, which are used for access decisions, consist of conditions and potentially obligations. We can express the informal policies for PBAC mechanism with the XACML policy language. For example, policy 2 can be expressed in an equivalent XACML policy as shown in Listing 1. Here, the function "provenance-query-SPARQL" corresponds to δ while the other functions capture other rules evaluation functions correspondingly.

3) PBAC Reasoner Component : In order to enable PBAC mechanisms in the XACML framework, we propose an extension to the existing XACML architecture as shown in Fig 3. Specifically, we introduce a new component, PBAC Reasoner. The PBAC Reasoner communicates directly with the PDP and PAP components and is responsible for the specification of provenance-based access control policies. It also provides specific mechanisms for storing and extracting provenance data for access control request evaluation. The PBAC Reasoner component is further composed of three additional subcomponents:

- Dependencies Repository (DR): The dependencies repository component is responsible for storing application-specific dependency lists. In this work, we

assume only one set of dependency path patterns and associating dependency name constructs is at use, as only one system is in concern. However, it is possible to have multiple dependency lists when multiple systems are considered, as in a cloud or distributed environment. The dependency list is utilized by the PAP for policy specification purposes.

- Provenance Data Repository (PDR): The PDR component is responsible for storing captured provenance information. This can include both the base provenance as well as attributes data.
- Query Engine (QE): The query engine is mostly associated with the provenance database employed by the PDR. It is important that the QE is capable of performing regular path queries as provenance graph tracing is essential in the PBAC approach.

B. Implementation

Our prototype employs various state-of-the-art tools available in the community. We are using the Oasis XACML[5] implementation for policy specifications and the existing implementations of the XACML components. For the PDR, we are utilizing the Jena framework [6], where provenance graph is stored in RDF-triples [15] format. In this work, we are using Jena-2.7.4 and the corresponding ARQ package [6]. The query engine associating with Jena is ARQ, which we utilized to execute SPARQL [24] queries on the RDF-format provenance graph database. Dependency lists are trivially implemented as arraylists of String values-pairs. To enable the communication between the existing components of the XACML framework, specifically the PDP and the PBAC Reasoner components, we implemented a specialized function that can be incorporated to the XACML framework at runtime. The function is essentially an extended class to the *FunctionBase* class available in the Sun's framework. The function essentially performs the tasks equivalent to the functions δ elaborated in the model specification. It it assigned a NAME-ID, in our case "provenance-query-SPARQL", and then incorporated, after which it can be used in policy rulesets, as shown in Listing 1.

Listing 1. Sample Policy 2 in XACML

```
<Policy PolicyId="replacePolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
     combining-algorithm:ordered-permit-overrides">
<Target>
...
<Actions>
<Action>
<ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0
     :function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
     string">replace</AttributeValue>
<ActionAttributeDesignator AttributeId="
     urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</ActionMatch>
</Action>
</Actions>
</Target>

<Rule RuleId="ReplaceRule" Effect="Permit">
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:and">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:string-is-in">
<Apply FunctionId="provenance-query-SPARQL">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:string-one-and-only">
<SubjectAttributeDesignator AttributeId="
     urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
     string">hasPerformedActions:hasAttributeOf(actingUser)
</AttributeValue>
</Apply>
<Apply FunctionId="provenance-query-SPARQL">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:string-one-and-only">
<ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
     id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
     string">wasUploadedBy
</AttributeValue>
</Apply>
</Apply>

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:integer-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
     integer">0</AttributeValue>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:string-bag-size">
<Apply FunctionId="provenance-query-SPARQL">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0
     :function:string-one-and-only">
<ResourceAttributeDesignator AttributeId="
     urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
     string">wasSubmittedVof
</AttributeValue>
</Apply>
</Apply>
</Apply>
</Condition>

</Rule>

<Rule RuleId="FinalRule" Effect="Deny" />
</Policy>
```

C. Evaluation

In order to evaluate the performance of our prototype, we design an experiment to test and capture the runtime execution of a request instance. We deploy the implemented prototype onto a virtual machine instance of an Ubuntu 12.10 image with 4GB Memory and a 2.5 GHz quad-core CPU. We design sample XACML policies and sample XACML requests that would require the prototype to invoke the PBAC Reasoner components to gather necessary data for the access decisions. More specifically, we measure the time it takes to complete the following operations flow[7]:

- The access request is received by the PDP.
- An appropriate access policy is matched.
- Relevant parameters are extracted from the policy and passed to the PBAC Reasoner.
- SPARQL regular path queries are formed by utilizing the DR and QE configuration.

