Integrated Provenance Data for Access Control in Group-centric Collaboration

Dang Nguyen, Jaehong Park and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio
dnguyen@cs.utsa.edu, jae.park@utsa.edu, ravi.sandhu@utsa.edu

***Abstract*—In a provenance-aware environment, as data objects are created and used, the transaction information is captured as provenance data. Provenance-based access control utilizes the captured provenance information to control access to the underlying data. In a group-centric collaboration environment, data objects are shared and modified by multiple organizations/systems while the relevant provenance data are captured and stored in the local systems. While captured provenance data are readily available for access control within the local system, provenance-based access control in a group-centric collaboration environment requires integrated use of provenance data from other collaborating systems for effective access control. However, some provenance information maintained by a system may be too sensitive to be directly viewed or used by other systems. In this paper, we demonstrate and discuss the issue relating the incorporation of an access control model in the context of group-centric secure collaboration environment. We also discuss two potential solution approaches and their significance in building the foundation for further research.**

I. INTRODUCTION

Provenance is a term that originates in the art world to refer to the origin and history of ownership of a valued object or work of art or literature [2]. The concept, when adopted digitally, can be informally defined as the documentation of the origin of a data object and the processes that influence and lead to any particular state of that object. Digital or data provenance first found its use in the database community. Over time, data provenance found utility in many computer science areas such as semantic webs, workflows, etc. As evident from the community's increasing efforts in studying different aspects of data provenance, the significance of data provenance should be expected to arise in any application that involves data, which almost means any application these days.

Today, information sharing plays a significant role in governing daily functions, from trivial to critical in scope, in both the public and private sectors. It is at the heart of the notion of collaboration, which comes in various shapes, forms and sizes. In order to achieve effective collaboration, secure information sharing is essential. Group-centric collaboration [12] is a promising approach for addressing this issue. In this approach, authorization of users and controlled flow of data objects are vital. Thanks to data provenance utilities, the incorporation of provenance-awareness into the group-centric secure collaboration environment can effectively assist such tasks. For example, pedigree and usage tracking can ensure that a data object would not be accessed by any party that can potentially cause a conflict of interests violation.

The collected provenance data can further enhance the security of the environment when employed as a basis for access control decisions. The Provenance-based Access Control (PBAC) framework, introduced in [19], is a current effort toward that goal. Fundamentally, the framework utilizes the causality dependencies of objects that can be extracted from the provenance data collected in the system. Based on these, the framework creates new constructs, which are termed abstracted names of dependency path patterns, as control units to regulate access to the underlying data. PBAC is fully capable of providing security capabilities such as origin-based control, dynamic separation of duties, workflow control, and object versioning. We believe the framework elevates security by facilitating additional capabilities beyond those available in traditional access control models.

In this paper we attempt to incorporate the base model for PBAC into the Group-centric collaboration environment. As initially conceived, PBAC only deals with a single provenance-aware system. In group collaboration, there are multiple participating systems that collect, store, and maintain their own provenance data. In many cases, such provenance data are stored in the local systems. Consequently, this presents challenges to the integrated use of provenance data for seamless access control. In this paper, we explore these issues with the use of a simplified, generic collaboration scenario. We also discuss our insights and propose several potential approaches to address these issues.

II. BACKGROUND

In this section, we discuss three preliminary topics that are essential for understanding the subsequent sections. In particular, we first describe the Open Provenance Model for representation of provenance data. Next, we describe the base model for Provenance-based access control ($PBAC_B$) that can be incorporated into the aforementioned collaboration environment. Finally, we review the concept of Group-centric collaboration.

A. Open Provenance Model

The Open Provenance Model (OPM) [14] is the result of the community's efforts in creating a standard, technology-agnostic definition of provenance for data capture and repre-

IEEE IRI 2012, August 8-10, 2012, Las Vegas, Nevada, USA

978-1-4673-2284-3/12/$31.00

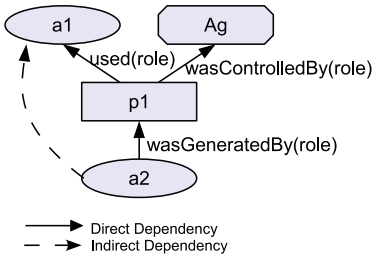

Fig. 1. OPM Causality Dependencies

sentation purposes. We utilize several core OPM notions for such purposes in the group collaboration environment and in the design of $PBAC_B$.

