An Access Control Model for Protecting Provenance Graphs

Liang Chen, Peter Edwards, John D. Nelson and Timothy J. Norman
dot.rural Digital Economy Hub, University of Aberdeen, UK
{l.chen, p.edwards, j.d.nelson, t.j.norman}@abdn.ac.uk

***Abstract*—Securing provenance has recently become an important research topic, resulting in a number of models for protecting access to provenance. Existing work has focused on graph transformation mechanisms that supply a user with a provenance view that satisfies both access control policies and validity constraints of provenance. However, it is not always possible to satisfy both of them simultaneously, because these two conditions are often inconsistent which require sophisticated conflict resolution strategies to be put in place. In this paper we develop a new access control model tailored for provenance. In particular, we explicitly take into account validity constraints of provenance when specifying certain parts of provenance to which access is restricted. Hence, a provenance view that is granted to a user by our authorisation mechanism would automatically satisfy the validity constraints. Moreover, we propose algorithms that allow provenance owners to deploy fine-grained access control for their provenance data.**

I. Introduction

In computer systems, provenance of data is information about the origin, derivation, or history of the data, which can be used to make informed judgements about quality and authenticity of the data. The ability to record and exploit provenance has now emerged as an important requirement in a variety of systems such as databases, scientific workflow and cloud-based storage. Rather than defining provenance individually for each type of system, there have been efforts to standardise a common data model [11], [13] for representing and exchanging provenance. Typically, such a model can be represented as a graph which captures causal dependencies between different types of entities. Examples of an entity include a file in a file system, or a (computational) process. Thus we use the term of *provenance graph* to refer to provenance that is expressed according to a particular data model.

It has been recognised that security is a critical factor in the wide adoption of provenance technologies [8]. Like any data produced or consumed by computer systems, some parts of a provenance graph may be sensitive, and will need to be protected against unauthorised access. Simply omitting sensitive elements may break the casual dependencies among entities in the provenance graph. As a result, a user who views the graph may not be able to certify the origins of the data item. This suggests that any generated view of a provenance graph that satisfies security policies may need to be repaired, in order to preserve its *validity* in the sense that no false (in)dependencies arise. On the other hand, changing provenance views to preserve validity may lead to security policies not being properly enforced.

Existing research has focused on how to transform a provenance graph into a view that satisfies both security policies and validity constraints, which is referred to as the problem of *provenance sanitization* [5]. Cheney and Perera [5] review recent work on provenance sanitization and conclude that current methods are immature in several aspects including semantics, formal guarantees, and techniques for detecting and resolving conflicting policies. We believe that some of the problems in existing work arise from the fact that satisfying security policies and validity constraints are treated as two separate conditions to be met. This means that it is not clear how to satisfy both conditions in a balanced way. For example, Missier *et al.* [12] have put great effort on developing a grouping operator that enables a provenance graph to be transformed into a valid view, given a set of vertices that is required to be removed. However, the grouping operator often results in a view which does not satisfy strict security policies defined based on the Bell–LaPadula model. As another example, the work of Dey *et al.* [7] requires a provenance owner to reconcile conflicts that can occur between her security requirements and the need to satisfy validity constraints. While the rule-based approach they employed can successfully resolve these conflicts, we believe that this is not necessary if security policies can be designed taking into account validity constraints.

In this paper we propose an authorisation model for controlling access to provenance graphs. Our model is based on the observation that an object within a provenance graph that is to be protected should be a *contractable* subgraph in the sense that contracting this subgraph would result in a valid provenance graph. This means that, if we wish to deny access to a particular object, we can simply supply an authorised user with a view in which the object is contracted. Unlike other approaches, we allow validity constraints associated with the graph to be explicitly considered when specifying certain regions of the graph to which access is restricted. We then divide the whole provenance graph into a set of protection objects that has a particular hierarchical structure for implementing conflict-free access control policies. Specifically, we present an algorithm that enables a provenance owner to create such a hierarchy of objects, and to assign authorised roles to each of these objects. As a result, our approach enables a provenance owner to deploy fine-grained access control for her provenance graph, and the effort required in specifying policies is minimised.

Prior to presenting our model, we introduce some pre-requisite concepts from graph theory, and define the open provenance model.

II. Background

A. Definitions and Notation

A *directed graph* (or *digraph*) is a pair $G = \langle V, E\rangle$, where V is a non-empty set of elements called *vertices*, and $E \subseteq V \times V$ is a collection of ordered pairs of vertices called *edges*. If $e = \langle v, w\rangle$ is an edge of G, then vertices v and w are called *end-vertices* of e, and e is said to *join* v to w. We also say that e is *incident from* v and *incident to* w. For convenience, we will denote edge $\langle v, w\rangle$ by vw. A *path* is a sequence of edges $v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$. A path is a *cycle* if it starts and ends in the same vertex. In this paper we will be concerned with digraphs containing no cycles: namely, *directed acyclic graphs* (DAGs).

If uv and vw are two edges of G, we say that u is an *in-neighbour* of v, and w is an *out-neighbour* of v. We write $N^i_G(v) = \{u \in V \mid uv \in E\}$ to denote the set of vertices that are in-neighbours of v in the graph G. Similarly, $N^o_G(v) = \{w \in V \mid vw \in E\}$ denotes the set of vertices that are out-neighbours of v in the graph G.

