A Provenance-based Access Control Model

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

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

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

***Abstract*—Existence of data provenance information in a system raises at least two security-related issues. One is how provenance data can be used to enhance security in the system and the other is how to protect provenance data which might be more sensitive than the data itself. Recent data provenance-related access control literature mainly focuses on the latter issue of protecting provenance data. In this paper, we propose a novel provenance-based access control model that addresses the former objective. Using provenance data for access control to the underlying data facilitates additional capabilities beyond those available in traditional access control models. We utilize a notion of *dependency* as the key foundation for access control policy specification. Dependency-based policy provides simplicity and effectiveness in policy specification and access control administration. We show our model can support dynamic separation of duty, workflow control, origin-based control, and object versioning. The proposed model identifies essential components and concepts and provides a foundational base model for provenance-based access control. We further discuss possible extensions of the proposed base model for enhanced access controls.**

I. INTRODUCTION

Provenance is the documentation of the origin of a data object and the processes that influence and lead to any particular state of that object. Capturing and storing provenance data enables higher trustworthiness and utility of the underlying data. As the importance of its role in application systems increases, there also arises provenance related security and trustworthiness issues in the system. While securing provenance data has been studied in recent years, how to utilize provenance data to control access to the underlying data is rarely discussed in literature.

The myriad of software applications available in both private and public sectors demands more complex protection mechanisms for the resulting large networks of information flow. Traditional access control mechanisms are built for specific purposes and are not easily configured to address the complex demands associated with these new technologies. We strongly feel that access control systems built upon provenance data by fully utilizing its unique characteristics will provide a foundation for new access control mechanisms that are highly capable of supporting features that were not easily achievable with traditional access control solutions. Specifically, as data provenance provides utilities such as pedigree information search, usage tracking, and versioning, using provenance data for access control allows more versatile control capability such as controls based on pedigree information, past usage of data, past activity of users and versioning information. This further supports dynamic separation of duties and workflow controls.

In this paper, we introduce the framework for a family of Provenance-based Access Control (PBAC) models with extensions that can handle traditional access control issues in a simple and effective manner. We identify the essential foundations and discuss in depth the base model upon which additional functionalities can be built. We apply our model to a case study on a course grading system. We demonstrate that our model is capable of handling various access control principles and features such as dynamic separation of duties, workflow control, origin-based control, and object versioning.

II. PROVENANCE DATA DEPENDENCY AND ACCESS CONTROL

In this section, we discuss two preliminary topics necessary for understanding our proposed model. We first discuss our view on causality dependencies of provenance data identified in the Open Provenance Model (OPM) [20] and then identify two approaches related to data provenance in access control.

A. Provenance Data Dependency

In any active system, transactions occur and involve subjects, objects, and the corresponding actions describing the interaction between these. The log of all such transactions can be seen as the basis of provenance information. However, without relevant semantics to be assigned to the transactions log, not much benefits can be gained. For transactions information to be considered useful provenance information, causality dependencies of transaction data should be utilized. Without causality dependency as semantics foundation, it is hard to utilize transaction flows and associated information. We acknowledge this essential provenance property, for which purpose we need a suitable provenance model. Our work builds on OPM since it provides a foundation for the causality dependencies of provenance data.

In essence, the model's main components consist of artifacts, processes, and agents. Five main dependencies between two components are defined as 'used' (process on artifact), 'wasGeneratedBy' (artifact on process), 'wasControlledBy' (process on agent), 'wasDerivedFrom' (artifact on artifact), and 'wasTriggeredBy' (process on process). Altogether the components and dependencies form a directed acyclic graph where the main components correspond to nodes and the dependencies correspond to edges.

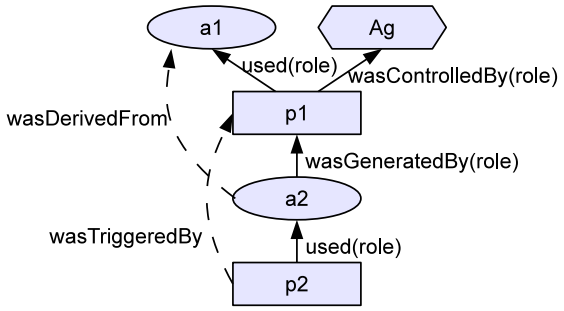

Fig. 1. OPM Causality Dependencies

The most basic graph exhibiting all these graph entities is depicted in Figure 1. Here the agent Ag controlled the process p1 which used the artifact a1 to generate the new artifact a2 which was then used by the process p2. Notice the direction of the arrows specifies a causality relationship. The source of the arc represents the effect while the destination represents the cause. Also, the fact that p1 used a1 and generated a2 does not guarantee that a2 was derived from a1, hence that needs to be asserted with the 'wasDerivedFrom' edge from a2 to a1. Similarly, the 'wasTriggeredBy' edge is used to assert the causality dependency between p1 and p2.

