Secure Data Provenance

April 5, 2013
CS6393 - Research Challenges in Cyber Security
(Slide available at http://drjae.com/Activities.html)

Jae Park
Institute for Cyber Security
University of Texas at San Antonio

*World-leading research with real-world impact!*
Provenance Publications

- From Luc Moreau, “The Foundation for Provenance on the Web”
Dictionary Definition of Provenance

• **Oxford English Dictionary**
  – the fact of coming from some particular source or quarter; origin, derivation.
  – the history or pedigree of a work of art, manuscript, rare book, etc.; concretely, a record of the ultimate derivation and passage of an item through its various owners

• **Merriam-Webster Online Dictionary**
  – the origin, source
  – the history of ownership of a valued object or work of art or literature
Data Provenance in Computer Systems

• “The provenance of a piece of data is the process that led to that piece of data”

• “In computer systems, activities are carried out by processes that take input data, input state, input configuration, and produce output data and output state. Such processes are compositional by nature and can be the result of sophisticated compositions (sequential, parallel, conditional, etc) of simpler processes.” (Luc Moreau, “The Foundation for Provenance on the Web”)
Other Definition of Provenance

• **Why-Provenance**
  – identify the set of tuples, whose presence justifies a query result (e.g., \{t2,p3,b2\})

• **Where-Provenance**
  – identify where information was copied from
  – E.g., a typo in mashup is came from b2

• **How-Provenance**
  – a polynomial representation that hints at the structure of the proof explaining how an output tuple is derived
  – E.g., whole graph
Provenance Example
Provenance Data

- Information of operations/transactions performed against data objects and versions
  - Actions that were performed against data
  - Agents who performed actions on data
  - Data used for actions
  - Data generated from actions

- Directed Acyclic Graph (DAG)
- Causality dependencies between entities (acting users, action processes and data objects)

- Dependency graph can be traced for the discovery of Origin, usage, versioning info, etc.
Open Provenance Model (OPM)

- **3 Node Types**
  - Artifact (ellipse): Object
  - Process (Rectangle): Action
  - Agent (Octagon/Hexagon): User/Subject

- **5 Causality dependency edge Types** (not a dataflow)
  - U: Used(Role)
  - G: wasGeneratedBy(Role)
  - C: wasControlledBy(Role)
  - wasDerivedFrom
  - wasTriggeredBy

World-leading research with real-world impact!
OPM Example

100g Sugar -> 100g Flour -> 100g Butter -> Two Eggs -> Bake -> Cake

wasDerivedFrom: Bake -> Cake
wasGeneratedBy: Bake -> Cake
used: 100g Sugar, 100g Flour, 100g Butter, Two Eggs
wasControlledBy: John -> Bake

World-leading research with real-world impact!
Provenance-aware Systems

- Capturing provenance data
- Storing provenance data
- Querying provenance data
- Using provenance data
- Securing provenance data

Provenance Data Model
Access Control
Access control in Provenance-aware Systems

• Provenance Access Control (PAC)
  – Controlling access to provenance data which could be more sensitive than the underlying data
  – Two concerns
    • Needs access control models/mechanisms (e.g., RBAC)
    • (Meaningful) control granularity? Right level of abstraction?

• Provenance-based Access Control (PBAC)
  – Using provenance data to control access to the underlying data
  – Provenance-based policy specification

Meaningful granularity of provenance data?

Typed Provenance Model (TPM)
Access Controls in Provenance-aware Systems

PBAC
- Extended PBAC
- Base PBAC

Access control

PAC
- Sanitization/Filtering/Redaction/....
- Role-based PAC
- Prov-based PAC

Prov Data Trust

Typed Provenance Model (TPM) as a Common Foundation
DName (named abstraction) and matching DPath (Dependency Path Pattern)

Dependency List
Base Provenance data
From Open Provenance Model (OPM)

- 3 Node Types
  - Object (Artifact)
  - Action (Process)
  - User/Subject (Agent)

- 5 Causality dependency edge Types (not a dataflow)