The OPM model aims to represent provenance data in the form of a directed-acyclic graph (DAG). In essence, the model captures the main components of transactions and associating relations with the node and edge entities in such a graph. In general, the node entities are distinguished with different graphical representations: artifacts are represented by ellipses, processes by rectangles, and agents by octagons. The edges are divided into two categories: direct and indirect dependency edges. Direct edges, including $used(Role)$, $wasGeneratedBy(Role)$, and $wasControlledBy(Role)$, are system-captured and represented by solid lines. The other edge types are indirect and represented by dash-lines. These indirect edges are either system-computed or user-declared.

In Figure 1, we demonstrate how the entities relate graphically in a simplified generic use case. The agent Ag controlled the process $p1$ which used the artifact $a1$ to generate the new artifact $a2$. The readers should note that the direction of the arrows specifies a causality relationship instead of a data flow. The source of the arc represents the effect while the destination represents the cause.

As it is designed to be technology-agnostic, the OPM model does not dictate specific semantics for its constructs. Rather, semantics assignments are left to application-specific specifications. The OPM model enables this through the use of a $Role$ parameter associated with each direct dependency edge. For example, an object may have a dependency 'wasGeneratedBy(add)' with an *add* process, meaning that the object was generated by the *add* action (as opposed to the *merge* action).

B. Base Model for Provenance-based Access Control ($PBAC_B$)

Various works from different fields in the literature affirm the benefits of incorporating provenance into a system. The produced utilities include, but are not limited to, pedigree, usage tracking, and object versioning. While protection of provenance data is essential, many enhanced capabilities to protect the underlying data can also be found from applications of provenance data, e.g, origin-based access control, separation of duties, etc.

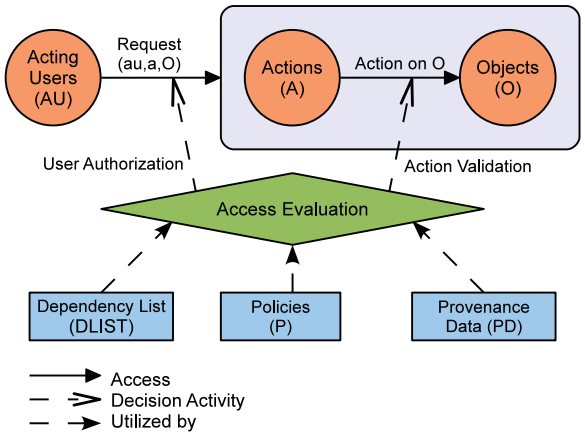

Fig. 2. $PBAC_B$ Components

Previously, we proposed a general foundation for access control based upon provenance data via semantical constructs called abstracted names for dependency path patterns [16]. Essentially, these constructs provide abstractions over semantical relationships between multiple data objects. These relationships arise naturally from the causality dependency provenance data capture in different application domains.

For the remainder of this subsection, we will provide descriptions of the aforementioned semantical constructs and elaborate their usage in the context of $PBAC_B$. Figure 2 exhibits the core components of the model.

1) Model Components: We provide brief definitions of the model components in Figure 2 as follows:

A **Request** for access consists of an acting user, the user's requested action instance, and a set of action-target objects. For more specific semantics, each object is assigned a role specific to the type of the action instance. **Acting Users** (AU) represent the users of a system and the corresponding subjects they use to interact with the system. **Action instances** (A) represent the set of system supported operations, which are initiated and performed by users in AU. **Objects** (O) represent the set of data objects upon which users in AU can perform operations in A.

Provenance data (PD) consist of base provenance data and user-declared provenance data. Essentially, **base provenance data** (PD_B) are transactions of performed actions that are captured in OPM format. Storing the provenance data in sets of triples also allows tracing mechanisms that can traverse the DAG provenance graph bi-directionally in time (i.e., forward or backward). Such capability is essential to policy specification in our access control model. In contrast to PD_B, **user-declared dependency data** (PD_U) represents provenance data that is beyond the capability of the system to capture or compute from transactions. Such provenance data can only be manually declared or specified by the acting users. These are typically specified via indirect dependency edges.

The semantical relationship between provenance data entities serve as the control units for access control. Built

on the stored provenance data triples, the system is also required to maintain **Dependency Lists** ($DLIST$) as pairs of abstracted dependency names ($DNAME$) and corresponding dependency path expressions ($DPATH$). $DPATH$ are used to express the semantical relationship between a starting node and all nodes to which the connected paths match such path expressions.

Policies (P) contain rules that are built on $DNAME$ and $DPATH$ to specify access control authorization of the system's data objects.

There are other components and concepts in $PBAC_B$ that we do not discuss here as they are not essential to the discussion of this paper.