Let $G = \langle V, E\rangle$ be a digraph. We say $G' = \langle V', E'\rangle$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. If G' contains *all edges* of G that have end-vertices in V', then G' is said to be the subgraph *induced* by V'. In this case we also say that G' is an *induced subgraph* of G and is denoted by $G[V']$. Let $G = \langle V, E\rangle$ be a digraph, and let $e = vw \in E$. The *contraction* of edge e in G is a process that removes e and merges its end-vertices vw into a new vertex u in such a way that u is an end-vertex of all those edges which were originally incident to v or incident from w. We write G/e to denote a graph that is obtained by contracting an edge e in G. Let $G' = \langle V', E'\rangle$ be a subgraph of G. The contraction of G' in G is essentially a process that obtains a graph from G by contracting all edges in E', and is denoted by G/G'.

Let L be a set of labels. A *labelled digraph* is a pair $G_l = \langle V, E_l\rangle$, where $E_l \subseteq V \times V \times L$ is a set of labelled edges. We will omit the subscript l when it is obvious from context.

B. The Open Provenance Model

There have been many efforts to define a data model for provenance, aiming to support inter-operable interchange of provenance information in heterogeneous environments. The Open Provenance Model (OPM) [13] is one of the results of these efforts, which then served as a basis for the specification of the W3C PROV Data Model [11]. In this paper we have adopted OPM as the basis upon which our security model is built, because it provides core features for representing provenance that other models, such as PROV, inherit. Also, there exist a variety of systems that have implemented OPM. We now introduce core elements of OPM using a graph-based formulation.

Definition 1. *OPM is comprised of three types of entities T:* agent (ag), process (p) *and* artifact (a), *three relationship labels L:* used, wasGeneratedBy (wgb) *and* wasControlledBy (wcb), *and a labelled DAG $G = \langle T, E\rangle$, where $E = \{(p, a, used), (a, p, wgb), (p, ag, wcb)\}$.*

Intuitively, an edge $(p, ag, wcb) \in E$ means that process p was controlled by agent ag, an edge $(a, p, wgb) \in E$ means that artifact a was generated by process p, and an edge $(p, a, used) \in E$ means that artifact a was used by process p, also written as p used a. Figure 1a shows the core structures of OPM, depicting the types of entities and their relationships that the OPM graph can support. Note that two additional dashed edges labelled with wasDerivedFrom and wasTriggeredBy respectively represent inference rules defined in OPM, which are not considered in the construction of our security model, and are omitted henceforth.

Definition 2. *Given the OPM $\langle T, L, G\rangle$, an OPM* instance *is defined by a* provenance *graph $G_i = \langle V_i, E_i\rangle$, where V_i is a set of entities and $E_i \subseteq V_i \times V_i \times L$. Let $\tau : V_i \to T$ be a function that maps an entity to its type, we say G_i is* valid *if for each entity $v \in V_i$, $\tau(v) \in T$, and for each edge $(v, v', l) \in E_i$, $(\tau(v), \tau(v'), l) \in E$.*

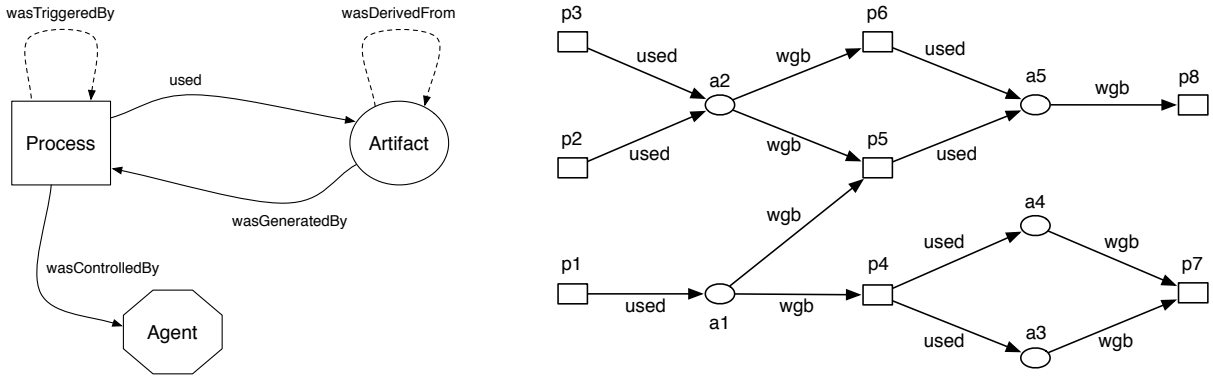
Figure 1b shows an example of a valid provenance graph which will be used as the basis for all further examples in this paper. Note that this is a simplified version of provenance graphs in which vertices ag and edges wcb are excluded. We refer to it as a *simple* provenance graph, and denote it by G^s_i. We write G^c_i to denote a *complete* provenance graph that includes all the core elements of OPM as defined above. We will omit the subscript i when it is obvious from context. In the next section we start by introducing our security model on the basis of G^s, and then discuss how our approach can be easily extended to support G^c in Section IV.

