# Scalable Access Controls for Lineage

Arnon Rosenthal, Len Seligman, Adriane Chapman, Barbara Blaustein
*The MITRE Corporation*
*Bedford MA, McLean VA*
*{arnie, seligman, achapman, bblaustein}@mitre.org*

## Abstract

Lineage stores often contain sensitive information that needs protection from unauthorized access. We build on prior work for security and privacy of lineage information, focusing on complex conditions and scalable administration. We use Attribute-Based Access Control (ABAC) to express conditions based on many *attributes*, instead of roles. We then make administration and management more scalable, instead of managing large, monolithic access predicates for each object. To do so, we first support modular traceability and maintainability for separate concerns (e.g. security, legally mandated privacy, organizationally mandated privacy). We then provide constructs to manage authority when multiple administrators must collaborate. We show that these security techniques are needed for easy lineage security administration.

## 1. Introduction

Several papers have noted that lineage (also known as provenance) information may often contain sensitive information that must be protected, e.g. [8, 18, 29], and a few have described access control mechanisms appropriate for lineage data [7, 8, 12]. This paper focuses on managing the access policies on nodes, edges, and properties of a lineage graph. We extend the prior work with these contributions:

- We allow finer-grained policies—i.e. for particular properties of an individual lineage node or edge—and illustrate their importance. We also categorize properties in a way that helps assign administrators for parts of a policy.
- We base our model on attribute-based access control (ABAC). Unlike role-based access control (RBAC), the predominant model in prior lineage security work, ABAC can express general access predicates, referencing any available attribute information in the environment.
- We enhance modularity of ABAC by adding a model for separate capture and combination of multiple concerns. The explicit decomposition of access predicates makes them easier to understand, maintain, and trace to specific concerns. We illustrate its applicability to lineage security and describe key stakeholder roles for lineage scenarios drawn from enterprise applications.
- We provide a way to manage split authority, where different concerns (or *stakes*) are managed by different people (*stakeholders*).

Following the conventions of the Open Provenance Model (OPM) [22], data (artifacts) and processes are represented by nodes; each edge represents a relationship (e.g. generatedBy). Edges represent causality and point *inverse* to data flow; they may also be labeled with specific roles, e.g. input_arg_X.

**Example:** Consider the lineage graph in Figure 1, which shows the data (ovals) and processes (rectangles) used to produce an Emergency Preparedness Office's (EPO) Epidemic Warning Report (dashed outline). Mary, a Health Department recipient of an epidemic warning report wants to know how it was produced in order to know how to best interpret it, whether to trust it for her purposes, etc.

However, in determining which parts of the lineage graph to reveal to Mary, the lineage system should consider several stakeholders' interests. For example, the provider of animal test data may want funding agencies to know he contributed to the intelligence report, but may not want the public to know. This same investigator demands that high-level approvals be required to release the edge with role "Animal_Tests". Additionally, the properties of a lineage node may contain even more sensitive information. For instance, while most cleared analysts can see the BioThreat Intelligence report node, the authoring agent's identity should be protected by only releasing this data to a very restricted group of users.

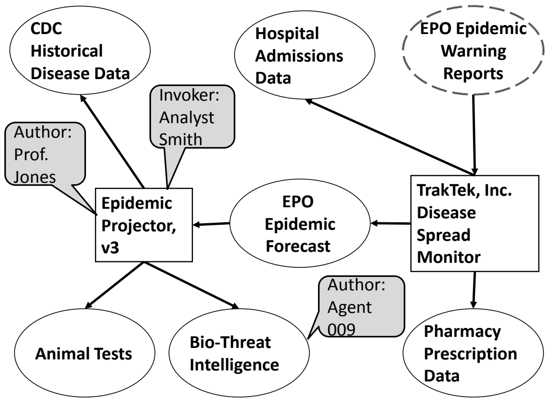


**Figure 1: Lineage Graph for EPO Epidemic Warning Reports.**

The requirement to protect specific properties of a node, such as the author of a report or invoker of an execution, illustrates the need for fine-grained access controls; treating a given node or edge as a monolith is often inadequate. There may also be conflicts among interested parties about how restricted the lineage information should be. For example, the author of the Epidemic Projector, Prof. Jones, may claim that the information about the algorithm should be visible to anyone, while Analyst Smith, the invoker of the program, wishes this particular use to be considered extremely sensitive. Thus, any security model must allow both Analyst Smith and Professor Jones to express their concerns, and determine how best to honor them. A good model will let each of them reexamine and edit their concerns, and regenerate the access predicates.

