Towards Semantics for Provenance Security

Stephen Chong
School of Engineering and Applied Sciences
Harvard University

Abstract

Provenance records the history of data. Careless use of provenance may violate the security policies of data. Moreover, the provenance itself may be sensitive information, necessitating restrictions on the use of both data and provenance to enforce security requirements. This paper proposes extensional semantic definitions for provenance security. The semantic definitions require that provenance information released to the user does not reveal confidential data, and that neither the provenance information given to the user, nor the program's output, reveal sensitive provenance information.

1 Introduction

The interaction between information security and provenance is complex, and not well understood. When data manipulated by a system contains confidential information, careless use of provenance can lead to violations of information security. When some or all of the provenance information itself is confidential, care must be taken with the use of both data and provenance information to ensure appropriate security is enforced.

It is an open question what constitutes "appropriate security" for provenance. Previous work has considered access control to provenance information (e.g., [4, 1, 2, 7, 6]), but it is unclear what security guarantees this provides. Preventing unauthorized access to confidential provenance is insufficient to ensure that confidential provenance remains confidential. Also, permitting access to (non-confidential) provenance may reveal information about confidential data.

In this paper, we propose extensional semantic definitions for provenance security. We require that provenance information released to the user does not reveal confidential data, and that neither the provenance information given to the user nor the program's output reveal sensitive provenance information.

Data and provenance can have different security requirements. For example, at the end of the peer review process, authors are allowed to know the contents of the reviews, but are typically not allowed to know the identity of the reviewers. Thus, the data (the reviews) are public, but the provenance (who wrote the reviews) is confidential. Another example, due to Braun et al. [1], is a letter of recommendation, where the subject of the letter is typically not allowed to know the contents, but is permitted to know the author. In this case, the data (the contents of the letter) are confidential, but the provenance (who authored the letter) is public.

Many applications manipulate both public and confidential data, and data with different provenance security requirements. In such settings, it is not always clear what provenance information a given user is allowed to access. Some supposedly public provenance may reveal confidential data, and some supposedly public data may reveal confidential provenance. This work provides a semantic basis for determining whether provenance information provided to a user satisfies the security policies of provenance and data.

We build on the work of Cheney, Acar, and Ahmed [3], who propose *provenance traces* as a semantic foundation for provenance in databases. Provenance traces explain a program's execution by showing how the output was produced from the inputs. However, a complete explanation of a program's execution may violate the security of data or provenance: it may reveal confidential data or confidential provenance. Thus, to respect the security of data and provenance, it may not be possible to provide a full provenance trace to a user. We allow parts of a provenance trace to be elided, and define conditions to ensure that the provenance trace contains as much information as possible, while still respecting the security of both data and provenance.

The rest of this paper is structured as follows. Section 2 presents syntax and semantics of a simple language, and defines provenance traces for it. Section 3

$$\frac{}{\sigma, l \Leftarrow e \Downarrow \sigma[l \mapsto \sigma(e)]}\ l \notin \mathrm{dom}(\sigma)$$

$$\frac{\sigma, l' \Leftarrow c_0 \Downarrow \sigma' \quad \sigma', l \Leftarrow c_1[l'/x] \Downarrow \sigma''}{\sigma, l \Leftarrow \mathsf{let}\ x = c_0\ \mathsf{in}\ c_1 \Downarrow \sigma''}\ l' \notin \mathrm{dom}(\sigma)$$

$$\frac{\sigma, l \Leftarrow c_i \Downarrow \sigma'}{\sigma, l \Leftarrow \mathsf{if}\ l'\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \Downarrow \sigma'}\ i = \begin{cases} 0 & \text{if } \sigma(l') \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

Figure 1: Operational semantics

defines security policies for data and provenance. Section 4 defines semantic conditions for provenance security. Section 5 discusses future work.