III. An Access Control Model

A. Protection Objects

As in classical access control, we define a *protection object* (or simply *object*) to be data for which an access control policy is specified. In a provenance graph, objects can be parts of the graph that satisfy an important property in terms of authorisation: a view of the provenance graph that a user is authorised to view has to be *valid*. This means that we need to transform the graph to obtain a valid view in which those regions of the graph that the user is denied access to are hidden. One extreme case that satisfies this property is to define each vertex of the graph as an object, which can be simply replaced by a new vertex of the same type when access control policies are enforced. While this provides fine-grained access control, it is unlikely that managing access control for every single vertex of a graph is a feasible solution, given that the number of vertices in a provenance graph is likely to be very large. In general, however, we would like to consider an object to be a subgraph whose edge set is *not* an empty set, and these objects are the focus of this paper.

Given a subgraph G' of a provenance graph G, we use the contraction of G' in G as a way to hide the graph G' in



(a) Core elements of OPM

(b) A valid simple provenance graph

Figure 1. OPM and provenance graph

G. It is important to note that the type of a new vertex that replaces the contracted graph G' cannot be artifact, because there must exist some vertex p in G' with edge (p, ag, wcb), then contracting G' into a vertex a of type artifact gives rise to an edge (a, ag, wcb) which is not valid with respect to OPM. Thus, unless otherwise stated, the contraction of G' always results in a new vertex of type process.

Definition 3. *Let $G^s = \langle V, E\rangle$ be a provenance graph, and let $G' = \langle V', E'\rangle$ be a subgraph of G^s. We say G' is an* object *of G^s if both G' and a graph obtained by contracting G' in G^s are* valid *provenance graphs.*

Figure 2 illustrates three possible objects that can be identified within the provenance graph shown in Figure 1b. Given a provenance graph, we need to consider an appropriate choice of objects with respect to the specification of access control policies. At first sight, it seems reasonable that there should be no *partial* overlap between different protection objects. Consider two objects o_1 and o_2 shown in Figure 2a and Figure 2b that share a common process vertex p_5. Suppose that objects o_1 and o_2 are associated with a set of authorised users U_1 and U_2 respectively, where $U_1 \cap U_2 = \emptyset$. When a user $u \in U_1$ requests to access object o_1, it is not clear whether to return o_1 with vertex p_5 included, since u is denied access to o_2 that contains vertex p_5. A conflict resolution strategy such as DenyOverride or AllowOverride can be put in place to resolve this situation. However, given a large provenance graph, it is not practical to define objects with such overlapping structure, because this requires considerable effort in policy design.

There is one situation where overlapping objects would be reasonable; that is, when one object is completely contained within the other. In this case, objects that have a hierarchical structure can be used to organise authorisation into a hierarchy, thereby providing fine-grained access control. Taking an example in Figure 2, we could grant a user u access to object o_3 but deny access to object o_1. In other words, u would only be able to access the graph o_3/o_1 where o_1 is contracted. While this achieves the same effect as defining two partially overlapping objects o_1 and o_2 with the DenyOverride strategy being employed, it requires less effort to define objects in general.

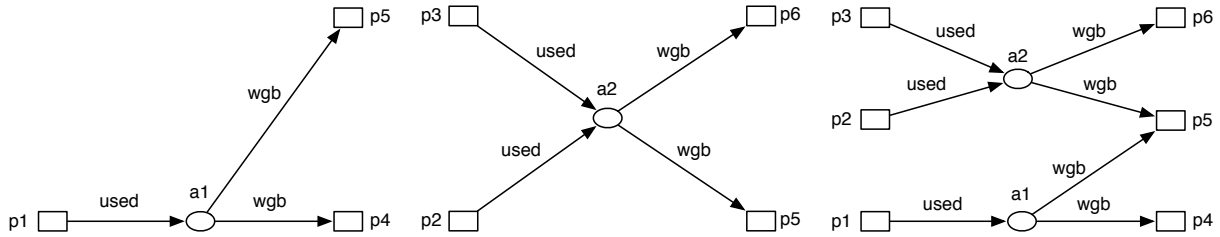
Definition 4. *Let $G^s = \langle V, E\rangle$ be a provenance graph. We say $\mathcal{H} = \{H_1, \ldots, H_n\}$, where H_i is an object of G^s, is a* protection partition *of G^s if $G^s = \bigcup_{i=1}^{n} H_i$, and for all $H_i, H_j \in \mathcal{H}$, one of the following conditions holds: (i) $H_i \cap H_j = \emptyset$, (ii) $H_i \subseteq H_j$, (iii) $H_j \subseteq H_i$.*

In other words, given a provenance graph G^s, a protection partition $\mathcal{H}$ of G^s consists of a set of objects, where every pair of objects has no intersection or one is completely contained in the other. Clearly, $\mathcal{H} = \{G^s\}$ is a trivial set of objects. Note that the set of objects $\mathcal{H}$ is partially ordered by subset inclusion, denoted by $\langle \mathcal{H}, \subseteq \rangle$. Given a provenance graph G^s, there may exist many possible protection partitions of G^s. Ideally, we wish to create the most appropriate protection partition automatically, which depends on the semantics of the underlying system and its authorisation requirements. Undoubtedly, this is a hard problem to solve in a general way, and will be the subject of future work when provenance graphs are represented using RDF. Instead, in the next section we introduce an approach that operates directly on the provenance graphs and allows a provenance owner to define a protection partition at her discretion.

B. Constructing a Protection Partition

Given a provenance graph G^s, we first look at how to identify the smallest objects of G^s that are to be subject to access restrictions. If G is a smallest object, then any graph G' that is a subgraph of G is *not* an object, which implies that a graph obtained by contracting G' from G^s is not valid. For example, in Figure 2, o_1 and o_2 are the two smallest objects of the provenance graph shown in Figure 1b.