[5]https://www.oasis-open.org/

[6]http://jena.apache.org/

[7]In this work, we disregard the use of PEP for simplification purpose as it does not make a significant impact on the scope of our topic

- Queries are executed against PDR and results are returned to the PDP.
- PDP performs evaluation process on the returned results and return the final decision.

For the application domain, we simulate the HWGS scenario as described earlier. We generate mock data of possible action transactions that occurred within the system and capture them using the proposed provenance data model and store them as RDF-format triples within an in-memory Jena model. Regular-path SPARQL queries are generated and executed against the Jena in-memory model using the ARQ engine.

We recognize the fact that the bottleneck of our prototype lies at the execution of regular path queries. The performance of such query execution depends heavily on the shape of the provenance graph and the pattern after which the tracing needs to be done. Therefore, in generating the mock data, we take into consideration two types of shape a provenance graph takes. One requires wide and the other requires deep tracing. A wide provenance graph tracing is necessary to query a large amount of actions that all take a single object as input. For example, in the HWGS scenario, one submitted homework object can be reviewed by a multitude of reviewers. A query which searches for all reviewers of an object would have to traverse through all edges branching out from the object itself. A deep provenance graph tracing is necessary to query a path of cause and effect relations between different versions of an object. For example, an uploaded homework object can be replaced multiple times by its owner. To obtain the original version of a homework, a query needs to traverse back through a large number of edges.

In order to test the flexibility of the framework, it is interesting to evaluate the performance of our framework against various quantity of edges that need to be traversed (the width and depth of a provenance graph). We design our experiment so that our prototype is validated against "extreme", yet reasonable, thresholds. In particular, we evaluated requests that would require incremental number of edges (to be traversed) in the quantities of 2000, 4000, 6000, 8000, 10000 and 12000. To produce precise results, requests of wide type would only require wide edges traversing. The same configuration applies to requests of deep type.

The experimental results are shown in Figure 4. For the most heavy tracing query, we obtain the result of 0.718 second per deep request and 0.069 second per wide request. At the same time, for the lightest tracing query, the result is 0.017 second per deep request and 0.014 second per wide request. We note that for both types of queries, the resulting runtime demonstrate the feasibility of the prototype. While a query that requires a deep tracing shows increased runtime, such phenomenon is most likely because of the SPARQL implementation of query execution that utilizes recursive calls for each successive process step. Regardless, we made an observation that in a practical system deployment, the depth and width traversals of the associated provenance graph do not typically exceed such quantities. Furthermore, in certain special cases, while a provenance graph can grow extremely large in both depth and width, a practical application system and its associated provenance graphs do more often resemble a large set of disconnected graph components with small depth

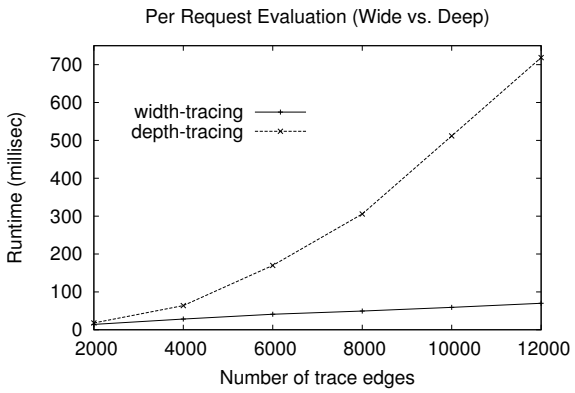

Fig. 4. Evaluation Results

and width values. Furthermore, we perceive that in the case of a large provenance graph, various optimization approaches, such as requests grouping and results caching, can be performed to improve the runtime results. The discussion of these concepts lays beyond the scope of this paper and belongs in our future works. Nonetheless, although the obtained results are not optimal, we strongly believe they clearly demonstrate the feasibility and potential enhancement of the prototype together with our $PBAC_C$ approach.

VI. Related Works

Provenance security has increasingly gained attention from the community. Researches generally put more emphasis on securing provenance data, such as works by [3], [11], [12], [29]. In contrast, we advocate a less popular approach , which utilizes provenance data for securing normal data, that has been garnering attention recently [1], [20], [28].

The XACML architecture [17] has been widely considered suitable for enabling Attribute-based Access Control [14], [23]. Its inherent characteristics, however, allow the enablement of various extensions which can address different access control issues. In this work, we extend the XACML architecture with the PBAC Reasoner component that activates provenance-awareness for enhancing DSOD. Ferrini et al [9] propose extending the XACML architecture to enable OWL ontology for RBAC purposes. They approach DSOD constraints through the obligations component. The XACML policy language is utilized and extended to support provenance access control use cases [5], [21].