2) Model Interaction: With the essential components of the $PBAC_B$ model defined above, we proceed to describe how they interact. When a $request$ is initiated by an *acting user*, the system parses the *action type* of the *action instance* found in the *request* to choose the appropriate *policy*. From the rules contained in the *policy*, all *dependency name* and *dependency path* expressions are extracted and then reduced to basic path expressions of direct dependency edges. Queries embedding these path expressions are then executed against the provenance data store of triples. Query results are then used to make authorization decision.

C. Group-centric Collaboration

The concept of secure information sharing utilizing "group" constructs is introduced and discussed by Krishnan et al in [13]. In such an environment, information is shared and data created in well-defined structures labeled as groups. Under a collaboration context [12], such groups are created and controlled by the involved collaborating organizations.

Krishnan et al focus on authorization issues, particularly the formalization and semantics specification of administrative and usage operations of users and data objects. In particular, they separate the two types of operational tasks into two sub-models with corresponding operations as follows.

- Administrative operations: Establish/Disband for managing the group, Join/Leave/Substitute for managing users/admins, and Add/Remove/Export/Import/Merge for managing objects that are shared or natively created within the groups.
- Usage operations: CreateRO/CreateRW/Kill for data flow control, Read/Update/Create for usage of objects/versions, and Suspend/Resume for controlling usage of objects/versions.

In our previous work [18], we discussed our methodology in capturing the provenance of the above operations in OPM. We recognized there exist many ways of performing such capture depending on the application-specific semantics. In this paper, we proceed to discuss only the usage operational model and its associating operations as they are more relevant to data provenance and more significant in the context of $PBAC_B$.

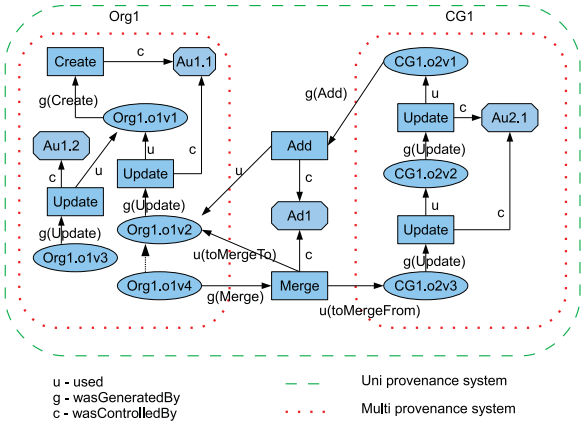

Fig. 3. A Collaboration Scenario Captured in OPM

III. A SIMPLIFIED GROUP-CENTRIC COLLABORATION SCENARIO CAPTURED IN OPM

We describe a simplified scenario of a group-centric collaboration environment that is depicted in Figure 3. Note that the figure only depicts two separate entities, namely $Org1$ and $CG1$, for simplicity. Theoretically, there must exist at least two organizations and one group for a collaboration to take place. Here, we omit the presence of another organization ($Org2$) as its inclusion is not essential for purpose of this paper. We include a user from organization 2, $Au2.1$, within the scenario to indicate a second organization's involvement.

A. Overview

In Figure 3, there are four usage operations that are supported by the provenance systems in both $Org1$ and $CG1$, namely $Create$, Add, $Update$ and $Merge$. A user $Au1$ creates an object $Org1.o1v1$ in the organization. The object is then updated separately to generate two new versions (of the same object), which are $Org1.o1v2$ and $Org1.o1v3$. These objects are updated by $Au1.1$ and $Au1.2$, respectively. The object version $Org1.o1v2$ is then added to the collaboration group so that users from any other organization can access it. A new copy of $Org1.o1v2$ is created and labeled as $CG1.o2v1$. Further updates are performed on this version to generate $CG1.o2v2$ and $CG1.o2v3$ by a group user $Au2.1$. The object version $CG1.o2v3$ is then merged back into $Org1$ and becomes a new version $Org1.o1v4$ of $Org1.o1v2$, the version that was added to the collaboration group earlier. The Add and $Merge$ processes are performed by an administrator user delegated from $Org1.Ad1$. All the transactions involving the objects and versions, processes, and "actors" are captured in OPM format.

Note that some action type may require more than one input object to be specified. Each input object has a different role in regard to the requested action. Such semantics can be addressed by attaching a role specification/label to a $used$ dependency edge. For example, the $Merge$ action type requires two input objects, a source organization object version that was added to the collaboration group and a new group object

version with modifications that is requested to be merged back to the source version. In Figure 3, we specify these different input objects/types through the dependency edges with roles $u(toMergeTo)$ and $u(toMergeFrom)$ respectively.