Unfortunately, current access control mechanisms are too hard to administer where there are multiple stakeholder concerns about a single object. An administrator must consider all the relevant stakeholders' concerns and define the complicated policies that combine them. Importantly, the separate concerns are not currently modeled. The resulting composite policies are not modular; they lack traceability; they're difficult to understand and edit, and they're not well-suited to gap analysis. For example, suppose access to lineage information about Animal Testing depends on the following predicate (the arguments are discussed in Section 3.1):

*Animal_Testing_Access*(user, resource, environment)
$:= [\text{User.Division} = \text{Intelligence} \wedge$
$\quad \text{User.AssignedProject.Type} = \text{Epidemiology} \wedge$
$\quad \text{Request.SourceDomain is in } \{.\text{gov}, .\text{mil}\} \wedge$
$\quad \text{Experiment.ReleaseMarking} = \text{Intel} \wedge$
$\quad (\text{ExperSubject.Type} = \text{inanimate} \vee$
$\quad \text{ExperSubject.Type} = \text{animal} \wedge$
$\qquad \text{experimenterName.pseudonym} = \text{true} \vee$
$\quad \text{ExperSubject.Type} = \text{human} \wedge$
$\qquad \text{releaseOnFile(ExperSubject)}$
$\vee\ [\text{Request.HasApproval.Level} \geq 4 \vee$
$\quad (\text{Request.HasApproval.Level} \geq 2 \wedge$
$\qquad \text{threat.Status} = \text{Red})]$

This predicate exploits a wide variety of knowledge about the request, coming from multiple sources. It is hard to imagine encompassing it all in a role hierarchy. There are several categories of concerns here, such as government secrecy, experimental subject privacy, and experimenter privacy. Even within categories, some stem from agency-wide substantive policy (e.g. do not release outside the agency), others are aware of the sort of information this is (only epidemiologists have access), and some are because an authority has insisted on crisis workarounds (the disjuncts at the end).

In such an environment, a lineage service exists mainly to let users execute queries over the lineage graph, e.g. to find all predecessors and successors of a node, while applying various predicates and projections. *Lineage security* ensures that each query executes on a database subset, i.e. nodes, edges, and property values for which the request satisfies the access control predicates. Previous researchers have described basic capabilities, but have not addressed three more advanced requirements:

First, the access predicate on a node, edge, or property may involve multiple subexpressions, dependent on different attributes of users, the resource, and the environment. Role-based access control is not easily extended to support this.

Second, the predicate may include terms representing many distinct concerns that ought to be managed modularly, such as security, legally mandated privacy, and organizationally mandated privacy. If one of these concerns changes, or if compliance is being audited, we do not want to wade through a 12-line predicate.

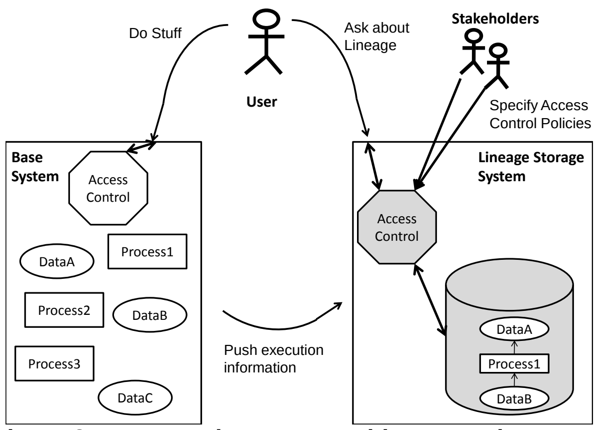


**Figure 2: An example system architecture. Lineage storage and access control are on the right.**

Third, there may be multiple stakeholders involved in setting the policy on a protected item, and it is necessary to govern how the different desires are to be combined. For example, if privacy officers want only doctors to see certain information, and existing workflows require that it be available to financial managers, how does the system help administrators manage these conflicting relationships?