We utilize the OPM model to capture base provenance data in this paper. Artifacts are used to capture data objects (e.g., submitted paper, review, etc.). Processes are used to capture the functional actions such as upload, submit, grade, etc. Agents correspond to users. We only utilize the direct dependencies of OPM, and omit the indirect dependencies of 'wasDerivedFrom' and 'wasTriggeredBy' in our base provenance data (variations of these could be utilized in the extended models). To assign additional semantics to the OPM graph and facilitate convenient adaptation of OPM to applications of various domains, the OPM model provides a notion of role that further specifies dependencies. For example, an object may have a dependency 'wasGeneratedBy(submit)' with a submit process, meaning that the object was generated by submit action (as opposed to say the grade action). In OPM the dependency role is applicable to 'used','wasGeneratedBy' and 'wasControlledBy' dependencies. For our purpose, we only adopt roles for 'used' and 'wasGeneratedBy' dependencies, as we allow only one agent for each process.

B. Data Provenance in Access Control

Using provenance data in a system raises at least two security related issues. We identify them as *provenance-based access control (PBAC)* and *provenance access control (PAC)*. PBAC focuses on how provenance data can be used to control access to data, while PAC concerns how access to provenance data should be controlled. There has been considerable attention recently on securing provenance data, i.e., PAC [5], [6], [8], [9], [16], [17], [22], [27]. In this paper we focus on PBAC. PBAC and PAC are complementary to each other in that PBAC can be used to control access to provenance data and PAC can be used to elevate trustworthiness of provenance data. Furthermore, they both require mechanisms to capture, store and retrieve provenance data. Therefore, though this paper focuses on a foundational model for PBAC, we believe the proposed model also provides a foundation for proper understanding of PAC since it identifies how provenance data should be structured and retrieved.

Although provenance-based access control utilizes provenance data to make access decision, it is likely the case that a real world system will also require other forms of access control systems together with PBAC. For example, consider a homework review and grading example in an online course management system presented in our sample case study. PBAC can support policies such as only the user who uploaded a homework can replace it with a newer version or can submit it, the user who submitted a homework cannot review the homework, or a user can append reviews to a grade report only if the review was completed for the homework. This application system is likely to additionally utilize role-based access control to enforce policies such as only students can submit homework or review other student's homework and only instructors can grade a homework or append reviews to a grade report. For this reason, the policies in the proposed model are defined as necessary rather than sufficient for access, since additional non-provenance based policies may also come into play.

III. A Family of PBAC Models

In this section, we identify a family of models for provenance-based access control based on three criteria.

The first criteria is the kind of provenance data is used in the system. Provenance-based access control utilizes provenance data to make decisions on users' access to data objects. The provenance data stores transaction records that are captured in a system from which causality dependencies can be computed. It may also store dependency information that are identified by users. For example, a PC member of a conference may delegate an actual reviewer of the paper that is assigned to her or an author may identify additional authors of a paper that he submitted.

The second criteria is whether policies are based on object dependencies or acting user dependencies. Policies using object dependencies verify transaction history of objects while policies using acting user dependencies utilize history of users.

The third criteria is whether the policies are given and readily available to the system or they are retrieved from other users or objects that are discovered by tracing provenance data based on additional policy retrieval policies.

Based on these criteria, we have identified one base model and three extended models for provenance-based access control as shown in Figure 2. $PBAC_B$ is our base model that focuses on system captured and system computable provenance data and object dependencies, and assumes policy is given. In the three extended models, $PBAC_U$ extends the base model to allow user-declared provenance data, $PBAC_A$ extends the base model to include acting user dependencies, and

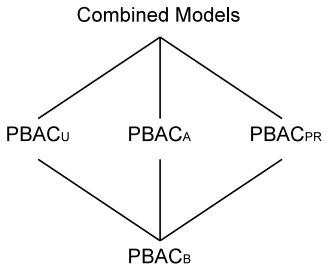

Fig. 2. A Family of PBAC Models

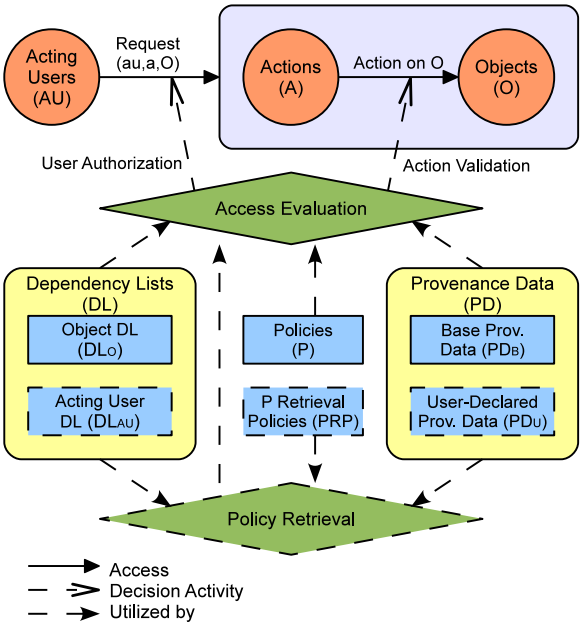

Fig. 3. PBAC Model Components

$PBAC_{PR}$ extends the base model by including provenance-based policy retrieval. Combinations of these extended models are also possible. In this paper we mainly focus on the base model which presents essential aspects of PBAC.