Based on the OPM graph, we can introduce $PBAC_B$ constructs and mechanisms and discuss how the deployment takes place in the next section. Before that, we identify the two possible types of provenance systems deployment within the scenario setting.

B. Uni-Provenance vs Multi-Provenance Systems

In Figure 3, we depict a uni-provenance system setting with the dashed rounded rectangle and multi-provenance systems setting with the dotted rounded rectangles.

A *uni-provenance* system is assumed to capture all transactions occurring within all entities (groups and organizations) within the environment. As such, any system within the environment is allowed to access any captured provenance data. We acknowledge that in distributed environments such a uni-provenance system is typically infeasible. However, it is suitable for our demonstration of $PBAC_B$ mechanisms.

In *multi-provenance* systems, each entity maintains its own methods of capturing and storing the provenance data of transactions occurring within its domain/boundary. It is simple for a system to access and utilize its own provenance data. However, complications arise when a system accesses provenance data owned by a different system. Solving such issues is essential in the context of $PBAC_B$ integration in Group-collaboration environment.

Note that in multi-provenance systems, the action types which carry information flow across entities (such as Add and $Merge$) can be captured in different ways. For our scenario, we assume such processes are captured by both participating parties and some forms of identification links are established by both participants.

IV. Integration of $PBAC_B$ in Group-centric Collaboration

In this section, we discuss the notion of integrating $PBAC_B$ in a Group-centric collaboration environment. In particular, we first demonstrate how basic $PBAC_B$ mechanisms can be applied in a uni-provenance system. Next, we explore the mechanisms under a multi-provenance systems setting and identify an essential related issue. We will describe the potential approaches to address this issue in the next section.

A. $PBAC_B$ in Uni-Provenance system

In this subsection, we discuss the deployment of $PBAC_B$, under a uni-provenance system setting, into a group-centric collaboration scenario, namely the one depicted in Figure 3.

Typically, a system with $PBAC_B$ will maintain a separate access control policy for each of the user action types that the application system supports. For example, an informal policy for the $Merge$ action type can state:

"An admin user is allowed to merge a group object version to an organization object version if the group object version is either an added copy of the organization object version, or is derived from that copy."

Formally, we can express the above policy in a policy language provided in [18], [19] as:

$$allow(au, merge, o_{from}, o_{to}) \Rightarrow$$
$$o_{to} \in (o_{from}, wasDerivedVersionOfCopyOf)$$

The first part of the policy (before the right arrow) specifies the request form. The form specifies the requesting user (au), the requesting action type ($merge$), and the input objects (o_{from},o_{to}). The second part of the policy provides the decision rule. The roles $from$ and to provides a mapping between the input objects in the request to their respective part in the decision rule. The decision rule specifies whether the input object o_{to} is in the set of nodes reachable from the input object o_{from} through the path expressed by the dependency name $wasDerivedVersionOfCopyOf$.

A dependency name is essentially a meaningful abstraction of path patterns in a provenance graph. A dependency name comprises of either direct dependency edges (such as $used$, $wasGeneratedBy$, or $wasControlledBy$) or other dependency names or a combination of both. Regardless, any dependency name can be reduced to a path comprising solely direct dependency edges.

If a $Merge$ request is to be granted, there must exist a path, as exhibited and reducible by $wasDerivedVersionOfCopyOf$, that starts from o_{from} and ends at o_{to}. In practice, the provenance system generates a query, which takes the starting node and a dependency path pattern, and execute it against the provenance graph. The results of the query can then be used in evaluation with the policy rules to arrive at the final decision regarding the access request.

In $PBAC_B$, we term such patterns $DPATH$. We also use $DNAME$ to further abstract these regular path patterns and facilitate their usages as control units in access control. Such a pair of ($DNAME$::$DPATH$) is contained in a $DLIST$, which is also a main component in the $PBAC_B$ model.

In the example scenario, in order to capture/specify the rule expressed in the provided policy, we can create a pair of ($DNAME$::$DPATH$) instance as

$$(wasDerivedVersionOfCopyOf::$$
$$[g(Update).u] * .g(Add).u)$$

Note that typically a resulting query path that matches the path pattern includes information of the intermediate nodes. Here, we do not show those nodes, such as $CG1.o2v1$ and $CG1.o2v2$, for simplicity. In addition, for $PBAC_B$ purposes, a query, which is evaluated against a provenance graph, only requires the edges to traverse the graph. For better expressiveness, we utilize regular expression based (or regular for short) dependency path patterns of arbitrary lengths. Regular path patterns can enable simple and effective policy specifications.[1]

[1]In our example, "[]" denotes grouping of edges, "*" denotes zero or more occurences of a preceding edge or group, "." denotes concatenation between any two edges/groups. Precedence is high-to-low respectively.