Any access control system for lineage must allow the following: determine access based on values of multiple attributes; handle multiple goals, each with separately evolving tradeoffs and legal requirements; and appropriately combine the all stakeholders' concerns.

One obvious but inadequate solution is to appoint a lineage system administrator. However, the stakeholders whose systems supply the lineage data may not be willing to give this person discretion to make substantive policy decisions. In our experience, the lineage service is often a political lightweight, not an 800-pound gorilla. Also, the appointment of an administrator does not make the problems of stakeholder conflicts and concern management go away, nor does it give the administrator guidance on how to resolve them.

When faced with complex expressions, current access control mechanisms lack *traceability* and *maintainability*, i.e. they do not connect clauses in the predicate to the concern that motivated them, nor do they help an administrator focus only on the relevant portion when editing predicates. For example, the system should be able to show which clauses in the predicate exist to protect patient privacy vs. experimenter protection vs. national security. If a privacy regulation changes, we want a capability to edit just the relevant sub-specifications. When the HIPAA[1] auditors arrive, we want to highlight the controls motivated by patient privacy. If HIPAA rules change, we want to edit this portion without needing to extract it from a dozen other clauses.

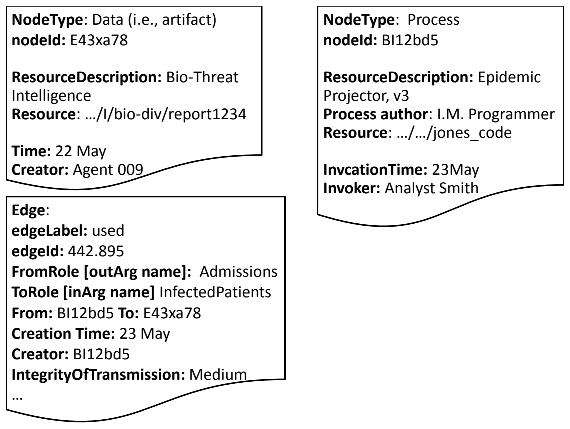
We lay out our system model in Section 2. In Section 3, we begin by showing how a general purpose attribute-based access control capability can support fine-grained access control for lineage data. We then extend vanilla ABAC to provide better maintainability, traceability, and sharing of authority, and show how this fits the needs of lineage. Section 4 discusses related work, and Section 5 describes our ongoing efforts to implement these ideas and areas for future research.

## 2. High-Level Architecture

As users go about their ordinary tasks, creating or manipulating data on base systems, information about their actions is reported to the lineage store in ways that minimize intrusion on the *base* (i.e. application) systems [10, 15, 17, 25]. Figure 2 shows that the lineage storage, querying and access controls are separate from these base systems. Access controls on base systems are unaffected; the lineage store controls access to lineage information. The lineage store is logically unified but may be physically distributed.

A lineage graph describes a series of process invocations, executed by one or more users, in pursuit of their various goals. We follow ES3 [17] and PASS [24] in that generation is *ad hoc* – covering whatever was reported to the lineage system. Unlike traditional workflows (discussed in Section 3.4), the tasks, or steps, need not be defined prior to execution, so graphs may grow indefinitely and in unpredictable ways. In fact, information is tied together by data usage (i.e. graph connectivity), rather than by pre-defined patterns.

[1] Privacy regulations imposed by the U.S. Health Insurance Portability and Accountability Act of 1996



**NodeType**: Data (i.e., artifact)
**nodeId:** E43xa78

**ResourceDescription:** Bio-Threat Intelligence
**Resource**: …/I/bio-div/report1234

**Time:** 22 May
**Creator:** Agent 009

**NodeType**: Process
**nodeId:** BI12bd5

**ResourceDescription:** Epidemic Projector, v3
**Process author**: I.M. Programmer
**Resource**: …/…/jones_code

**InvcationTime:** 23May
**Invoker:** Analyst Smith

**Edge**:
**edgeLabel:** used
**edgeId:** 442.895
**FromRole [outArg name]:** Admissions
**ToRole [inArg name]** InfectedPatients
**From:** BI12bd5 **To:** E43xa78
**Creation Time:** 23 May
**Creator:** BI12bd5
**IntegrityOfTransmission:** Medium
…

**Figure 3: Properties and values for some nodes from Fig. 1.**

A *lineage graph* consists of a set of nodes, *N*, and a set of edges, *E*. As in [8], each object (node or edge) has descriptive properties that optionally appear on each node, as illustrated in Figure 3. A lineage manager should predefine and categorize a starter set of properties, such as resource description, invocation time, etc.