IV. PBAC Model Components

The proposed provenance-based access control model consists of several core components. Figure 3 shows these components. They are acting users, action instances, action types, objects, object roles, provenance data, dependency lists, policies, access evaluation function for user authorization and action validation, policy retrieval policies, and policy retrieval function. Those components with solid line boundaries are necessary for the proposed base model. Others with dashed line boundaries are part of the extended models. While we also identify and briefly discuss the additional components for the extended models in this section, our overall focus is on the base model.

Acting Users (AU) represent human beings who initiate requests for actions against objects. To be precise, provenance data captures acting subjects, not acting users. While this is true, since one user can have many acting subjects that are stored in provenance data and provenance-based access control policies are defined based on acting users, we assume a corresponding user of an acting subject can be identified. For the sake of simplicity, we use acting users instead of subjects in our model.

Action instances (A) are initiated by users for an access to objects. Provenance-based policies are defined by using **action types** (AT) rather than action instances. AT is a fixed finite set of action types predefined by system architects. We assume a system can derive action types from given action instances.

Objects (O) are resource data that are accessed by users. Our provenance-based access control model supports object versioning and allows multiple versions of an object. Provenance data captures object versions as vertices. When a transaction modifies an object version, the new version is represented as a new vertex in provenance data. If an object version is copied or modified into a new object, the output is represented as a new object.

A **Request** consists of an acting user, an action instance and a set of objects with object roles that are to be accessed. Specific object roles for each action type are pre-defined in the system. In a request, the action type of the action instance may require multiple objects with different **object roles** (OR) assigned to them. For example, an $append$ action type requires one reference object and one source object to which the contents of the reference are appended. Object roles are necessary since objects with different roles will, in general, need different rules for granting an action request. Once a request is allowed and performed, the corresponding transaction data is captured and stored as part of the base provenance data. Transaction data includes acting user, action instance, input object(s) and output object(s).[1]

Provenance data (PD) consist of base provenance data and user-declared provenance data. The **base provenance data** (PD_B) store transactions data that are captured as a result of performed actions and form a directed graph that is similar to the one identified in OPM [20]. Each transaction is stored as a set of triples that consists of two entities and one causality dependency. The causality dependency utilizes three basic dependency types of 'wasControlledBy', 'wasGeneratedBy' and 'used' out of five causality dependencies identified in OPM [20].[2] For example, suppose a user $u1$ appended object $o1$ to object $o2$. The transaction data will be $< u1, append1, (o1, o2), o1v2 >$ and corresponding provenance data will store $< append1, u1, wasControlledBy >$, $< append1, o1, used(source) >$, $< append1, o2, used(ref) >$ and $< o1v2, append1, wasGeneratedBy(append) >$. Here, $o1$ and $o2$ are input objects, $o1v2$ is the output object and $source, ref, append$ are roles. OPM allows a notion of role

[1]While other contextual information such as timestamp, location, platform and purpose can be captured and utilized for provenance-based access control, we do not consider such auxiliary information in this paper since they are not the basic ones for provenance-based access control.

[2]The remaining two dependencies are not used in our base provenance data since they are indirect dependencies that either can be system-computed from the direct dependencies or are user-declared.

for these three base dependencies. However we utilize roles for only 'wasGeneratedBy' and 'used' dependencies to specify how objects are generated or used. As mentioned earlier, we do not allow roles for 'wasControlledBy' since we assume there is only one acting user per action instance.

Unlike OPM, in the provenance data of our model, we make use of all the matching inverse dependencies of the dependencies that are captured as a result of transactions. Using normal dependencies, provenance data can be traced only backward in time. The inverse dependencies are necessary for traversing some dependency data since, for example, a request to modify an object may need some verification whether its newer version had been viewed or not. This rule can be verified by traversing the provenance data forward in time. It may also require changes in direction multiple times in case, for example, one may want to check whether any of the related objects and object versions was ever viewed or accessed by someone.

If allowed, users can declare new dependencies or deny existing dependencies. While these **user-declared dependency data** (PD_U) can be utilized in provenance-based access control together with base provenance data, the proposed base model only focuses on base provenance data since user-declared provenance data are not as critical as base provenance data and can be better discussed only with a precise understanding of a model that utilize base provenance data.

Dependency lists (DL) are constructed as pairs of abstracted dependency names (DN) and corresponding dependency path expressions ($DPATH$). Each $dn \in DN$ is paired with, and defined by, exactly one $dpath \in DPATH$, where $dpath$ may use other previously defined dn's. Recursive or cyclic definitions are not permitted so each $dpath$ can be reduced to a regular expression using only base dependency types by expanding the dn definitions inline. There can be object dependency lists and acting user dependency lists. Object dependency lists include dependencies between objects and other entities in provenance data such as other objects, acting users (agents in OPM), or action instances. Likewise acting user dependency lists include dependencies of acting users. The proposed model consider only object dependency lists as object dependency is an essential notion of provenance based access control.[3]

Policies include a set of rules that need to be evaluated for granting access. These rules are either for user authorization or action validation. **User authorization rules** specify whether the requester is qualified for the request or not while **action validation rules** specify whether the requested action can be performed against the requested objects. Both types of rules are specified using dependency names. There is only one policy per action type.