We propose an algorithm (Figure 3) that is able to find a smallest object, given a provenance graph G^s and a vertex a of type artifact, denoted by SmiObj(G, a). The vertex a is used to identify those regions of G^s where the object lies. In order to find an induced subgraph of G^s, Algorithm SmiObj



(a) o_1 (b) o_2 (c) o_3

Figure 2. Examples of protection objects for the OPM graph in Figure 1b

Require: A provenance graph G^s, an artifact vertex a
Ensure: An induced subgraph G of G^s
1: initialise an empty queue Q
2: initialise an empty set V and an empty set E
3: insert a in Q
4: **while** Q is not empty **do**
5: remove a from the top of Q into V
6: initialise an empty set A and an empty set A'
7: **for all** $p \in N^i_{G^s}(a) \setminus V$ **do**
8: insert p in V
9: insert pa in E
10: insert $N^o_{G^s}(p) \setminus \{a\}$ in A
11: **end for**
12: **for all** $p \in N^o_{G^s}(a) \setminus V$ **do**
13: insert p in V
14: insert ap in E
15: insert $N^i_{G^s}(p) \setminus \{a\}$ in A'
16: **end for**
17: **if** $A \cap A' \neq \emptyset$ **then**
18: insert all $a \in A \cap A'$ into Q
19: **end if**
20: **end while**
21: **return** $G = \langle V, E \rangle$

Figure 3. SmiObj: Finding a smallest object

works by storing all the vertices that are the out-neighbours and in-neighbours of a and all the associated edges (lines 7-16). The algorithm then repeats the process with a new artifact vertex a' that shares both its out-neighbours and in-neighbours with the last artifact vertex a being examined (lines 17-18), until all such artifact vertices are discovered. This algorithm is essentially a simple modification of a breadth-first search.

Theorem 1. *Given a valid provenance graph G^s and an artifact vertex a in G^s, graph G obtained from the algorithm* SmiObj(G^s, a) *is a smallest object.*

Proof: We first prove that G is an object. Suppose, for contradiction, that G is not an object. Then, by Definition 3, G^s/G must not be valid since G is valid due to being a subgraph of valid G^s. This means that there exists an edge pp' or $p'p$ in G^s/G, where p' is a process vertex into which G is contracted. But this is not true since vertex p has been included in G when SmiObj(G^s, a) executed. Hence G is an object as required.

We now prove that G is a smallest object. Suppose, for contradiction, G is not smallest but is an object as we proved above. Then there must exist a subgraph $G' \subset G$ that is a smallest object. By Definition 3, G/G' is valid since both G and G' are objects. Since $G' \subset G$, if there exists an artifact vertex a in G, which is not in G'. Then there must exist process vertices p and p' in G' such that pa and ap' in G. Hence G/G' results in a cycle $p''a$ and ap'' that contradicts G/G' being valid, where p'' is a vertex that replaces G'. Similarly, we can show G/G' is not valid when G contains a process node p or an edge pa or ap that is not in G'. Thus G' is not an object and so G must be the smallest object. ■

Having introduced the method of computing a smallest object, we are now in a position to explore how to make a provenance owner create a hierarchical structure of objects. We assume that each provenance graph G^s is owned by a user w. Our basic strategy for w to create a protection partition of G^s is to apply the SmiObj algorithm iteratively. It means that w only needs to choose one vertex a from G^s when generating a particular object o. Then w repeats the same process for a contracted graph G^s/o, terminating when the whole graph G^s is reduced to an artifact vertex. The procedure is illustrated in Figure 4 for the provenance graph G^s shown in Figure 1b. The figures shows a sequence of valid provenance graphs, each of which is derived by contracting a smallest object (w defines) in the preceding one.

As a general approach, w should be free to choose an artifact vertex in computing a smallest object in each iteration. However, we propose an approach that seeks to identify as many disjoint smallest objects as possible. For example, in Figure 4a, our approach would suggest w to choose either a_2 or a_5 in generating a smallest object o_2 that does not contain vertex p_{o_1}. This approach provides w with a way to identify different possible domains to which different access control policies may be specified. It is easy to see that each vertex o in Figure 4 essentially represents a smallest object. We can construct a directed rooted tree $T = \langle O, E \rangle$ in which O are these smallest objects and edges E reflect the containment relation between objects. That is, $(o, o') \in E$ if and only if $p_{o'}$ is a vertex in o. For example, Figure 4e shows a rooted