Now, let us consider a sample request

$$Req(au, merge, CG1.o2v3, Org1.o1v1)$$

After parsing the request and obtaining the starting node (in this case, $CG1.o2v3$) and a regular path pattern ($[g(Update).u] * .g(Add).u$) (as obtained from looking up $wasDerivedVersionOfCopyOf$ in $DLIST$), a query, which embeds these two components, is generated and evaluated against the provenance graph. Such a query will obtain a result set which contains $Org1.o1v2$ because there exists the following path in the graph

$$< CG1.o2v3 > [g(Update).u.g(Update).u.g(Add).u] < Org1.o1v2 >$$

which is matched by $([g(Update).u] * .g(Add).u)$.

Evaluating the result set against the policy rule shows that $Org1.o1v1 \notin \{Org1.o1v2\}$. Therefore, the request is disallowed.

On the other hand, a request to merge $CG1.o2v3$ and $Org1.o1v2$ is allowed. Such a $Merge$ transaction is executed and then captured in the provenance graph. Here, a copy object version of $CG1.o2v3$ is created in the organization as $Org1.o1v4$. There exists an indirect dependency between this object version and the source object version $Org1.o1v2$. This is depicted with an indirect edge between the two corresponding nodes in the provenance graph.

B. $PBAC_B$ in Multi-Provenance Systems

In a multi-provenance systems setting, since each entity maintains its own provenance system, provenance data of one system may not be readily available for a seamless $PBAC_B$ usage in another system. In this subsection, we demonstrate this problem through another example within the scenario depicted in Figure 3. The proposed solution approaches for this issue are discussed in the following section.

To demonstrate the limitation of multi-provenance systems in enabling $PBAC_B$, let us introduce a new policy that regulates $Update$ of object versions within $CG1$ to avoid potential conflict of interests:

"A group user is allowed to update an object version if he is not the creator of the original organization version of that group object version."

We also introduce the formal policy statement as:

$$allow(au, update, o) \Rightarrow au \notin (o, creatorOfOriginalVersionOf)$$

where the following mapping can be found in $DLIST$

$$(creatorOfOriginalVersionOf :: [g(Update).u] * .g(Add).u.[g(Update).u] * .g(Create).c)$$

Under this policy, any request to $Update$ a group object version will generate a query embedding the object version as the starting node and the above regular path pattern. Executing the query against the provenance graph in Figure 3 with any of the nodes ($CG1.o2v1$, $CG1.o2v2$, $CG1.o2v3$) will return a result set of $\{Au1.1\}$. Obtaining such result set from traversing the provenance graph with the regular path pattern may not be possible if $Org1$ does not readily release access to its portion of the provenance graph.

More specifically, a query generated in $CG1$ can only traverse the portion that is reachable from the group portion of the provenance graph. That is, only the first subpart of the regular path pattern, in quotation,

$$"[g(Update).u] * .g(Add)".u.[g(Update).u] * .g(Create).c)$$

can be traversed up to $Org1.o1v2$. The remaining path pattern cannot be traversed as such information belong to the organization, which opts to not share its provenance data for various reasons. Without resolving such issue, any request to $Update$ an object version in $CG1$ would be falsely disallowed.

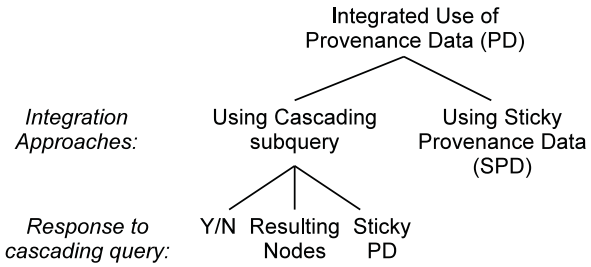

Fig. 4. A Taxonomy of Provenance Data Integration in Multi-Provenance Systems

V. Integrated Uses of Provenance Data in Multi-Provenance Systems

For seamless use of provenance data in multiple provenance systems, we believe there are at least two approaches. As shown in Figure 4, one approach is by utilizing cascading subqueries. The other approach is by utilizing sticky provenance data which are transmitted together with an associated data when the associated data is added/moved to another collaborating entity (organization or collaboration group).