Access Evaluation function evaluates a request by utilizing user authorization rules and action validation rules found in the policy for the type of the requested action and returns a boolean value. The algorithm for access evaluation is provided in Algorithm 1.

Policy Retrieval Policies and **Policy Retrieval Function** shown in the Figure 3 are used to discover policies that need to be used for access evaluation. The proposed base model assumes policies are given and readily available to the system hence does not consider policy retrieval function and policies. They are identified as components for one of the extended models, and are not further discussed in this paper.

[3]Note that while acting user dependency information is available in provenance data, it is not likely to be the main information that provenance data captures. In fact, by definition, provenance is history of objects not acting users though it also includes user's activity histories.

V. The $PBAC_B$ Model

In this section, we first introduce a policy specification grammar that can support our base provenance-based access control ($PBAC_B$) model. Then we define the base model and provide an access evaluation algorithm to show how access control decision in the proposed model is made.

A. Policy Specifications

In this section we define a policy specification grammar PG as shown in Table I. Policies for the proposed model consist of a set of user authorization rules ($UARules$) and action validation rules ($AVRules$). The overall result of both of these is combined by conjunction. Each rule is defined using path rules that consist of a starting node and a dependency name to which a regular expression-based dependency path pattern is mapped in a dependency list. (See the model definition section below.) A user authorization rule is defined using an acting user, a path rule and an operator and checks (non-)existence of acting user in the vertices found using the path rule, while an action validation rule is defined using one or two of the path rules and an operator and either checks (non-)existence or frequency of vertices in the path or compares two sets of vertices found in the two paths. These three types of rules (one user authorization rule and two action validation rules) are by no means exhaustive but are sufficient to capture the sample use case scenario presented in this paper. Each user authorization rule is individually evaluated to a boolean result. The individual results are then combined using disjunction and conjunction as specified. Action validation rules are similarly individually evaluated and then the results are combined using disjunction and conjunction as specified.

B. Model Definitions

Based on the core components identified in the previous section, we define a base model for PBAC as follows.

1) AU, A, AT, O and OR are acting users, action instances, action types, objects, and object roles respectively.
2) G, U, G^{-1} and U^{-1} are sets of role-specific variations of 'wasGeneratedBy' and 'used' dependencies and matching sets of inverse dependencies, respectively.
3) $\{$'c', 'c^{-1}'$\}$ is the set of 'wasControlledBy' dependency and its inverse dependency.
4) Base provenance data PD_B forms a directed graph and is formally denoted as a triple $< V_B, E_B, D_B >$:

$Policy ::=$ "$allow$" $< Req >$ " $\Rightarrow$ " $< UARules >$ " $\wedge$ " $< AVRules >$ |
"$true$"
$Req ::=$ "(" $< ActUser >$ "," $< ActType >$ "," $< ObjRoles >$ ")"
$ObjRoles ::= < ObjRole >$ | $< ObjRole >$ "," $< ObjRoles >$
$UARules ::= < UARule >$ |"(" $< UARules >$ ")"|
$< UARules >< Connect >< UARules >$
$AVRules ::= < AVRule >$ |"(" $< AVRules >$ ")"|
$< AVRules >< Connect >< AVRules >$
$Connect ::= \vee | \wedge$
$UARule ::= < ActUser >< oper1 >< PathRule >$
$AVRule ::=$ "|" $< PathRule >$ "|" $< oper2 >< Number >$ |
$< PathRule >< oper3 >< PathRule >$
$PathRule ::=$ "(" $< ObjRole >$ "," $< DName >$ ")"
$oper1 ::=$ " $\in$ "|" $\notin$ "
$oper2 ::=$ " $=$ "|" $\neq$ "|" $\geq$ "|" $\leq$ "|" $<$ "|" $>$ "
$oper3 ::=$ " $=$ "|" $\neq$ "|" $\subseteq$ "
$DName ::= dn_1|dn_2|\ldots|dn_n$
$Number ::= [0-9]+$
$ActUser ::= au$
$ActType ::= at_1|at_2|\ldots|at_m$
$ObjRole ::= o_{role_1}|o_{role_2}|\ldots|o_{role_k}$

TABLE I
A POLICY SPECIFICATION GRAMMAR PG

- $V_B = AU \cup A \cup O$, a finite set of acting users, action instances, and objects that have been involved in transactions in the system and are represented as vertices;
- $D_B = \{\text{`}c\text{'}\} \cup U \cup G \cup \{\text{`}c^{-1}\text{'}\} \cup U^{-1} \cup G^{-1}$, a finite set of base dependency types;
- $E_B \subseteq \{(A \times AU \times \text{`}c\text{'}) \cup (A \times O \times U) \cup (O \times A \times G) \cup (AU \times A \times \text{`}c^{-1}\text{'}) \cup (O \times A \times U^{-1}) \cup (A \times O \times G^{-1})\}$, denoting dependency edges, is the set of existing base dependencies in the provenance data.[4]