2 Language

We use a simple calculus that is sufficient to investigate issues in provenance security. We expect the results of this paper to apply in a straightforward manner to richer languages. Programs c include let-commands, conditionals, and expressions e. We assume that expressions contain variables x, literal values v, and locations l, but leave them otherwise unspecified.

$$c ::= e \mid \mathsf{let}\ x = c_0\ \mathsf{in}\ c_1 \mid \mathsf{if}\ x\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1$$
$$e ::= x \mid v \mid l \mid \ldots$$

We define a large-step operational semantics for the language. The operational semantics uses *stores*, which are functions from locations to values. Following the notation of Cheney et al. [3], the judgment $\sigma, l \Leftarrow c \Downarrow \sigma'$ means that given initial store σ, the program c evaluates to store σ', and the result of the program is $\sigma'(l)$; we call l the *output location*, as this is where the result is stored. Inference rules for the semantics are given in Figure 1. We write $\sigma(e)$ for the result of evaluating e with all locations l replaced by $\sigma(l)$. We write $\sigma[l \mapsto v]$ for the store that maps location l to value v and otherwise behaves exactly like σ. We write $c[l/x]$ for the result of substituting location l for every occurrence of free variable x in program c.

Note that a value stored in a location is never overwritten: if $\sigma, l \Leftarrow c \Downarrow \sigma'$ then $l \notin \mathrm{dom}(\sigma)$ and for all $l' \in \mathrm{dom}(\sigma)$, $\sigma(l') = \sigma'(l')$. Also, c will read the value of location l in initial store σ only if l appears in c. We write $\mathrm{loc}(c)$ for the set of locations that appear in c, and refer to $\mathrm{loc}(c)$ as the *input locations* of c.

2.1 Provenance traces

A *provenance trace* summarizes the execution of a program. The provenance traces used in this paper are similar to those of Cheney et al. [3], but we permit parts of the trace to be elided, indicated with the symbol $\star$. The syntax of provenance traces is:

$$T ::= l \leftarrow e \mid T_0; T_1 \mid \mathsf{cond}(l, v, T)$$
$$\mid l \leftarrow \star \mid \mathsf{cond}(\star, v, T) \mid \mathsf{cond}(l, \star, \star) \mid \star$$

Assignment trace $l \leftarrow e$ records that new location l was created, and the value assigned to l was the evaluation of expression e. *Partial assignment trace* $l \leftarrow \star$ records that new location l was created, but does not record the provenance of the value stored there. *Sequential composition trace* $T_0; T_1$ records that first trace T_0 was performed, and then trace T_1. *Conditional trace* $\mathsf{cond}(l, v, T)$ records that a conditional command was executed: location l was evaluated to determine which branch to take, $v \in \{0, 1\}$ indicates which branch was taken, and trace T records the evaluation of the branch. The partial conditional trace $\mathsf{cond}(\star, v, T)$ records which branch was taken, and the trace for that branch, but does not record which location was used as the test. Similarly, the partial conditional trace $\mathsf{cond}(l, \star, \star)$ records which location was used as the test, but not the result of the test or the trace for the appropriate branch. Finally, *unknown trace* $\star$ records no information about the execution.

Given evaluation $\sigma, l \Leftarrow c \Downarrow \sigma'$ and provenance trace T, *trace consistency judgment* $\sigma, l \Leftarrow c \Downarrow \sigma' \vDash T$ states that T is consistent with the evaluation. Figure 2 presents inference rules for the judgment. Intuitively, elided trace $\star$ is consistent with any evaluation, partial assignment trace $l \leftarrow \star$ is consistent with any assignment to location l, and the partial conditional traces $\mathsf{cond}(\star, v, T)$ and $\mathsf{cond}(l, \star, \star)$ are consistent with conditional commands regardless of, respectively, the location tested, and the branch taken. Assignment trace $l \leftarrow e$ is consistent with only command e. Conditional trace $\mathsf{cond}(l, v, T)$ is consistent with conditional command $\mathsf{if}\ l\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1$ provided that branch c_v was evaluated and T is consistent with that evaluation. Finally, trace $T; T'$ is consistent with command $\mathsf{let}\ x = c_0\ \mathsf{in}\ c_1$ if T is consistent with the evaluation of c_0 and T' is consistent with the evaluation of c_1, where variable x is replaced with the output location for the evaluation of c_0.