- Provenance data: *a set of 2 entities & 1 dependency*

  - E.g., \((ag,p1,a1,a2)\): <\(p1,ag,c\)>, <\(p1,a1,u\)>, <\(a2,p1,g\)>
Typed Provenance Model (TPM)

- **Primitive (direct) dependency types**
  - Variations of `Used (u)`, `wasGeneratedBy (g)`, `wasControlledBy (c)`
  - Used to capture transactions as base provenance data

- **Complex (Indirect) dependency types**
  - A set of pairs consisting of abstract name (dependency names) and regular expression-based composition of primitive dependency types or other complex dependency types (dependency paths)
Dependency List

- **Object Dependency List (DL<sub>O</sub>):** A set of identified dependencies that consists of pairs of
  - Dependency Name: abstracted dependency names (DNAME) and
  - regular expression-based object dependency path pattern (DPATH)

- **System-computable (complex) dependency instances**
  - using pre-defined dependency names and matching dependency path patterns in DL (and querying base provenance data)

- **User-declared (complex) dependency instances**
  - using pre-defined dependency names in DL

- **Examples**
  - `<wasSubmittedVof, g_submit_.U_input>`
  - `<wasAuthoredBy, wasSubmittedVof?.wasReplacedVof *.g_upload_.c>`
A Family of PBAC Models

Combined Models

PBAC\textsubscript{U} \quad PBAC\textsubscript{A} \quad PBAC\textsubscript{PR}

PBAC\textsubscript{B}
PBAC_B: A Base Model

- System-captured Base Provenance Data only
  - Using sub-types of 3 direct dependencies (u, g, c)
  - No user-declared provenance data
- Object dependency only
- Policy is readily available
  - No policy retrieval required
PBAC_B Model Definitions

1) AU, A, AT, O and OR are acting users, action instances, action types, objects, and object roles respectively.
2) G, U, G_1 and U_1 are sets of role-specific variations of ‘wasGeneratedBy’ and ‘used’ dependencies and matching sets of inverse dependencies, respectively.
3) \{‘c’, ‘c_1’\} is the set of ‘wasControlledBy’ dependency and its inverse dependency.
4) Base provenance data PD_B forms a directed graph and is formally denoted as a triple < V_B, E_B, D_B >:
   - \( V_B = AU \cup A \cup O \), a finite set of acting users, action instances, and objects that have been involved in transactions in the system and are represented as vertices;
   - \( D_B = \{‘c’\} \cup U \cup G \cup \{‘c_1’\} \cup U_1 \cup G_1 \), a finite set of base dependency types;
   - \( E_B \subseteq \{(A \times AU \times ‘c’ \cup (A \times O \times U) \cup (O \times A \times G) \cup (AU \times A \times ‘c_1’) \cup (O \times A \times U_1) \cup (A \times O \times G_1)\} \), denoting dependency edges, is the set of existing base dependencies in the provenance data.
5) DN_O, disjoint from D_B, is a finite set of abstracted names for dependencies of objects.
6) Let \( \Sigma \) be an alphabet of terms in D_B \cup DN_O. The set DPATH of regular expressions is inductively defined as follows:
   - \( \forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH; \)
   - \( (P_1|P_2, (P_1.P_2), P_1*, P_1+, P_1? \in DPATH, \) where \( P_1 \in DPATH \) and \( P_2 \in DPATH \).
7) DPATH_B \subseteq DPATH, is the set of regular expression using only alphabet of terms in D_B.
8) DL_O : DN_O \rightarrow DPATH, defines each \( dn \in DN_O \) as a path expression. DL_O is also viewed as a list of pairs of object dependency names and corresponding dependency paths.
9) \( \lambda_O : DN_O \rightarrow DPATH_B \), maps each \( dn \in DN_O \) to a path expression using only base dependency types \( db \in D_B \) by repeatedly expanding the definitions of any \( dn_i \in DN_O \) that occurs in DL_O(dn).
10) PE is a language specified in the policy expression grammar PG.
11) \( P \subseteq PE \), is a finite set of policies.
12) \( \gamma : AT \rightarrow P \), a mapping of an action type to a policy.
Access Evaluation Algorithm