5) DN_O, disjoint from D_B, is a finite set of abstracted names for dependencies of objects.
6) Let Σ be an alphabet of terms in $D_B \cup DN_O$. The set $DPATH$ of regular expressions is inductively defined as follows:
   - $\forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH$;
   - $(P_1|P_2), (P_1.P_2), P_1*, P_1+, P_1? \quad \in \quad DPATH$, where $P_1 \in DPATH$ and $P_2 \in DPATH$.
7) $DPATH_B \subseteq DPATH$, is the set of regular expression using only alphabet of terms in D_B.
8) $DL_O : DN_O \rightarrow DPATH$, defines each $dn \in DN_O$ as a path expression. DL_O is also viewed as a list of pairs of object dependency names and corresponding dependency paths.
9) $\lambda_O : DN_O \rightarrow DPATH_B$, maps each $dn \in DN_O$ to a path expression using only base dependency types $d_b \in D_B$ by repeatedly expanding the definitions of any $dn_i \in DN_O$ that occurs in $DL_O(dn)$.
10) PE is a language specified in the policy expression grammar PG.
11) $P \subseteq PE$, is a finite set of policies.
12) $\gamma : AT \rightarrow P$, a mapping of an action type to a policy.
13) $\delta_O : O \times DPATH_B \rightarrow 2^{V_B}$, a function mapping an object and a base dependency path to vertices in PD_B such that $o_2 \in \delta(o_1, dpath)$ iff there exists a path in PD_B from o_1 to o_2 whose edge labels form a string that satisfies the regular expression $dpath$.

[4]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 AU edge must be labelled 'c' for example). By definition each edge is accompanied by its inverse edge to facilitate traversal in forward and backward direction. If the inverse edges are dropped the graph will be acyclic as in the OPM model.

Definition 2-3 define base dependencies which are the building blocks of base provenance data as defined in definition 4. In definition 4, these dependencies can be used only between certain kinds of vertices. For example, 'c' and 'c^{-1}' can be used to connect an action instance to an acting user and an acting user to an action instance, respectively. The simplicity and effectiveness in policy specification and access control management are achieved with the utilization of dependency names and matching dependency paths in dependency list (DL_O) as shown in Definition 5-9. Definition 10-12 provide the means for defining policies and attaching them to action types. Definition 13 defines the δ_O function necessary for access request evaluation with respect to the given PD_B.

C. Access Evaluation Procedure

The Access Evaluation Procedure is specified in Algorithm 1. In the algorithm, line 2 - 6 shows the rule collecting phase that identifies all the user authorization rules and action validation rules from the policy applied to the action type of the request. The user authorization phase (line 7 - 14) and action validation phase (line 16 - 25) differ in that action validation may need to compute multiple sets of vertices from multiple path rules while user authorization only evaluates one path rule. The user authorization phase compares the acting user of the request to the vertices found as a result of checking the path rules against the base provenance data (PD_B) and returns a boolean value. An action validation rule evaluates the existence or number of vertices found by a path rule or compares multiple sets of vertices found by multiple path rules, and then returns a boolean value. User authorization rules (as well as action validation rules) are connected using conjunctive and disjunctive connectives. Once the truth values of these two phases is computed, the algorithm evaluates the final truth value using conjunctive connective.

We provide a partial analysis for the complexity of this algorithm. It is trivial that all steps outside the FOR loop from line 7 - 13 and the nested FOR loops from line 16 - 24 are in PTIME. Since the number of rules and path rules are finite, all the extraction steps in the two FOR loops except lines 11 and 21 are also in PTIME. Hence the complexity is dominated by lines 11 and 21. Lines 11 and 21 require a tracing algorithm on the provenance graph. In practice this tracing algorithm would be embedded in queries that support regular expression-based path patterns. We conjecture that the complexity in some subsets of practical problem space is achievable in PTIME while in other cases may be NP-complete or worse. While further investigation is necessary, a detailed complexity analysis is beyond the scope of this paper.

VI. A CASE STUDY ON AN ONLINE GRADING SYSTEM

Consider a homework grading example in an online course management system with the following policies:

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

(Rule Collecting Phase)
$at \leftarrow a$'s action type
$p \leftarrow \gamma(at)$
$RULE_{UA} \leftarrow$ user authorization rules $UARule$ found in p
$RULE_{AV} \leftarrow$ action validation rules $AVRule$ found in p
(User Authorization Phase)
for all $rules$ in $RULE_{UA}$ **do**
  Extract the path rule $(ObjRole, DName)$ from $rules$
  Determine the object $o \in O$, whose role is $ObjRole$
  Extract dependency path expression $dpath_b$ in $DPATH_B$ from $DName$ using λ_O function
  Determine vertices by tracing base provenance data PD_B through the paths expressed in $dpath_b$ that start from the object o using δ_O function
  Determine the truth value by evaluating the result against the rule
end for
$UAuth \leftarrow$ a combined truth value based on conjunctive or disjunctive connectives between rules
(Action Validation Phase)
for all $rules$ in $RULE_{AV}$ **do**
  Extract path rules $(ObjRole, DName)$ from $rules$
  for all path rules extracted **do**
    Determine the object $o \in O$, whose role is $ObjRole$
    Extract dependency path expression $dpath_b$ in $DPATH_B$ from $DName$ using λ_O function
    Determine vertices by tracing base provenance data PD_B through the paths expressed in $dpath_b$ that start from the object o using δ_O function
  end for
  Determine the truth value by evaluating the results of all the extracted path rules
end for
$AVal \leftarrow$ a combined result based on conjunctive or disjunctive connectives between rules
Evaluate a final truth value of $UAuth$ and $AVal$ using conjunctive connective