Trace consistency is similar to consistency as defined by Cheney et al. [3]: both require that the trace describe what happened during the execution. Cheney et al. [3] also define *fidelity*, which requires that traces contain enough information to describe how the program would have executed with different inputs. We ignore fidelity in this paper: traces with parts elided for security purposes may not contain enough information to fully reconstruct the execution of a program, and thus will not satisfy fidelity.[1]

[1] Our disregard of fidelity is also the reason our conditional traces

$$\frac{}{\sigma, l \Leftarrow e \Downarrow \sigma' \ \vDash \ l \leftarrow e} \qquad \frac{\sigma, l' \Leftarrow c_0 \Downarrow \sigma' \ \vDash \ T \quad \sigma', l \Leftarrow c_1[l'/x] \Downarrow \sigma'' \ \vDash \ T'}{\sigma, l \Leftarrow \mathsf{let}\ x = c_0\ \mathsf{in}\ c_1 \Downarrow \sigma'' \ \vDash \ T; T'} \qquad \frac{}{\sigma, l \Leftarrow e \Downarrow \sigma' \ \vDash \ l \leftarrow \star}$$

$$\frac{\sigma, l \Leftarrow c_v \Downarrow \sigma' \ \vDash \ T \quad v = \begin{cases} 0 & \text{if } \sigma(l') \neq 0 \\ 1 & \text{otherwise} \end{cases}}{\sigma, l \Leftarrow \mathsf{if}\ l'\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \Downarrow \sigma' \ \vDash \ \mathsf{cond}(l', v, T)} \qquad \frac{\sigma, l \Leftarrow c_v \Downarrow \sigma' \ \vDash \ T \quad v = \begin{cases} 0 & \text{if } \sigma(l') \neq 0 \\ 1 & \text{otherwise} \end{cases}}{\sigma, l \Leftarrow \mathsf{if}\ l'\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \Downarrow \sigma' \ \vDash \ \mathsf{cond}(\star, v, T)}$$

$$\frac{\sigma, l \Leftarrow c_v \Downarrow \sigma' \ \vDash \ T \quad v = \begin{cases} 0 & \text{if } \sigma(l') \neq 0 \\ 1 & \text{otherwise} \end{cases}}{\sigma, l \Leftarrow \mathsf{if}\ l'\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1 \Downarrow \sigma' \ \vDash \ \mathsf{cond}(l', \star, \star)} \qquad \frac{}{\sigma, l \Leftarrow c \Downarrow \sigma' \ \vDash \ \star}$$

Figure 2: Trace consistency $\sigma, l \Leftarrow c \Downarrow \sigma' \ \vDash \ T$

$$\frac{}{T \leq T} \qquad \frac{}{\star \leq T} \qquad \frac{T_0 \leq T_1}{\mathsf{cond}(l, v, T_0) \leq \mathsf{cond}(l, v, T_1)}$$

$$\frac{}{l \leftarrow \star \leq l \leftarrow e} \qquad \frac{T_0 \leq T_1}{\mathsf{cond}(\star, v, T_0) \leq \mathsf{cond}(l, v, T_1)}$$

$$\frac{T_0 \leq T_1 \qquad T_0' \leq T_1'}{T_0; T_0' \leq T_1; T_1'} \qquad \frac{}{\mathsf{cond}(l, \star, \star) \leq \mathsf{cond}(l, v, T)}$$

Figure 3: Information order $\leq$ on traces

We define an information order $\leq$ on traces. Inference rules for the relation are given in Figure 3. Intuitively, if $T_0 \leq T_1$ then T_0 contains less information about an evaluation than T_1. The following theorem shows that if T_0 contains less information than T_1, and T_1 is consistent with an evaluation, then so is T_0.