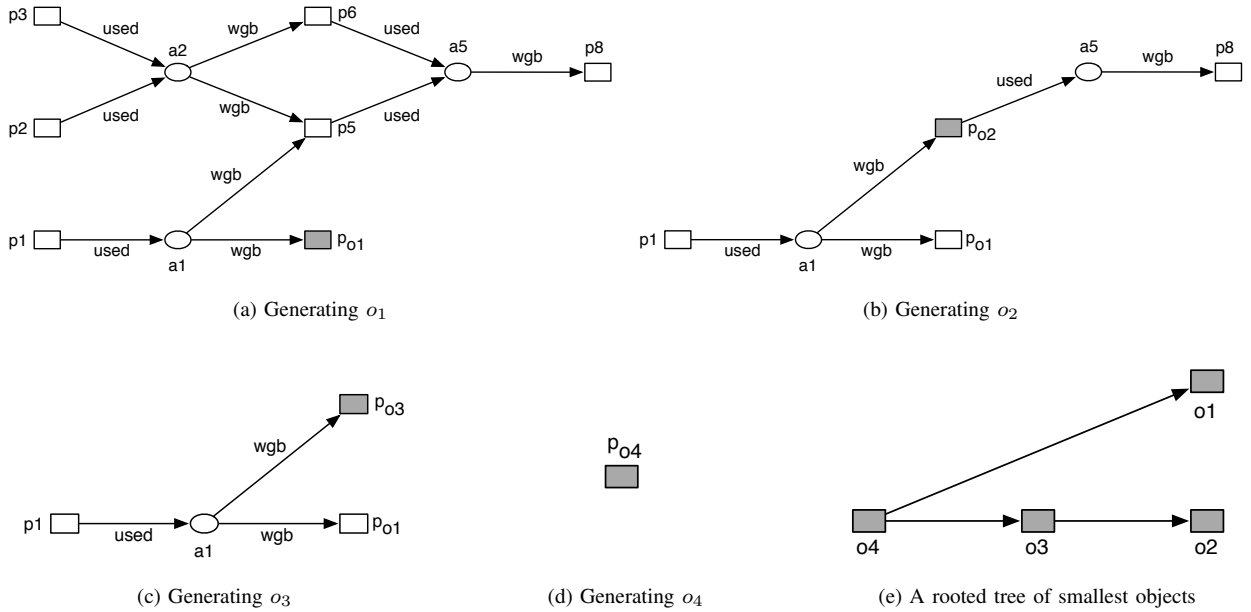


Figure 4. Constructing a hierarchical structure of smallest objects from the provenance graph in Figure 1b

tree of smallest objects that are generated from the procedure shown in Figures 4a–4d.

In order to provide fine-grained access control, our approach makes the owner w at each iteration define an object which is a smallest subgraph satisfying the validity property. One may argue that it would be more suitable for w to define objects of arbitrary size. For example, w may wish to choose an object o_3 in Figure 2c as a smallest unit to which access is restricted, rather than its subgraph o_1 or o_2 in Figure 2. To accommodate this, we can make a slight modification of our SmiObj algorithm to take a set of artifact vertices as input and output an object that contains these vertices. For example, with input of $\{a_1, a_2\}$ chosen by w, algorithm SmiObj may compute o_1 and o_2 based on a_1 and a_2 respectively, and then return o_3 by taking union of o_1 and o_2.

C. Access Control Policies

In this section we explore how to specify and enforce access control policies for the objects we have defined. We assume the existence of a set of roles R. Users are assigned to roles using a binary relation $UR \subseteq U \times R$, where U denotes a set of users. When an owner w defines an object following the procedure described above, we allow w to define a set of roles $R' \subseteq R$ who are authorised to read the object[1]. This essentially gives rise to a labelled rooted tree in which each smallest object o is associated with an access control list of the form $o \leftarrow r_1 \vee \cdots \vee r_n$, where $r_i \in R$. Informally, this list can be interpreted as: any user who is assigned to one of the roles r_1,...,r_n is authorised to read object o.

In order to evaluate access requests in an efficient manner, however, we need to transform these access control lists into role-centric policies, sometimes known as *capability lists*. Let us consider a system in which there are four roles r_1, r_2, r_3, r_4, and owner w defines a rooted tree of objects as shown in Figure 4e with its access control lists enumerated at the left column of Table I. We can see that r_3 is authorised to read two isolated objects o_1 and o_3, and o_3 contains a process vertex p_{o_2} that hides away object o_2 which r_3 is not authorised to read. However, objects o_2 and o_3 which r_1 is authorised to read should not be isolated. Instead, they need to be merged into a single graph by replacing vertex p_{o_2} in o_3 with object o_2. The merging process is essentially the reverse of contracting o_2. We write $o_3; o_2$ to denote the graph that is obtained by this process. Clearly, by Definition 3, $o_3; o_2$ is an object. Our role-centric access control policies can, therefore, be formed in two steps:

- Transform owner-created object-centric lists into role-centric policies each of which has the form of $r \leftarrow o_1 \vee \cdots \vee o_n$, where $r \in R$. It means that role r is authorised for the objects $o_1, \ldots, o_n$.
- For each policy $r \leftarrow o_1 \vee \cdots \vee o_n$, if there exists a path $o_h, o_i, \ldots, o_k$, where $h, i, \ldots, k \in [1, n]$, in the rooted tree, we merge these objects into a single object $o_h; o_i; \ldots; o_k$, and refine the policy with respect to the new object as $r \leftarrow o_1 \vee \cdots \vee o_h; v_i \ldots; o_k \vee \cdots \vee o_n$.

For example, the right column of Table I shows role-centric policies that are derived from the object-centric policies in the left column.

One may argue that our approach creates a false independency by separating two objects $o_4; o_1$ and o_2 for role r_2. Obviously, in this case, we could make them a single object by creating an wgb edge that joins a_1 to p_5 and removing edge (a_1, p_{o_3}, wgb). However, in general, it may not be appropriate to join two objects o and o' together if there is no *direct*

[1]Since we are only concerned with confidentiality of provenance graphs in this paper, we restrict the access mode to read.