A. Using Cascading Subquery

When an access request is parsed and a corresponding query is generated, the query is executed and evaluated against the local provenance system. Certain path patterns, however, would lead to objects that are added copies from different system entities. The local provenance system then does not contain sufficient information to make an access decision. It becomes necessary to ask for provenance information in another provenance system entity. In other words, subqueries need to be generated and evaluated against the provenance graphs maintained by other systems. These queries are generated from the original query with modifications of the path patterns to remove those already traversed. As a provenance trace can potentially span across many system entities, different subqueries may become necessary every time such a cross-system transition occurs. We call such a query as a *cascading subquery*.

In a uni-provenance setting, evaluating a query returns a set of resulting nodes upon which access control decisions are made. In a multi-provenance setting, once a provenance system receives a cascading subquery, there are at least three different ways such a subquery can be handled. For each case, different granularity of additional information is required from the requesting entity. Specifically, the receiving system can utilize its provenance data to return

- Y or N: ($startingNode_{new}$,$dPath_{new}$,$rule_{new}$) must be transmitted.
- Resulting Nodes: ($startingNode_{new}$,$dPath_{new}$) must be transmitted.
- Provenance Data Set: ($startingNode_{new}$) must be transmitted.

In the first case, the receiving provenance system is asked to make the access control decision. This could be the case when either the requesting entity does not possess enough computation resources to perform the evaluation itself or the receiving entity does not allow direct access to its provenance data. One such an example could be found in a group collaboration between a government agency and a contracting company where the agency is not allowed to reveal the details of its provenance data. For evaluating such a subquery, the receiving provenance system requires additional information. In particular, it requires the recomputed $startingNode_{new}$, $dPath_{new}$, and $rule_{new}$. Here, the $rule_{new}$ is necessary for the receiving provenance system to be able to make a decision for the requesting entity.

In the second case, the receiving provenance system requires the recomputed $startingNode_{new}$ and $dPath_{new}$ to evaluate the subquery but does not need make any access control decision for the requesting entity. This could be the case of a collaboration between a government agency and a contract company where the agency (requesting system) requires the contract company (receiving system) to reveal its provenance data as the agency may not want to reveal (part of) its access control policy.

In the third case, the receiving provenance system receives only the recomputed $startingNode_{new}$. Here, the system returns all provenance entities reachable by a recomputed $startingNode_{new}$. Similar to the second case, access control decisions are made by the requesting system.

Below, we show the first approach using the example scenario we discussed above. More specifically, we identify the recomputed $startingNode_{new}$ as $Org1.o1v2$. This node is computed from traversing the first part of the regular path pattern (in quotation). The recomputed $dPath_{new}$ is the remaining part of the regular path pattern. The recomputed $rule_{new}$ can then be expressed as

$$au \notin (Org1.o1v2, u.[g(Update).u] * .g(Create).c)$$

Depend on the type of interaction, combination of these recomputed components are assembled in a cascading subquery.

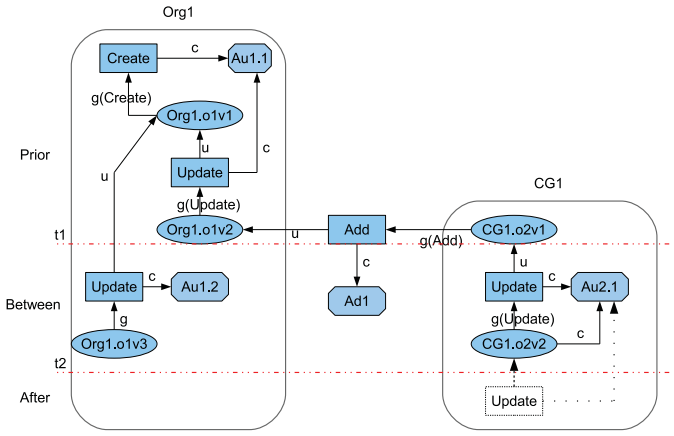

Fig. 5. Sticky Provenance Data in Simplified Scenario

B. Using Sticky Provenance Data

Another approach for resolving the stated issue is through the use of sticky provenance data. In this subsection, we discuss the concept of sticky provenance data in the context of a modified version of the scenario discussed earlier in the paper.