A *lineage query* consists of edge traversal forward or backward from a start node (or node set), applying access predicates to node properties to determine which property values should be returned.

Our security model protects objects (nodes and edges) and the values of their properties. These are called *protected items.* There is an *access predicate* that controls the visibility of each protected object and property.[2] Edges which are not visible to a particular user are not traversed in executing lineage queries for that user. Thus, if the lineage graph contains three nodes that the user is entitled to see, but he is not entitled to see the edges between them, a query on that graph will return only the single start node. To aid security administration, we categorize properties into buckets that help determine default stakeholders.

- *Description of an entity in the base system.* Examples include data description, data location, process description, etc. These items will be in the lineage graph only if the base system reports them to the lineage system. For instance, in Figure 3, the process node includes the description "Epidemic Projector, v3".
- *Link to the underlying information.* Base systems may keep more extensive information under their control, providing the lineage system only with a link, as in the Resource property of the nodes in Figure 3**.**
- *Description of a process invocation or creation of a dataset.* For instance, the time started, time ended, who invoked it, whether data integrity was protected, etc. In the right-hand node record of Figure 3, we can see that Analyst Smith invoked the process.

This list is not intended to be exhaustive, but it gives lineage system security administrators an initial set of node and edge properties to address in formulating access policies.

## 3. Access Control Extensions for Lineage

We now discuss desirable extensions to prior work on access controls for lineage data: attribute-based access controls, modularization of concerns, and sharing of authority.

Section 3.1 shows the advantages of applying ABAC in lineage security. The next two subsections propose general purpose enhancements to ABAC formalisms and administrative processes, extensions that are particularly useful in settings such as lineage. In Section 3.2, we show how to handle with multiple concerns, while in Section 3.3 we deal with multiple stakeholders. Section 3.4 shows opportunities to build in lineage-specific definitions and defaults, without which the administrative burden would be prohibitive.

### 3.1. Moving Toward Attribute-based Access Control

In this section, we argue that ABAC, not RBAC, is the right basis for lineage security research, and give an overview of ABAC. Later sections exploit the flexibility of ABAC to propose additional capabilities.

Prior lineage security proposals used role-based access control (RBAC) [8, 12]. Unfortunately, RBAC is known to suffer from serious scalability problems: As "policy becomes finer-grained and more attributes are involved, one gets a separate role for each

[2] A practical system will encourage use of the same predicate to control multiple items.

combination of attribute values, making the user-to-role assignment and permission-to-role assignment tasks prohibitively expensive" [31]. If a new resource needs new policies, it needs a new set of roles, and users need to be provisioned into these. For example, a project management system may already know that Joe has joined a project team, that Mary is an MD, or that Task6 has been completed and roles in it are no longer valid, but this knowledge must be separately expressed and kept current as role assignments [26]. Finally, RBAC does not permit predicates that address resource or environment attributes, e.g. that a medical report is from Psychiatry, or that a request was submitted at 3AM.

Attribute-based access control (ABAC) provides a more scalable alternative that satisfies these objections. Each relevant factor (e.g. project assignment, threat severity) is an attribute, which can be independently managed[3]. A predicate can reference as many different attributes as needed. Typically, attributes represent uncontroversial factual statements asserted by a trusted source, e.g. the current date, or that Analyst Smith's assignment is BioDesk. One can also define computed attributes, e.g. $att_{new} \cong (att_1 \vee att_2) \wedge (att_3 > 2)$.

For each protected item, an administrator defines an *access predicate*, which is formally a derived, unnamed attribute. The evaluator, or "policy decision point", obtains attribute-value pairs for this request from an attribute service.[4] If the predicate returns True, access is permitted.

The approach has several advantages. Since each attribute is managed independently, one supplies a linear amount of information (sum, not product, of the attribute extents). It is easy to incorporate multiple clauses. To reference a wider set of information, the predicate can reach into the existing information system.