In this work, we utilize a provenance data model that is inspired from the OPM model [16]. The OPM model captures causality dependencies between provenance entities. Other provenance data models [4], [8], [13] are designed to capture different aspects of provenance data that are most suitable for their application domain.

Our prototype is designed with the assumption that active provenance capture mechanisms already exist in the underlying system. Providing provenance capture mechanisms is outside the scope of our work and addressed elsewhere [7], [18].

VII. Conclusion

Traditional DSOD literature assumes necessary history information of user activities is readily available to the system and does not articulate related issues in utilizing this

history information. In this paper, we propose an approach toward DSOD issues that utilizes provenance data for enabling the mechanisms for capturing, storing, and extracting system history information. Specifically, we propose an extension to the base $PBAC$ model which utilizes context information of transaction events for enhanced DSOD capabilities, both in addressing the traditional DSOD features as well as newly identified features such as past attribute-based DSOD and dependency path pattern-based DSOD that have not been discussed in literature. Based on the popular XACML architecture, we built a proof-of-concept prototype.The evaluation results of our prototypes demonstrated the feasibility of the prototype and the potential for additional enhancements. In summary, we strongly believe our approach will lay a foundation for not only a finer-grained DSOD enforcement that are not explored elsewhere in the existing literature, but also more expressive and versatile access control mechanisms in general.

Acknowledgement

This work is partially supported by NSF CNS-1111925.

REFERENCES

[1] A. Bates, B. Mood, M. Valafar, and K. Butler. Towards secure provenance-based access control in cloud environments. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY '13, pages 277–284, New York, NY, USA, 2013. ACM.

[2] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes. *PROV-DM: The PROV Data Model*. 2012.

[3] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, July 2008. USENIX Association.

[4] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *Proceedings of the international conference on Management of data*, SIGMOD, pages 539–550. ACM, 2006.

[5] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. A language for provenance access control. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 133–144. ACM, 2011.

[6] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 74–83, New York, NY, USA, 2004. ACM.

[7] A. Chapman, B. Blaustein, L. Seligman, and M. Allen. Plus: A provenance manager for integrated information. In *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*, pages 269 –275, aug. 2011.

[8] A. P. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 993–1006, New York, NY, USA, 2008. ACM.

[9] R. Ferrini and E. Bertino. Supporting rbac with xacml+owl. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 145–154, New York, NY, USA, 2009. ACM.

[10] V. Gligor, S. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 172 – 183, may 1998.

[11] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 13–18, New York, NY, USA, 2007. ACM.

[12] R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *Trans. Storage*, 5(4):12:1–12:43, Dec. 2009.

[13] T. Heinis and G. Alonso. Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1007–1018, New York, NY, USA, 2008. ACM.

[14] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy*, DBSec'12, pages 41–55, Berlin, Heidelberg, 2012. Springer-Verlag.

[15] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.

[16] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743 – 756, 2011.

[17] T. Moses. eXtensible Access Control Markup Language TC v2.0 (XACML), Feb. 2005.

[18] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[19] D. Nguyen, J. Park, and R. Sandhu. Dependency path patterns as the foundation of access control in provenance-aware systems. In *4th USENIX Workshop on the Theory and Practice of Provenance*, TaPP'12. USENIX Association, Jun. 2012.

[20] D. Nguyen, J. Park, and R. Sandhu. Integrated provenance data for access control in group-centric collaboration. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 255–262, 2012.

[21] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han. An access control language for a general provenance model. In *Proceedings of the 6th VLDB Workshop on Secure Data Management*, SDM '09, pages 68–88, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *10th Annual Conference on Privacy, Security and Trust*, PST 2012. IEEE, Jul. 2012.

[23] J. Park and R. Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, Feb. 2004.

[24] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.

[25] R. Sandhu. Transaction control expressions for separation of duties. In *Proc. of the Fourth Computer Security Applications Conference*, pages 282–286, 1988.

[26] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38 –47, feb 1996.

[27] R. Simon and M. Zurko. Separation of duty in role-based environments. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 183 –194, jun 1997.

[28] L. Sun, J. Park, and R. Sandhu. Engineering access control policies for provenance-aware systems. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY '13, pages 285–292, New York, NY, USA, 2013. ACM.

[29] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, and L. Moreau. Security issues in a soa-based provenance system. In *In Proceedings of the International Provenance and Annotation Workshop*. Springer, 2006.