Table I
ACCESS CONTROL POLICIES

Object-centric policies	Role-centric policies
$o_1 \leftarrow r_2 \vee r_3 \vee r_4$	$r_1 \leftarrow o_3; o_2$
$o_2 \leftarrow r_1 \vee r_2$	$r_2 \leftarrow o_4; o_1 \vee o_2$
$o_3 \leftarrow r_1 \vee r_3$	$r_3 \leftarrow o_1 \vee o_3$
$o_4 \leftarrow r_2 \vee r_4$	$r_4 \leftarrow o_4; o_1$

dependency between o and o' in the original graph (that is, there does not exist an edge that joins a vertex in o to another vertex in o'). Alternatively, we can dynamically adjust a number of roles available at each iteration based on the roles that have been allocated to objects in preceding iterations. For example, if r_2 was assigned to o_2 but not o_3, then r_2 becomes unavailable for assignment to o_4. While this approach prevents false independencies from arising, it imposes a constraint on the assignment of roles to objects. The investigation of this is an issue for future research.

With role-centric policies in place, checking access requests is straightforward. We employ the notion of *session* defined in RBAC96 [15], but we impose the restriction that only one role to which u is assigned can be activated in a session. This is a well-known practice reflected in most RBAC systems to enforce the least privilege principle [9]. Formally, a user u may activate a role r in a session s if there exists $r \in R$ such that $(u, r) \in UR$. We then model an access request as a pair $\langle r, G^s \rangle$ representing an attempt by user u activating role r to read provenance graph G^s. Given such a request, the access control mechanism would locate the policy $r \leftarrow o_1 \vee \cdots \vee o_n$ and returns a set of objects $\{o_1, \ldots, o_n\}$ that u is authorised to view. Of course, if u requests to read a graph G that is a subgraph of G^s, then the mechanism would return a set of objects O such that for all $o \in O$, $o \subseteq G$

IV. EXTENSION TO A COMPLETE PROVENANCE GRAPH

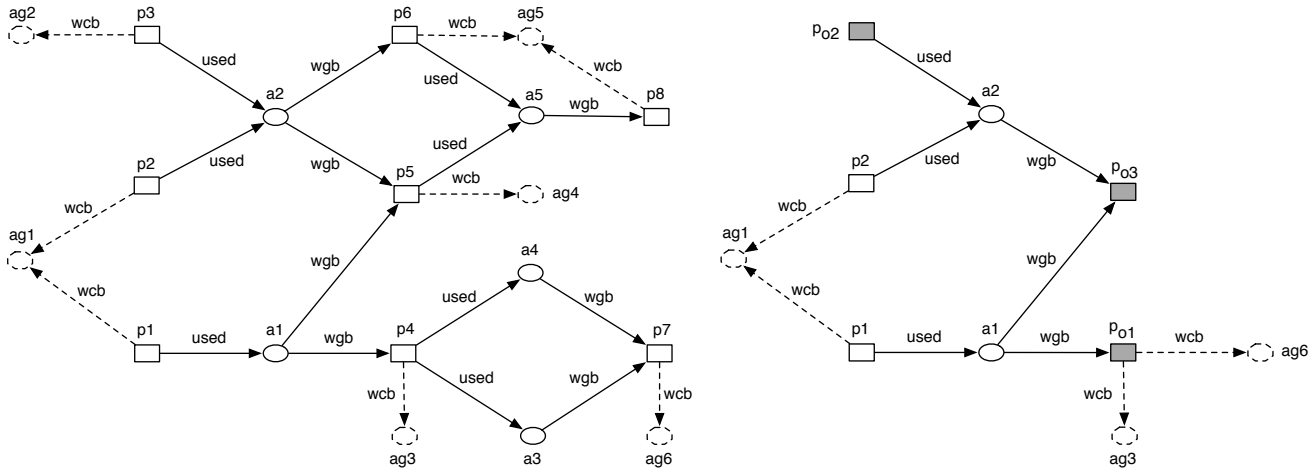
In this section we explore how our approach can be extended to deal with a complete provenance graph, particularly focusing on how a provenance owner can define a hierarchical structure of objects. Note that, in a complete provenance graph, an agent is only associated with vertices of type process via the wcb relations. This indicates that a simple provenance graph can be viewed as an inner layer of the complete one. As an illustration, Figure 5a shows a complete provenance graph in which the simple graph is marked with solid lines, whereas agents and wcb relations are shown as dotted lines.

There are three forms of object we may define in a complete provenance graph G^c. We may define an object to be an edge $e = (p, ag, wcb)$, because contracting e into a process vertex p' results in a valid graph G^c/e. Secondly, as we did in the previous section, an object can be a subgraph of its inner simple graph. A graph obtained by contracting such a subgraph in G^c remains valid. The third form of object is a kind of combination of these consisting of all three types of entities and relations of OPM. Again, this form of object is required to be contracted into a process vertex in order to make sure that the resulting graph is valid. Our aim here is to enable owner w, at each time, to define an object that can be in any of the three forms, in order to support fine-grained access control. As an illustration, w may have a requirement that user u who can read agent ag vertex should not be able to view artifact vertices that were generated with ag involved. This requirements leads w to define two objects of two different forms: one as edge (p, ag, wcb), and the other one as a simple graph that contains the artifact vertices.