Administrators and system owners collaborate to manage the attribute set. Each attribute has a unique name, e.g. a URI. The unavoidable decentralization gives rise to the usual problems of semantic heterogeneity, and of motivating data providers across organizations [27]. Administrators may define new attributes, and as in other flexible large scale data environments, they are allowed to invent or import attribute names in their own name spaces.

Standards and commercial ABAC implementations are maturing, and we have personally observed several large government organizations exploring ABAC. They particularly like the fact that one can add new users simply by making their attributes available -- there is no need to "provision" them (i.e. to insert users into each relevant role). Additionally, new objects can be protected by defining predicates over existing attributes. XACML [2] is a standard language for passing predicates to enforcers. The SAML standard [1] allows assertions about attribute values and the trust in them to be passed around a distributed system.

## 3.2. ABAC and Modular Concerns

ABAC has attracted wide interest in the security community. We enhance it here by tying it to explicit concerns. Even when all the concerns on an item are managed by a single person, the modularity and explicit links make it easier to edit or audit.

We propose an extension of ABAC with explicit support for modular capture of concerns, i.e. named requirements that are linked to expression fragments that may be put into the access predicate. The administrator can delegate to other stakeholders the ability to define new attributes as well as access predicates. The administrator then writes (or, preferably, selects from a pre-defined library) a *combiner* predicate. The most common combiners are likely to be *conjunction*, which gives all authorized stakeholders a veto, *weighted voting*, and *disjunctions* representing alternate scenarios, as illustrated in the *Animal_Testing_Access* predicate, but administrators may develop their own as needed. The advantage of this approach is increased modularity, traceability, and (optionally) delegation.

Each concern (e.g. to protect privacy or proprietary information such as the Epidemic Projector algorithm) can be assigned an attribute. When change is needed, the effort is modular: only one attribute requires inspection. Traceability is easy.

[3] For compatibility with existing systems, a role can be treated as an attribute.

[4] Attribute infrastructures typically provide a service interface through which evaluators can request attributes they need. Behind the scenes, attributes may come from the request message, directories, services, and databases.

While lineage access controls motivated us to add modular capture of stakeholder concerns to ABAC, we note that these capabilities have much broader applicability. For example, before a billing record can be released, a hospital must examine proprietary pricing concerns plus both the patient's and the doctor's privacy concerns. Similarly, before a research study can be released, one needs to see whether patients have given permission (or been de-identified), whether the researcher is willing to reveal his data to potential competitors, and whether the funding agency is satisfied.

### 3.3. ABAC and Sharing the Power

The example access predicate in Section 1 had contributions covering many different concerns, from many different people. One needs an authority structure to determine who can say how those contributions are to be put together, and then a process for those with proper authority to specify each needed combiner.

Complex predicates like the one shown can arise in any arena, not just lineage. However, lineage is particularly prone to complex authority structures, because it often tracks information passing through many organizations, utilizing data and processes derived from disparate individuals and entities, and because a graph edge often connects independently owned processes. Other e-science stakeholders who may want a voice include suppliers of workflow templates, scientific funding agencies, and oversight agencies.

Thus, it is rarely acceptable to appoint a lineage system administrator and give her full authority to decide what access controls to specify. Rather, the access predicate should reflect the different stakeholders' contributions, combined in a way specified by higher level stakeholders. Furthermore, when a stakeholder wants to change what she has specified, she should be able to change it directly, rather than sending an email to an administrator requesting a change.

We outline here how to build on an ABAC system to achieve this flexibility, providing constructs and processes. The delegation operation is ordinary; the novelty is the process for using it. We begin with an overview, and then give an explicit algorithm.

For uniformity, the access predicate for each protected item is treated as an attribute, and each attribute has a single authority. Normally the authority is an administrator; occasionally (e.g. for edges) it is the lineage system itself; for attributes that are statements of fact, it is an external source (e.g. an enterprise directory).