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

1: (Rule Collecting Phase)
2: \( at \leftarrow a \)’s action type
3: \( p \leftarrow \gamma(at) \)
4: RULE\(_{UA}\) \leftarrow user authorization rules UARule found in \( p \)
5: RULE\(_{AV}\) \leftarrow action validation rules AVRule found in \( p \)
6: (User Authorization Phase)
7: for all rules in RULE\(_{UA}\) do
8: Extract the path rule \((ObjRole, DName)\) from rules
9: Determine the object \( o \in O \), whose role is \( ObjRole \)
10: Extract dependency path expression dpath\(_b\) in DPATH\(_B\) from DName using \( \lambda_O \) function
11: Determine vertices by tracing base provenance data PD\(_B\) through the paths expressed in dpath\(_b\) that start from the object \( o \) using \( \delta_O \) function
12: Determine the truth value by evaluating the result against the rule
13: end for
14: UAuth \leftarrow a combined truth value based on conjunctive or disjunctive connectives between rules
15: (Action Validation Phase)
16: for all rules in RULE\(_{AV}\) do
17: Extract path rules \((ObjRole, DName)\) from rules
18: for all path rules extracted do
19: Determine the object \( o \in O \), whose role is \( ObjRole \)
20: Extract dependency path expression dpath\(_b\) in DPATH\(_B\) from DName using \( \lambda_O \) function
21: Determine vertices by tracing base provenance data PD\(_B\) through the paths expressed in dpath\(_b\) that start from the object \( o \) using \( \delta_O \) function
22: end for
23: Determine the truth value by evaluating the result of all the extracted path rules
24: end for
25: AVal \leftarrow a combined result based on conjunctive or disjunctive connectives between rules
26: Evaluate a final truth value of UAuth and AVal using conjunctive connective
PBAC_B Policy Language Grammar

```
Policy ::= “allow” < Req > “ ⇒ ” < UARules > “∧” < AVRules > | “true”
Req ::= “(” < ActUser > “,” < ActType > “,” < ObjRoles > “)"
ObjRoles ::= < ObjRole > | < ObjRole > “,” < ObjRoles >
UARules ::= < UARule > |“(” < UARules > “)”|
< UARules > < Connect > < UARules >
AVRules ::= < AVRule > |“(” < AVRules > “)”|
< AVRules > < Connect > < AVRules >
Connect ::= ∨|∧
UARule ::= < ActUser > < oper1 > < PathRule >
AVRule ::= “[” < PathRule > “[” < oper2 > < Number > |
< PathRule > < oper3 > < PathRule >
PathRule ::= “(” < ObjRole > “,” < DName > “)”
oper1 ::= “∈”|“∉”
oper2 ::= “=”|“≠”|“≥”|“≤”|“<”|“>”
oper3 ::= “=”|“≠”|“⊆”
DName ::= dn1|dn2|...|dn_n
Number ::= [0-9]+
ActUser ::= au
ActType ::= at_1|at_2|...|at_m
ObjRole ::= o_role_1|o_role_2|...|o_role_k
```

World-leading research with real-world impact!
Example: A Homework Grading System

1. Anyone can upload a homework.
2. A user can replace a homework if she uploaded it (origin-based control) and the homework is not submitted yet.
3. A user can submit a homework if she uploaded it and the homework is not submitted already. (workflow control)
4. A user can review a homework if she is not the author of the homework (DSOD), the user did not review the homework earlier, and the homework is submitted already but not graded yet.
5. A user can grade a homework if the homework is reviewed but not graded yet.
Sample Transactions
Sample Transactions & Base Provenance Data