Require: A provenance graph G^c, an artifact vertex a, a set of agent vertices Ag
Ensure: An induced subgraph G of G^c
1: initialise an empty set V and an empty set E
2: **if** $Ag = \emptyset$ **and** $a \neq$ null **then**
3: **return** $G = \mathsf{SmiObj}(G^c, a)$
4: **else if** Ag $\neq \emptyset$ **and** $a =$ null **then**
5: **for all** $ag \in Ag$ **do**
6: **if** $N^i_{G^c}(ag)$ is a singleton $\{p\}$ **then**
7: insert ag in V
8: insert pag in E
9: **end if**
10: **end for**
11: **return** $G = \langle V, E \rangle$
12: **else if** Ag $\neq \emptyset$ **and** $a \neq$ null **then**
13: let $G' = \langle V', E' \rangle = \mathsf{SmiObj}(G^c, a)$
14: initialise a set $Q = \{v \in V' \mid \tau(v) = \mathsf{process}\}$
15: **for all** $ag \in Ag$ **do**
16: **if** $N^i_{G^c}(ag) \subseteq Q$ **then**
17: add ag in V
18: add pag in E for all $p \in N^i_{G^c}(ag)$
19: **end if**
20: **end for**
21: **return** $G = \langle V \cup V', E \cup E' \rangle$
22: **else**
23: **return** "invalid input!"
24: **end if**

Figure 6. Obj: Defining an object

We propose an algorithm Obj (shown in Figure 6) to compute a smallest object for each of the three forms. Specifically, the particular form of object being generated depends on what kind of input owner w gives to the Obj algorithm. If Obj takes input G^c and artifact vertex a, then it returns a graph that is computed by invoking the SmiObj algorithm (lines 2-3). For example, in Figure 5b, process vertex p_{o_1} abstracts object o_1 (from G^c in Figure 5a) that is output by SmiObj with input a_3 or a_4. If w provides G^c and agents Ag as input, Obj returns a graph that only includes those $ag \in Ag$ which have wcb relation with only *one* process vertex p, and its associated edge (p, ag, wcb) (lines 4-11). This is because a graph G that contains edges having a common end-vertex ag may not be an object. Consider the example of Figure 5a, if u selects $\{ag_2, ag_5\}$ as input to Obj, then it returns an object o_2 (illustrated as p_{o_2} in Figure 5b) that only includes edge



(a) G^c (b) $G^{c\prime}$

Figure 5. Complete provenance graphs

(p_3, ag_2, wcb), since ag_5 has two in-neighbours p_6 and p_8. When Obj encounters input for all the parameters: G^c, a and Ag, it first constructs a graph G' based on SmiObj(G^c, a), and then checks whether any of its input $ag \in Ag$ has in-neighbours who are not in G'. If this is the case, such an ag and its associated edges are not included in the graph being returned (lines 12-21). Again, take the example of G^c in Figure 5a, if Obj runs on input a_5 and $\{ag_4, ag_5\}$, then it outputs an object o_3 that is contracted into process vertex p_{o_3} in Figure 5b. However, if w inputs a_1 and $\{ag_1\}$, then the graph generated by Obj will not include vertex ag_1 and edge (p_1, ag_1, wcb), otherwise the graph is not an object, as contracting the graph into process vertex p_o in G^c results in an invalid edge (p_2, p_o, wcb).

Note that the Obj algorithm reuses SmiObj to ensure that each object computed is the smallest at its own form. This suggests that owner w is capable of partitioning the complete graph into very fine-grained objects by using our approach. Similarly, we allows w to assign different roles to each object in order to form role-centric policies for evaluating access requests.

V. Related Work

Provenance sanitization that refers to the problem of controlling access to provenance graphs while preserving validity constraints is a relevantly new topic, sometimes under other names such as provenance abstraction [12] or provenance redaction [3]. Our work takes inspiration from a survey paper by Cheney and Perera [5], which examines seven approaches to provenance sanitization, and points out shortcomings of these approaches. We will now examine two existing approaches that are closest to ours, and discuss the merits of our approach. For a full review we refer the interested reader to the survey paper.

Missier *et al.* [12] propose provenance abstraction policies based on the simple security property of the Bell-LaPadula model [2], which requires that every vertex of a provenance graph is associated with a sensitivity value, and every user is assigned a clearance level. Hence, when a user requests to view the provenance graph, those vertices V of the graph whose sensitivity values are higher than the clearance level of the user need to be abstracted. The authors introduced a graph transformation operator Group which takes the original graph and vertices V as input, and output a modified graph that the user is authorised to view. In order to preserve validity, the Group operator often abstracts away (in addition to V) some parts of graph that should be visible to the requester according to the security property. In other words, the policy model they proposed is not compatible with the mechanism for performing abstractions over provenance graphs. In contrast, we provide a coherent model for addressing the problem of provenance abstraction.

The work of Cadenhead *et al.* [3] is also based on graph writing techniques to define provenance redaction policies. Specifically, regular expressions are used to identify a sensitive region G of a provenance graph that needs to be transformed according to redaction policies. A redaction policy consists of two components: a production rule of the form $r : L \rightarrow R$, where L is a subgraph of G that will be replaced by R, and an embedding mechanism that specifies how the rest of graph $G \setminus L$ should be connected to R. This approach is not without its problems. Firstly, there is no guarantee that the transformed graphs would always preserve validity constraints with respect to OPM. Moreover, as the authors themselves point out, there inevitably exist conflicting redaction policies in their model, but it is not clear how these conflicts are to be detected and resolved. In contrast, our approach identifies a hierarchical structure of provenance objects that not only provides fine-grained access control and formal guarantees of validity, but also yields conflict-free policies to be enforced.