1) Modified Scenario with Sticky Provenance Data: As depicted in Figure 5, the example scenario introduces some temporal aspects of sticky provenance data in group-centric collaboration environment. In particular, there are essentially three phases where the transactions occur and related provenance data are captured. These phases, along with the related transactions, are:

- *Prior to* ***t1***: In this phase, the artifact $Org1.o1v1$ is created and then updated into $Org1.o1v2$ by the acting user $Au1.1$. The artifact $Org1.o1v2$ is then added to the collaboration group $CG1$ to generate a new copied object $CG1.o2v1$. This transaction is controlled by the administrator $Ad1$.
- *Between* ***t1*** *and* ***t2***: In this phase, transactions occur simultaneously and independently in $Org1$ and $CG1$. In $Org1$, the artifact $Org1.o1v1$ is updated by another acting user $Au1.2$ to generate a new version $Org1.o1v3$. In $CG1$, $CG1.o2v1$ is updated by $Au2.1$ to generate $CG1.o2v2$.
- *After* ***t2***: In this phase, a request to update $CG1.o2v2$ is initiated. At this point in time, no access control decision had been made.

In phase 1, when $CG1.o2v1$ is added to $CG1$, all the provenance data of $Org1.o1v2$ (including the add transaction information) are collected and transmitted to $CG1$ together with the $Org1.o1v2$ then stored with $CG1.o2v1$. We call the transmitted provenance data **sticky provenance data (SPD)**.

With sticky provenance data available in the local system, a cascading query may not be necessary to obtain provenance information necessary in making access control decisions. Rather, a locally generated query can be evaluated and completely executed against the local provenance graph similar

to how a query under a uni-provenance system is treated. However this may not always be the case. The sticky provenance data of an object/version contains all the provenance information of that object/version up to the point in time when the information flow takes place. This means the transactions on the source data ($Org1.o1v2$ in our example) that occur after the information flow (phase 2 in the example) are not captured in the sticky provenance data unless the sticky provenance is constantly modified to reflect all the transactions that occur on all the previous versions ($Org1.o1v1$ in the example) of the added object ($Org1.o1v2$) and the added object itself.

2) Benefits of Sticky Provenance Data: There are multiple advantages associated with using sticky provenance data. One main advantage is the elimination of repeating computational efforts required from the source system. More specifically, once an organization data object is added to a group, policy rules for subsequent access request may base the decision evaluation on the provenance information of the organization data object. As discussed above, if no SPD is used, some forms of cascading subqueries may be passed to the organization asking for results. The organization then would have to spend its computation resources in satisfying such requests. By sending sticky provenance data along with the data object copy, the requests and queries can be evaluated and answered locally.

3) Issues of Sticky Provenance Data: There are also multiple issues related to the static nature of sticky provenance data. Essentially, once sticky provenance data is moved to a new system entity along with the copy of an associating data object, the sticky provenance data only contains provenance information of the copied data object up to the point in time when such information flow occurs. Any processes occurring upon that object version thereafter are not captured by the sent sticky provenance data.

While this should be fine for the queries that need only backward tracings of provenance data, if a query requires to check all the transactions occurred against all of the related objects, forward tracings of provenance data may be necessary and the available sticky provenance data are not likely to be enough for access control decision. For example, suppose a policy rule which dictates that no $Update$ actions can be performed on a group version if the original organization object version had been updated in some ways (application-specific context). In the context of the scenario depicted in Figure 5, granting access to a request to update $CG1.o2v2$ requires no modifications had been done on $Org1.o1v2$. If the query only checks provenance data obtained in the SPD of $(CG1.o2v1)$ stored in $CG1$, then the access request is granted. However, as shown in $Org1$, some modifications had been done on $Org1.o1v2$ during phase 2 and generated $Org1.o1v3$ at the current point in time. Such provenance information is not captured in the SPD of $(Org1.o1v2)$ because at the point in time when SPD of $(Org1.o1v2)$ was sent to $CG1$, such process had not occurred.

For sticky provenance data to be useful and reliable in such use cases, we require some mechanisms to keep the SPD up-to-date at all points in time or require additional tracings of provenance data that are not reflected in SPDs. In practice, keeping up-to-date SPD could be quite costly or even unrealistic. This is further complicated as the complexity of the group-centric collaboration environment becomes larger.