1) Anyone can upload a homework. 2) A user can replace an old version of a homework with a new version (versioning control) if the user is the author of the old version and the old version has not been submitted. 3) An author can submit her homework (origin-based control) if it was not submitted already. 4) A user can review only a submitted homework (workflow control) if she is neither the author nor one of the existing reviewers of the homework (dynamic separation of duty) and the homework has been reviewed less than 3 times and not been graded. 5) A review can be revised if the user created the review and the referred homework is not graded yet. 6) A homework can be graded if it was reviewed at least 2 times. 7) A review can be appended into a grade if the acting user created the grade and the review was made for the homework that the grade is made against.

In PBAC, policies are defined using dependency names. These dependency names are defined in the dependency list and mapped to a combination of other dependency names and paths which eventually can be expressed in regular expressions ($DPATH_B$) that only utilizes base dependency types (D_B). We first define a sample object dependency list to construct policies for the sample case as shown below:

Object Dependency List DL_O:

1) $< wasReplacedVof, g_{replace}.u_{input} >$
2) $< wasSubmittedVof, g_{submit}.u_{input} >$
3) $< wasReviewedOof, g_{review}.u_{input} >$
4) $< wasRevisedVof, g_{revise}.u_{input} >$
5) $< wasGradedOof, g_{grade}.u_{input} >$
6) $< wasAppendedVof, g_{append}.u_{src} >$
7) $< wasOneOfReviewOf, wasRevisedVof * .g_{review}.u_{input} >$
8) $< wasAuthoredBy, wasSubmittedVof?.wasReplacedVof * .g_{upload}.c >$
9) $< wasReviewedBy, wasReviewedOof^{-1}.g_{review}.c >$
10) $< wasCreatedReviewBy, wasRevisedVof * .g_{review}.c >$
11) $< wasGradedBy, wasAppendedVof * .g_{grade}.c >$

The dependency list items 1 - 6 define some dependency names and their dependency path patterns using base dependencies. Items 7 - 11 define additional dependency names using both base dependencies and previously defined dependency names. Based on this object dependency list, the sample policies are presented using the proposed policy specification grammar PG as follows:

Sample Policies:[5]

1) $allow(au, upload, o) \Rightarrow$ true.
2) $allow(au, replace, o) \Rightarrow au \in (o, wasAuthoredBy) \wedge |(o, wasSubmittedVof)| = 0.$
3) $allow(au, submit, o) \Rightarrow au \in (o, wasAuthoredBy) \wedge |(o, wasSubmittedVof)| = 0.$
4) $allow(au, review, o) \Rightarrow au \notin (o, wasAuthoredBy) \wedge au \notin (o, wasReviewedBy) \wedge |(o, wasSubmittedVof)| \neq 0 \wedge |(o, wasReviewedOof^{-1})| < 3 \wedge |(o, wasGradedOof^{-1})| = 0.$
5) $allow(au, revise, o) \Rightarrow au \in (o, wasCreatedReviewBy) \wedge |(o, wasOneOfReviewOf.wasGradedOof^{-1})| = 0.$
6) $allow(au, grade, o) \Rightarrow ((|(o, wasReviewedOof^{-1})| \geq 2 \wedge |(o, wasGradedOof^{-1})| = 0).$
7) $allow(au, append, o_{src}, o_{ref}) \Rightarrow au \in (o_{src}, wasGradedBy) \wedge (o_{src}, wasGradedOof) = (o_{ref}, wasOneOfReviewOf).$

As discussed earlier, transaction data is different from a request in that transaction data knows exactly what objects are used and generated, while a request does not include any objects that will be generated from the performed request. In the list below, we show some sample transaction data and the matching base provenance data. The OPM graph diagram of the resulting base provenance data is illustrated in Figure 4.

Sample Transactions and equivalent Base Provenance Data:

1) $(au_1, upload1, o_{1v1})$: $< upload1, au_1, c >$, $< o_{1v1}, upload1, g_{upload} >$
2) $(au_1, replace1, o_{1v1}, o_{1v2})$: $< replace1, au_1, c >$, $< replace1, o_{1v1}, u_{input} >, < o_{1v2}, replace1, g_{replace} >$
3) $(au_1, submit1, o_{1v2}, o_{1v3})$: $< submit1, au_1, c >$, $< submit1, o_{1v2}, u_{input} >, < o_{1v3}, submit1, g_{submit} >$
4) $(au_2, review1, o_{1v3}, o_{2v1})$: $< review1, au_2, c >$, $< review1, o_{1v3}, u_{input} >, < o_{2v1}, review1, g_{review} >$
5) $(au_3, review2, o_{1v3}, o_{3v1})$: $< review2, au_3, c >$, $< review2, o_{1v3}, u_{input} >, < o_{3v1}, review2, g_{review} >$
6) $(au_2, revise1, o_{2v1}, o_{2v2})$: $< revise1, au_2, c >$, $< revise1, o_{2v1}, u_{input} >, < o_{2v2}, revise1, g_{revise} >$
7) $(au_5, grade1, o_{1v3}, o_{4v1})$: $< grade1, au_5, c >$, $< grade1, o_{1v3}, u_{input} >, < o_{4v1}, grade1, g_{grade} >$
8) $(au_5, append1, o_{4v1}, o_{2v2}, o_{4v2})$: $< append1, au_5, c >$, $< append1, o_{4v1}, u_{src} >, < append1, o_{2v2}, u_{ref} >$, $< o_{4v2}, append1, g_{append} >$

VII. Extended Models

The proposed base model is a foundational model for provenance-based access control and can be further extended for additional security enhancements. As identified earlier, one crucial extension that will enhance the base model is allowing **user-declared provenance data** (PD_U in Figure 3) in addition to the system-computed base provenance data.

[5]We assume users can access only the newest object versions in the system. If not, policies like 2) in the sample will need an additional rule to make sure the object or the newer versions of the object have not been used for a submission.

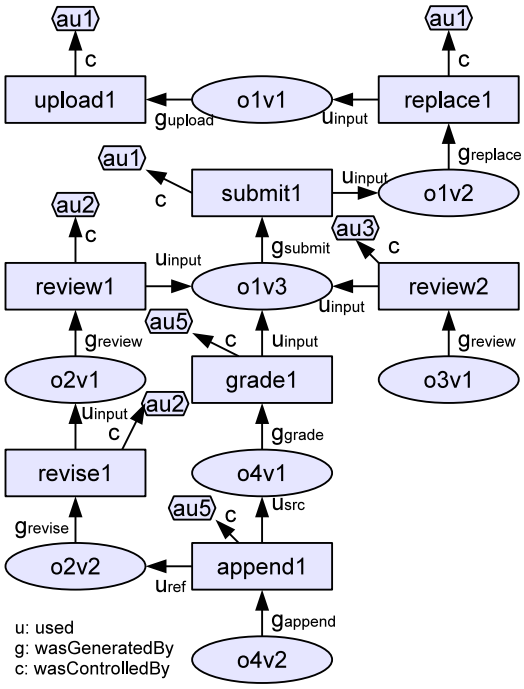

Fig. 4. Sample Provenance Data for Online Grading System in OPM Graph

This means a user can declare a specific dependency between entities using a dependency name that is predefined by the system and available to the user. The user-declared dependency may cause conflict with the dependencies with the same name that are computed using base provenance data. For example, in Figure 4, a user au_2 may declare another user, say au_6 as an actual reviewer who generated a review o_{2v1} for the object o_{1v3} while the system-computed dependency will point to au_2 as a reviewer based on the base provenance data. One approach to address this issue is by explicitly identifying the intentions of the declaration. For this, we believe there are at least three types of intentions: ***inclusive, exclusive*** and ***denying*** types. The inclusive-intent dependency means, for example, both au_2 and au_6 are considered as reviewers who created o_{2v1}. If au_2 declared au_6 as an exclusive reviewer, the system computed dependency will be voided. In addition, if allowed, au_2 may deny that he is not the actual reviewer. Furthermore, this extended model also needs to resolve an authorization issue of who can declare what kind of dependency intentions under which circumstance. We believe the proposed base model provides a concrete foundation for this extension.

Another interesting extension is including **acting user dependencies** in addition to object dependencies. Unlike object dependencies, acting user dependencies utilize acting users as a starting node of a path pattern. This means the rules using acting user dependencies are based on history of the user's previous actions that are independent from the objects related to a request. While this type of rules may not require dependency paths to be traced, the necessary information is captured in provenance data and readily available for this kind of queries. This extension is likely to allow users' activity history-based access controls using provenance data.

The third extension we have identified is **policy retrieval** which concerns how to discover necessary policies using provenance data in case the policies are not readily available to the system. For example, if a system enforces originator control policy for certain objects, the system can use provenance data to find all the contributors of an object from whom the system can retrieve relevant polices. This provenance-based policy retrieval enriches the base model which assumes policies are given to the system. Further developments on this extension will enhance the proposed model.

VIII. Related Works

Provenance is captured and utilized in many different application domains [7], [11], [18]. Based on the intrinsic characteristics of each application domain, different models for provenance are defined. Ni et al [22] propose a general provenance model to capture provenance information but do not explicitly capture the causal dependencies between the model components. OPM [20] is a provenance model that captures provenance data in terms of causality dependencies between the provenance data model components and allows different profile definitions to be constructed to capture multi-domain environments, one of which is in distributed systems [14]. Park et al [23] utilized OPM to capture and express provenance information in the group-centric secure information sharing environment [19]. In this paper, our proposed access control model utilizes OPM to capture the dependencies on which access control policies and decisions are based.