In addition to the work on provenance sanitization, there is a significant literature on the relationship between security and

provenance. Martin *et al.* [10] discuss how provenance can be regarded as a security measure, playing a role in the protection of data integrity and confidentiality. Along this line, Park *et al.* [14] propose a provenance-based access control model in which authorisation decisions are made based on the analysis of provenance pertaining to entities appearing in an access request. Cheney [4] offers a formal model for provenance in which security properties such as disclosure and obfuscation are defined, and the complexity of enforcing these policies are explored.

Our idea is to partition a provenance graph into a hierarchical structure (a rooted tree) of protection objects, which is essentially the structure of XML documents. This means that existing work on access control models and enforcement mechanisms for XML documents may be applied into our context. For example, we require additional resources to store and maintain hierarchical objects in order to efficiently enforce access control policies associated with these objects. The work of using cryptographic techniques [1], [6] and the dynamic view management scheme [16] to enforce policies for XML documents is complementary to ours in this regard.

VI. Conclusion

In this paper we have introduced a new owner-based access control model for protecting access to provenance graphs. To our knowledge, this is the first model to define protected objects within a OPM-style provenance graph by taking into account validity constraints of the graph. We proposed an algorithm that enables a provenance owner to define a set of objects that forms a nested partition of a simple provenance graph, and to specify role-based policies for restricting access to each of these objects. We then extended our approach in a natural way to a complete graph that includes all the core components of OPM. In particular, we exploited the unidirectional causality relationship between agent and process vertices to enable the owner to divide the complete graph into very fine-grained objects. We believe that our approach provides a simple and lightweight access control mechanism for provenance graphs, and also inter-operates well with other hierarchical access control frameworks.

Nevertheless, a considerable amount of research needs to be done. Of immediate interest is the development of a prototype implementation of our authorisation model for OPM, and assessment of feasibility of our approach in a food traceability project in which we are involved. On a more theoretical level, we intend to undertake a more thorough investigation of our authorisation model by considering the semantics of provenance graphs. In particular, it would be interesting to assign meanings to those process vertices that represent contracted subgraphs in such a way that viewers of the provenance graph cannot determine those regions of the graph for which she is not authorised. This is relevant to *the inference problem* raised by Thuraisingham *et al.* [17] in the context of semantic web and provenance. An inference problem occurs if someone can infer some provenance information based upon other information she can view along with her background knowledge. We would like to investigate how to alleviate the possibility of such inferences by incorporating an inference controller into our authorisation model.

Acknowledgements

This research is supported by the award made by the RCUK Digital Economy and Energy programmes to the TRUMP UK-India project (award reference: EP/J00068X/1) and the RCUK Digital Economy programme to the dot.rural Digital Economy Hub (award reference: EP/G066051/1).

References

[1] M. Abadi and B. Warinschi, "Security analysis of cryptographically controlled access to XML documents," *Journal of the ACM*, vol. 55, no. 2, pp. 6:1–6:29, 2008.

[2] D. E. Bell and L. LaPadula, "Secure computer systems: Unified exposition and Multics interpretation," Mitre Corporation, Tech. Rep. MTR-2997, 1976.

[3] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham, "Transforming provenance using redaction," in *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, 2011, pp. 93–102.

[4] J. Cheney, "A formal framework for provenance security," in *Proceedings of the 24th IEEE Computer Security Foundations Symposium*, 2011, pp. 281–293.

[5] J. Cheney and R. Perera, "An analytical survey of provenance sanitization," in *Proceedings of the 5th International Provenance and Annotation Workshop*, 2014, pp. 113–126.

[6] J. Crampton, "Applying hierarchical and role-based access control to XML documents," *Computer Systems: Science & Engineering*, vol. 21, no. 5, 2006.

[7] S. C. Dey, D. Zinn, and B. Ludäscher, "Propub: Towards a declarative approach for publishing customized, policy-aware provenance," in *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, 2011, pp. 225–243.

[8] 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*, 2007, pp. 13–18.

[9] N. Li, J.-W. Byun, and E. Bertino, "A critique of the ANSI standard on role based access control," *IEEE Security & Privacy*, vol. 5, no. 6, pp. 41–49, 2007.

[10] A. Martin, J. Lyle, and C. Namiluko, "Provenance as a security control," in *Proceedings of the 4th Workshop on the Theory and Practice of Provenance*, 2012.

[11] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 773–776.

[12] P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. Danger, "ProvAbs: Model, policy, and tooling for abstracting PROV graphs," in *Proceedings of the 5th International Provenance and Annotation Workshop*, 2014, pp. 3–15.

[13] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. T. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. G. Stephan, and J. Van den Bussche, "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[14] J. Park, D. Nguyen, and R. S. Sandhu, "A provenance-based access control model," in *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust*, 2012, pp. 137–144.

[15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[16] M. Thimma, T. K. Tsui, and B. Luo, "HyXAC: a hybrid approach for XML access control," in *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, 2013, pp. 113–124.

[17] B. Thuraisingham, T. Cadenhead, M. Kantarcioglu, and V. Khadilkar, *Secure Data Provenance and Inference Control with Semantic Web*. Auerbach Publications, 2014.