- (au1, upload1, o\_1v1):
  - \(< upload1, au1, c >, <o\_1v1, upload1, g_{upload} >\>
- (au1, replace1, o\_1v1, o\_1v2):
  - \(< replace1, au1, c >, < replace1, o\_1v1, u_{input} >, < o\_1v2, replace1, g_{replace} >\>
- (au1, submit1, o\_1v2, o\_1v3):
  - \(< submit1, au1, c >, < submit1, o\_1v2, u_{input} >, < o\_1v3, submit1, g_{submit} >\>
- (au2, review1, o\_1v3, o\_2v1):
  - \(< review1, au2, c >, < review1, o\_1v3, u_{input} >, < o\_2v1, review1, g_{review} >\>
- (au3, grade1, o\_1v3, o\_3v1):
  - \(< grade1, au3, c >, < grade1, o\_1v3, u_{input} >, < o\_3v1, grade1, g_{grade} >\>
### Sample Object Dependency List ($DL_o$)

<table>
<thead>
<tr>
<th></th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>&lt; wasReplacedVof, $g_{\text{replace}}.u_{\text{input}}$</td>
</tr>
<tr>
<td>2.</td>
<td>&lt; wasSubmittedVof, $g_{\text{submit}}.u_{\text{input}}$</td>
</tr>
<tr>
<td>3.</td>
<td>&lt; wasReviewedOof, $g_{\text{review}}.u_{\text{input}}$</td>
</tr>
<tr>
<td>4.</td>
<td>&lt; wasReviewedOby, $g_{\text{review}}.c$</td>
</tr>
<tr>
<td>5.</td>
<td>&lt; wasGradedOof, $g_{\text{grade}}.u_{\text{input}}$</td>
</tr>
<tr>
<td>6.</td>
<td>&lt; wasAuthoredBy, wasSubmittedVof?.wasReplacedVof * $g_{\text{upload}}.c$</td>
</tr>
<tr>
<td>7.</td>
<td>&lt; wasReviewedBy, wasReviewedOof$^{-1}$.wasReviewedOby</td>
</tr>
</tbody>
</table>
A Sample Base Provenance Data
wasAuthoredBy
DL₀: <wasAuthoredBy, wasSubmittedVof?. wasReplacedVof *.g_upload.c >
wasReviewedBy
DL₀: < wasReviewedBy, wasReviewedOof⁻¹.
wasReviewedOby >
Sample Policies

1. Anyone can upload a homework.
2. A user can replace a homework if she uploaded it (origin-based control) and the homework is not submitted yet.
3. A user can submit a homework if she uploaded it and the homework is not submitted already. (workflow control)

1. allow(au, upload, o) ⇒ true
2. allow(au, replace, o) ⇒ au ∈ (o, wasAuthoredBy) ∧ |(o, wasSubmittedVof)| = 0.
3. allow(au, submit, o) ⇒ au ∈ (o, wasAuthoredBy) ∧ |(o, wasSubmittedVof)| = 0.
Sample Policies (cont.)

4. A user can **review** a homework if she is not the author of the homework (DSOD), the user did not review the homework earlier, and the homework is submitted already but not graded yet.

5. A user can **grade** a homework if the homework is reviewed but not graded yet.

4. allow(au, review, o) ⇒ au \notin (o, wasAuthoredBy) ∧ au \notin (o, wasReviewedBy) ∧ |(o, wasSubmittedV of)| \neq 0 ∧ |(o, wasGradedOof^{-1})| = 0.

5. allow(au, grade, o) ⇒ |(o, wasReviewedOof)| \neq 0 ∧ |(o, wasGradedOof^{-1})| = 0.
Access Evaluation Procedure

- Rule collecting phase
- User authorization (UAuth) phase
- Action validation (AVal) phase
- conjunctive decision of UAuth and AVal
Access Evaluation Example

- Policy: user can submit a homework if she uploaded it (origin-based control) and the homework is not submitted already. (workflow control)
Rule Collecting Phase