The system helps attribute owners establish shared responsibility for their attribute, beginning with the top level access predicate and recursing downward. The owner of attribute A specifies a derivation for the value of A, by one of the following methods:

- The owner provides a direct means (for example, one might look up a value in a database, or check membership in a traditional access control list), OR ELSE
- The owner provides an expression tree to derive the attribute value:
    - Pick a combiner function for the root. (One can create and register a new one as part of this step.)
    - Bind some of the combiner's input arguments to attributes that already exist.
    - For each input argument that is not yet bound,
        - Define a new attribute, with a unique URI (the definer is its owner, and must describe its meaning in text and bind it to input argument(s)), OR
        - Optionally delegate ownership to somebody else.
- Recursively, derive the value of newly created attributes, until it grounds in the first bulleted step.

Delegation enables the work to be shared and to be done by the proper stakeholder (especially when different organizations are involved).

### 3.4. Lineage Security over our ABAC Framework

As described above, fine-grained ABAC, complemented by tools to manage multiple stakeholder concerns, offers many benefits for lineage security. While the basic techniques apply to diverse applications (beyond lineage), we now consider special properties of lineage information that we can exploit to simplify access control policy administration. Ease of administration is critical; if administrators are forced to wade through long lists of attributes and stakeholders, the access control system is unusable. Therefore, we seek system defaults that predefine an initial set of protection concerns and their stakeholders. Additionally, pre-defined workflows present an opportunity for further reducing the security administration burden, and side agreements provide a convenient workaround for capabilities not built in.

For ease of administration, different types of objects within a lineage graph should have pre-defined default stakeholders. For example, for *process nodes*, the default stakeholders are the author (e.g., code developer) and the invoker of a specific execution. Meanwhile, for *data nodes*, we must distinguish between external data and data generated by a process known to the lineage system. While stakeholder concerns (if any) must be explicitly asserted for external data, for other data the default stakeholders are those defined for the process that produced the data. Most often, the default will be the invoker of the process. Finally, for edges, the stakeholders are the union of the stakeholders of the source and destination nodes, although the concerns of stakeholders for each node are combined separately. The combined predicate from source node stakeholders represents what they are willing to reveal about the edge, as does the combined predicate of destination node stakeholders. The concerns of both nodes' stakeholders are then reconciled to determine whether to reveal the edge. The default is a veto – access to the edge is only granted if both sets of stakeholders allow it.

Up to this point, we have discussed ad hoc executions. However, a large body of lineage research concerns routine computations in which a pre-defined workflow is run many times [10, 23, 25]. The lineage system can automatically generate much of the information needed to describe the resulting history to the lineage store. The workflow creator is allowed to express access control specifications on nodes or properties, and is automatically made an additional stakeholder whose concerns will be combined with those of others. Also, if a (process or data) object is to be used in many instances of the workflow, the item's policy can be propagated automatically into the instantiations. Much of this was done in [12]. We suggest going farther by propagating access predicates on the properties, and coping with mixed histories (partly *ad hoc* and partly derived from workflows).

To keep the lineage system simple, we did not attempt to provide all possible constructs, nor to define all possible classes of Concern or Stakeholder. One omission was sticky policies (a sort of mandatory access control that goes from a node to all nodes downstream, requiring that the certain clauses be attached to policies on the downstream objects). For example a GNU public license requires that derived products be freely shared, if deployed as a single package. Additionally, we did not automatically build in a veto right for certain stakeholders, e.g. a company which may hold proprietary rights to all data produced by its employees, or a funding agency which may insist that all grantees reveal how they produced their results.

We know of no ABAC constructs for this. Such situations can be handled by side agreements among administrators, e.g. agreeing to insert a veto or to extend access to organizations that maintain a non-disclosure agreement. In addition to keeping the lineage system of manageable size, there is another reason not to build constructs for all these cases into the lineage model. Many side agreements may need to cover the base system as well as lineage, and there is little chance of extending the models in all relevant base systems.

## 4. Related Work

### 4.1. Lineage Systems

Lineage has become an active research area. Some systems collect lineage generated from executing pre-defined, explicit workflows [10, 15, 23, 25]. Other systems monitor users' ad hoc executions [17, 24]. Lineage within databases [4, 9], mechanisms to help users query and navigate the lineage data [14], and topics such as lineage storage efficiency [11] have also been explored. The Open Provenance Model (OPM) [22] is a high level attempt to model generic lineage graphs and their component artifacts, processes and edges.