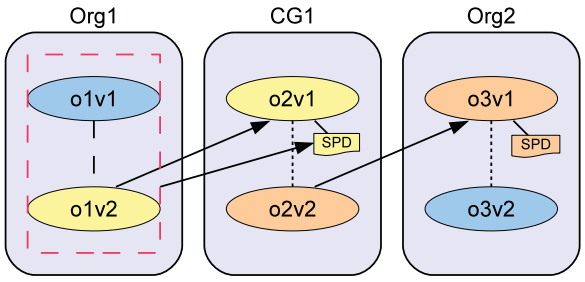

Fig. 6. A "Sticky" Multi-Provenance Scenario

Consider the scenario in Figure 6 which depicts information flow across three provenance systems where real-time updates of SPD are not available. Here, an object $o1v2$ is added to the collaboration group. Within the group, the added copy $o2v1$ is further updated and eventually generates $o2v2$, which is then added to a different organization $Org2$. In this information flow scenario, when $o1v2$ is added to $CG1$, all provenance data of $o1v2$ and the add transaction information are transmitted to $cg1$ together with $o1v2$ and stored in SPD of $o2v1$. When $o2v2$ is added to $Org2$, it seems logical to combine SPD of o2v1 to SPD of o3v1. However, this is largely dependent on the application-specific context and may not be the case. In other words, SPD of $o3v1$ may or may not contain SPD of $o2v1$. For example, while a government agency $Org1$ may trust $CG1$ and shares its provenance data with $CG1$ using SPD, it may not allow further dissemination of the shared SPD to a contract company $Org2$. Furthermore, as the number of local provenance systems increases, the configuration and efforts required for updating SPD can grow exponentially in complexity.

Regardless, we believe that sticky provenance data can prove to be useful in some cases where forward tracing is not necessary.

VI. Discussion and Related Work

We discussed the concept of sticky provenance data that accompanies any related data objects being moved across systems. Although this approach helps solve some of the problems in integrated provenance usage, it also raises many issues of its own. We identify several issues below.

- Sanitization of sticky provenance data: Depending on the organization policies, some information in the related provenance graph may be too sensitive for passing along with the data itself. It is up to the organizations to implement their own procedure of protecting such information. The papers [5], [8] focus on these issues.
- Usage control of sticky provenance data: The organization that owns and allows some of its provenance information to be shared in a different organization is obliged to establish mutual trust. However, that organization is

entitled to specify the manners in which such provenance information can be used in a different system.

In this paper, we also assume that in each local provenance system, the capture granularity of provenance data are essentially similar and therefore the communication is easily established. However, it has been shown in the literature that different provenance system designs can capture provenance data at many different levels of granularity. For example, the PLUS system [9] captures provenance information at the application layer while the PASS system [15] opts for capture at the file system layer. This presents challenges in integrating these collected provenance data for a common application or usage purpose such as PBAC. Angelino et al [4] identifies and proposes a solution to address these challenges.

Integrated use of provenance data for PBAC requires complex query mechanisms. In particular, the queries need to support regular path patterns of arbitrary lengths. The query language needs to also support complex forms of constructs for additional results transformation purposes. Currently, SPARQL [11] with GLEEN [10] is one such a language for implementing PBAC. PQL [1] is another language in its early development stage that can potentially address the stated requirements.

The $PBAC_B$ model we discussed in this paper is a core model upon which extended models of the $PBAC$ framework [19] can be built. The $PBAC$ framework emphasizes on utilizing provenance data to achieve access control protection to the underlying data. In contrast, various aspects of protection of provenance data have been studied extensively in the literature [3], [6], [7], [17]. Protection of provenance data in the group-centric collaboration environment is also essential. We strongly believe the incorporation of $PBAC_B$, and eventually the $PBAC$ framework, in such environment complements the pursuit of that objective.

VII. CONCLUSION

In this paper, we explored integrated use of data provenance for access control, specifically Provenance-based Access Control, in a group-centric secure collaboration environment. In particular, first, we provided an overview of the $PBAC_B$ model, the group-centric collaboration environment, and the provenance data model. Under the context of group-centric collaboration, we then demonstrated how $PBAC_B$ mechanisms can be deployed in a uni-provenance setting and the issue that arises when the deployment is carried out in a multi-provenance system. We identified and discussed two approaches of using cascading subqueries and using sticky provenance data. We believe the discussions presented in this paper are essential for understanding the issue in integrated use of provenance data for access control in group collaboration and will provide a foundation for further research on this line of work.

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] Provenance definition. http://www.merriam-webster.com/dictionary/provenance/. Accessed: 07/15/2012.

[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] E. Angelino, U. Braun, D. Holland, P. Macko, D. Margo, and M. Seltzer. Provenance integration requires reconciliation. In *4rd USENIX Workshop on Theory and Practice of Provenance*, TaPP'11. USENIX Association, June 2011.

[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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] R. Krishnan, R. Sandhu, J. Niu, and W. H. Winsborough. Foundations for group-centric secure information sharing models. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 115–124, New York, NY, USA, 2009. ACM.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.