Various aspects of provenance security have been discussed in the literature [6], [16], [17], [27]. Most of the related works mainly consider protecting provenance data. Approaches include building policy languages that provide fine-grained access control [22] as well as allow the generation of query-expressions of paths of arbitrary length on provenance graph [8]. Other works focus on protecting the sensitivity of path information on the provenance graph using redaction [9] or surrogate graphs [5]. To the best of our knowledge, our work is the first to provide security protection to regular data by utilizing provenance data and accommodating the intrinsic property of causality dependency.

History-based Access Control (HBAC) models provide access control to data objects based on the action and request history of the subjects [2], [4], [13]. In HBAC policies, a subject's request is decided based on what actions and action-requests the subject had performed before. In this context, the main motivation is to differentiate the "goodness" of subjects from their past behaviors. Such information can be retrieved from provenance data. More specifically, it is captured in transactions data in our model. In contrast, PBAC emphasizes the history of data objects, which creates dependency chains, and utilizes the intrinsic dependencies, which can also be extracted from transactions data, between these objects for access control purposes.

Separation of Duties (SOD) is extensively discussed in the literature [26]. Transaction Control Expression (TCE) [24], [25] provide a solution for dynamic SOD (DSOD) in a role-based environment through predefined sets of transaction

expressions that are attached to objects and subsequently become the objects' history once executed. Based on the stored executed transactions data, access control decisions can be made to enforce DSOD appropriately. Such transactions history along with corresponding DSOD enforcement can be captured and handled in PBAC. Predefined structure of TCEs can also be used to enforce workflow control. Elements of workflow requirements exhibited by TCE can also be expressed in PBAC through the use of predefined dependency names in the specification of policies.

There are also several implementation frameworks for capturing provenance data with the intent of security in mind. The PASS system [21] aims to capture provenance information at the file level. The PLUS system [10] captures provenance at the application level and use the information for taint analysis to handle insider threat [3]. In order to retrieve information from the provenance storage, many implementation of query languages are available. Park et al [23] employed SPARQL [15] with GLEEN [12] in a group collaboration environment. PQL [1] is a language in development that would provide useful functionalities for provenance data queries.

IX. Conclusion

In this paper, we have defined a family of models for provenance-based access control, then further developed a base model $PBAC_B$ that utilizes object dependencies that are computed based on transactions to evaluate access requests. In the model, we introduced a novel approach for policy specification and access evaluation by utilizing abstracted dependency names and matching dependency path patterns that are expressed using regular expressions. We believe the proposed model allows highly expressive policy specification but simple and effective access control management at the same time. The proposed model is the first effort in this line of work and presents a strong foundation for promising future research.

Acknowledgement

This work is partially supported by NSF CNS-1111925.

References

[1] Pql-path query language. http://www.eecs.harvard.edu/syrah/pql/. Accessed: 03/31/2012.

[2] M. Abadi and C. Fournet. Access control based on execution history. In *In Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 107–121, 2003.

[3] M. Allen, A. Chapman, L. Seligman, and B. Blaustein. Provenance for collaboration: Detecting suspicious behaviors and assessing trust in information. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 342 –351, oct. 2011.

[4] A. Banerjee and D. A. Naumann. History-based access control and secure information flow. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, International Workshop (CASSIS 2004), Revised Selected Papers, volume 3362 of Lecture Notes in Computer Science*, pages 27–48. Springer-Verlag, 2005.

[5] B. Blaustein, A. Chapman, L. Seligman, M. D. Allen, and A. Rosenthal. Surrogate parenthood: protected and informative graphs. *Proc. VLDB Endow.*, 4(8):518–525, May 2011.

[6] 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.

[7] 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.

[8] 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.

[9] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 93–102. ACM, 2011.

[10] 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.

[11] 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.

[12] L. Detwiler, D. Suciu, and J. Brinkley. Regular paths in sparql: querying the nci thesaurus. In *AMIA Annual Symposium Proceedings*. American Medical Informatics Association, 2008.

[13] G. Edjlali, A. Acharya, and V. Chaudhary. History-based access control for mobile code. In *IN ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY*, pages 38–48. ACM Press, 1998.

[14] P. Groth and L. Moreau. Representing distributed systems using the open provenance model. *Future Generation Computer Systems*, 27(6):757 – 765, 2011.

[15] S. Harris and A. Seaborne. Sparql 1.1 query language w3c working draft, jan 2012. http://www.w3.org/TR/sparql11-query/. Accessed: 03/31/2012.

[16] 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.

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

[18] 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.

[19] R. Krishnan, R. Sandhu, J. Niu, and W. Winsborough. Towards a framework for group-centric secure collaboration. In *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, pages 1 –10, nov. 2009.

[20] 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.

[21] 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.

[22] 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.

[23] J. Park, D. Nguyen, and R. Sandhu. On data provenance in group-centric secure collaboration. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 7th International Conference on*, pages 221 –230, oct. 2011.

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

[25] R. Sandhu. Separation of duties in computerized information systems. In *IN DATABASE SECURITY IV: STATUS AND PROSPECTS*, pages 179–189. North-Holland, 1990.

[26] 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.

[27] 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.