- **Request**: \((au1, \text{submit2}, o_{1v3})\)
- **Action type**: submit
- **Policy for submit**
  - \(\text{allow}(au, \text{submit}, o) \Rightarrow au \in (o, \text{wasAuthoredBy}) \land |(o, \text{wasSubmittedVof})| = 0\).
- **User authorization rule**
  - \(au \in (o, \text{wasAuthoredBy})\)
- **Action Validation rule**
  - \(|(o, \text{wasSubmittedVof})| = 0\)
User Authorization Phase

- **User Authorization Rule:** \( \text{au} \in (o, \text{wasAuthoredBy}) \)
- **Dependency List (DL)**
  - \(< \text{wasReplacedVof}, g_{\text{replace}} \cdot u_{\text{input}} >, < \text{wasSubmittedVof}, g_{\text{submit}} \cdot u_{\text{input}} >\)
  - \(< \text{wasAuthoredBy}, \text{wasSubmittedVof} \cdot \text{wasReplacedVof} \cdot g_{\text{upload}} \cdot c >\)
- \( \text{au1} \in (o1v3, [g_{\text{submit}} \cdot u_{\text{input}}] \cdot [g_{\text{replace}} \cdot u_{\text{input}}] \cdot g_{\text{upload}} \cdot c) = \{\text{au1}\} \)
Action Validation Phase

- **Action Validation Rule:** \(|(o, \text{wasSubmittedVof})| = 0\)
- **Dependency List (DL):** \(< \text{wasSubmittedVof}, g_{\text{submit}} \cdot u_{\text{input}} >\)
- \(|(o1v3, g_{\text{submit}} \cdot u_{\text{input}})| \neq 0\)
the query operation on immutable system captured provenance into account. This paper only considers both P'C and PB'C did not take user declared provenance into account in terms of adjudicating access requests. Currently, provenance can be modified by authorized users using pre-defined rules, and provenance could be added and be sensitive resources in a provenance graph. In both P'C and PB'C, provisions and different security implications for P'C because they have different set of operations and PB'C policies. As a decision factor being evaluated in PB'C policies, provenance could be added and be sensitive resource being protected in P'C policies. Furthermore, they introduced the concept of compatible languages. These languages are usually results of extending on edges of a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. In both P'C and PB'C, dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. Dependency names can be used to enable a simple query result is a meaningful control specification of complex queries on a provenance graph. In both P'C and PB'C, sensitive resources should be protected or used to adjudicate access requests. From PB'C point of view, provenance "user did not review the homework" is the sensitive resource being protected. From PB'C point of view, provenance "owner of a homework" is the sensitive resource being protected. In real settings, provenance-aware policies and other generic attribute-based policies usually need to work together with each other. For example, Part et al. also adopted regular expressions to dynamically identify sensitive provenance. Part et al. also adopted regular expressions to dynamically identify sensitive provenance. Instead, researchers presented policy languages and corresponding enforcement architectures for P'C. These languages are usually results of extending on edges of a provenance graph. Part et al. also adopted regular expressions to dynamically identify sensitive provenance. Part et al. also adopted regular expressions to dynamically identify sensitive provenance. Instead, researchers presented policy languages and corresponding enforcement architectures for P'C.
Implementation and Performance

• System
  – Ubuntu 12.10 image with 4GB Memory and 2.5 GHz quad-core CPU running on a Joyent SmartData center (ICS Private Cloud).

• Implementation
  – extended the PDP class in SUN’s XACML
  – Apache Jena 2.7.4 and ARQ package to provide both the RDF-enabled data store for provenance graph and the ARQ query engine for enabling SPARQL queries.

• Results for tracing 2k/12k edges
  – 0.0096/0.154 second per deep request
  – 0.035/0.04 second per wide request
Summary of PBAC\textsubscript{B}

- Proposed a foundation for PBAC and PAC
  - the notion of named abstractions of causality dependency path patterns
  - Regular expression-based dependency path pattern

- Identified a Family of PBAC models

- Developed a Base model for PBAC
  - Supports Simple and effective policy specification and access control management
  - Supports DSOD, workflow control, origin-based control, usage-based control, object versioning, etc.
• Questions and Comments?