As lineage systems gain traction, they will be used with sensitive data and processes, e.g. medical data [3, 28]. The need for secure lineage is outlined in [6, 8, 18, 29]. The current body of research can be broken out into three general categories: securing the underlying lineage information from tamper [32]; enforcing expected behavior [19], and specifying access controls. Several groups have proposed access control models for lineage. In [30], a high level overview of security concerns in a SOA environment are presented. In particular, a basic access control policy that allows users to access lineage nodes if the level(user) ≥ level(node). [7] breaks lineage information into nodes and edges, specifying a RBAC policy for distinct lineage information. In a different take, [13] utilizes information about the past ownership of an object (or lineage) to determine whether or not a data object should be released. Finally, [12] uses role-based access control, and explores how to release lineage views based on this model. Our work is the first to explore application of fine-grained ABAC to lineage information.

### 4.2. Security models

Access control lists simply designate which users may access each object. Role-based access control (RBAC) offers a major improvement over access control lists and is used in many systems. Two simple forms are specified by the NIST [16] and SQL standards, and in many application server and DBMS products. Dozens of extensions have been proposed, e.g. GTRBAC [5], but they have gained little industrial traction. In general, RBAC suffers from scalability problems since security administrators must maintain their own up-to-date model of resources and users [26]. SQL and XML security models both handle structured data. SQL protections (role based controls to table, column, or row) are insufficient to provide a convenient basis for securing lineage information; cell-granularity extensions are immature [20]. Furthermore, one cannot easily express general predicates directly (and use of a separate view for each user profile is too awkward). Also, there is no construct for two administrators to share control over a policy. XML security is less mature and less standardized than SQL security. It is also more complex, since it addresses nesting and paths in trees, e.g. to authorize a node to be bypassed so its children remain visible [21].

## 5. Conclusions

Among our users, we have encountered diverse needs for securing lineage information. Our approach emphasizes general purpose constructs for both lineage and security models, thereby giving vendors more incentive to build, and users more incentive to learn. At the same time, the general constructs are well suited to extension, customization, and traceability.

We have suggested that the lineage community move to attribute-based access controls, which are more flexible than roles in situations where an access predicate tests multiple kinds of information. We also saw that predicate expressions were a convenient way to combine multiple concerns into a decision rule.

We believe that concerns need to be managed explicitly, so their associated predicates can be separately explained, updated, validated, and audited. From here, it is a small step to managing the split of concerns and delegation of parts of it. We propose defaults that are suited to lineage systems.

We are implementing these access control techniques within the PLUS system [6]. For administration and also run-time efficiency, our prototype lets an administrator or stakeholder choose a set of items and attach the same predicate to all of them. We have identified a set of default concerns and their stakeholders; administrators may add additional ones. At the time of this writing, we use simple conjunctions – vetoes – for all combinations of concerns. Meanwhile, we demonstrate our prototype frequently to potential users of lineage information, and are gathering requirements and reactions.

Based on this information, we are investigating "surrogate" answers – a general facility that, when a user is unable to access some of the desired information, gives approximate or other answers that can help. Additionally, we plan to exploit the modular specification of access control predicates to explain authorization failures—e.g. the concerns about privacy of patient medical information were satisfied, but not those pertaining to financial information.

## References

[1] "SAML," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[2] "XACML," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[3] E. W. Anderson, S. P. Callahan, G. T. Y. Chen, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva,

and H. T. Vo, "Visualization in Radiation Oncology: Towards Replacing the Laboratory Notebook," SCI Institute Technical Report, No. UUSCI-2006-17, University of Utah 2006.

[4] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," *VLDB Seoul, Korea*, pp. 953-964, 2006.

[5] R. Bhatti, J. Joshi, E. Bertino, and A. Ghafoor, "X-GTRBAC Admin: A Decentralized Administration Model for Enterprise Wide Access Control," X-GTRBAC Admin: A Decentralized Administration Model for Enterprise Wide Access Control, 2004.

[6] B. T. Blaustein, L. Seligman, M. Morse, M. D. Allen, and A. Rosenthal, "PLUS: Synthesizing privacy, lineage, uncertainty and security," *ICDE Workshops*, pp. 242-245, 2008.

[7] U. Braun and A. Shinnar, "A Security Model for Provenance," in *Technical Report*, vol. TR-04-06: Harvard University Computer Science, 2006.

[8] U. Braun, A. Shinnar, and M. Seltzer, "Securing Provenance," Securing Provenance, 2008.

[9] P. Buneman, J. Cheney, and S. Vansummeren, "On the Expressiveness of Implicit Provenance in Query and Update Languages.," *ICDT*, pp. 209-223, 2007.

[10] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, and C. T. S. H. T. Vo, "VisTrails: Visualization meets Data Management," *SIGMOD*, pp. 745-747, 2006.

[11] A. Chapman, H. V. Jagadish, and P. Ramanan, "Efficient Provenance Storage," *SIGMOD*, pp. 993-1006, 2008.

[12] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang, "Scientific Workflow Provenance Querying with Security Views," *WAIM*, 2008.

[13] A. Cirillo, R. Jagadeesan, C. Pitcher, and J. Riely, "Tapido: Trust and Authorization Via Provenance and Integrity in Distributed Objects," Tapido: Trust and Authorization Via Provenance and Integrity in Distributed Objects, 2008.

[14] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson, "Addressing the provenance challenge using ZOOM," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 497-506, 2008.

[15] S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludascher, T. McPhillips, S. Bowers, and J. Freire, "Provenance in Scientific Workflow Systems," *IEEE Data Engineering Bulletin*, vol. 32, pp. 44-50, 2007.

[16] D. Ferraiolo, R. Kuhn, and R. Chandramouli, *Role Based Access Control*: Artech House, 2004.

[17] J. Frew, D. Metzger, and P. Slaughter, "Automatic capture and reconstruction of computational provenance," *Concurr. Comput. : Pract. Exper.*, vol. 20, pp. 485-496, 2008.

[18] R. Hasan, R. Sion, and M. Winslett, "Introducing Secure Provenance: Problems and Challenges," *Proceedings of the Third International Workshop on Storage Security and Survivability*, 2007.

[19] I. Khan, R. Schroeter, and J. Hunter, "Implementing a Secure Annotation Service," Implementing a Secure Annotation Service, 2006.

[20] K. LeFevre, R. Agarwal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. J. DeWitt, "Limiting Disclosure in Hippocratic Databases," Limiting Disclosure in Hippocratic Databases, 2004.

[21] M. Makoto, T. Akihiko, K. Michiharu, and H. Satoshi, "XML access control using static analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 9, pp. 292-324, 2006.

[22] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, "The Open Provenance Model," University of Southampton 2007.

[23] L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. C. JR., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, and Y. Zhao, "Special Issue: The First Provenance Challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 409-418, 2008.

[24] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-Aware Storage Systems," *USENIX Annual Technical Conference*, pp. 43-56, 2006.

[25] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1067-1100, 2006.

[26] A. Rosenthal, "Scalable Access Policy Administration: Opinions and a Research Agenda " in *Security Management, Integrity, and Internal*

*Control in Information Systems*, *IFIP International Federation for Information Processing*, 2006, pp. 355-370.

[27] A. Rosenthal, L. J. Seligman, and S. Renner, "From semantic integration to semantics management: case studies and a way forward," *SIGMOD Record*, vol. 33, pp. 44-50, 2004.

[28] T. Stef-Praun, B. Clifford, I. Foster, U. Hasson, M. Hategan, S. Small, M. Wilde, and Y. Zhao, "Accelerating Medical Research using the Swift Workflow System," *Health Grid*, 2007.

[29] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security Issues in a SOA-Based Provenance System," Security Issues in a SOA-Based Provenance System, 2006.

[30] W. T. Tsai, X. Wei, Y. Chen, R. Paul, J.-Y. Chung, and D. Zhang, "Data provenance in SOA: security, reliability, and integrity," *Journal Service Oriented Computing and Applications*, 2007.

[31] E. Yuan and J. Tong, "Attributed Based Access Control (ABAC) for Web Services," Attributed Based Access Control (ABAC) for Web Services, 2005.

[32] J. Zhang, A. Chapman, and K. LeFevre, "Fine-Grained Tamper-Evident Data Pedigree," *University of Michigan Technical Report*, 2009.