Theorem 1 *Given evaluation $\sigma, l \Leftarrow c \Downarrow \sigma'$ and traces T_0 and T_1 such that $T_0 \leq T_1$, if $\sigma, l \Leftarrow c \Downarrow \sigma' \ \vDash \ T_1$ then $\sigma, l \Leftarrow c \Downarrow \sigma' \ \vDash \ T_0$.*

We say that input location l *affects the result* of program c if the result produced by c depends on the value stored in location l. That is, changing the value stored in l has the potential to change the result of the program.

Definition 1 *Given initial store σ and location $l \in \mathrm{dom}(\sigma) \cap \mathrm{loc}(c)$, l* affects the result *of program c if there exists values v_0 and v_1 such that $\sigma[l \mapsto v_0], l' \Leftarrow c \Downarrow \sigma_0'$ and $\sigma[l \mapsto v_1], l' \Leftarrow c \Downarrow \sigma_1'$ and $\sigma_0'(l') \neq \sigma_1'(l')$.*

We write $\mathrm{depend}(c, \sigma)$ for the set of all locations in σ that affect the result of c.

A location may appear in a (consistent) provenance trace of an evaluation of c even if it does not affect the result of c. For example, given the program $l_0 \times 0 + l_1$, location l_0 does not affect the result, and l_1 does. However, both locations are in the trace $l_2 \leftarrow l_0 \times 0 + l_1$, which is consistent with the program's evaluation.

$\mathsf{cond}(l, v, T)$ have fewer annotations than those of Cheney et al. [3].

3 Security policies

In this section we define security policies for locations. The security policies declare whether the data stored in a location is confidential data or public data, and whether the location has confidential provenance or public provenance. A location has confidential provenance if it should not be known whether this location affects the result of the computation.

We assume a set of security levels $\mathcal{S}$ with a partial order $\sqsubseteq$. If $s_0 \sqsubseteq s_1$ then security level s_1 is at least as restrictive as security level s_0. For the rest of the paper we use the two element lattice $\mathcal{S} = \{L, H\}$ where $L \sqsubseteq H$. Level H represents *high confidentiality*, or *secret*, and level L represents *low confidentiality*, or *public*. The results generalize to arbitrary sets $\mathcal{S}$.

Policies for locations are of the form $s_0\ \mathsf{loc}_{s_1}$ where $s_0, s_1 \in \mathcal{S}$. The level s_0 is the *data security level*, the security level to enforce on the contents of a location; s_1 is the *provenance security level* and is the security level to enforce on information about whether the location affects the result of the evaluation.

A *security context for program c* is a function $\Gamma : \mathrm{loc}(c) \rightarrow \mathbf{Policy}$ that assigns security policies to the input locations of c. In this paper, we are concerned with enforcing security policies of input locations; determining and enforcing security policies on intermediate and output locations is left to future work.

For convenience we use Γ to define some sets of locations. Set $\mathbf{DataLow}_\Gamma$ are locations in $\mathrm{dom}(\Gamma)$ with data security level L; set $\mathbf{DataHigh}_\Gamma$ are locations in $\mathrm{dom}(\Gamma)$ with data security level H (or equivalently, the locations not in $\mathbf{DataLow}_\Gamma$). Set $\mathbf{ProvHigh}_\Gamma$ are locations in $\mathrm{dom}(\Gamma)$ with provenance security level H.

$$\mathbf{DataLow}_\Gamma = \{l \in \mathrm{dom}(\Gamma) \mid \Gamma(l) = L\ \mathsf{loc}_s, s \in \mathcal{S}\}$$
$$\mathbf{DataHigh}_\Gamma = \{l \in \mathrm{dom}(\Gamma) \mid l \notin \mathbf{DataLow}_\Gamma\}$$
$$\mathbf{ProvHigh}_\Gamma = \{l \in \mathrm{dom}(\Gamma) \mid \Gamma(l) = s\ \mathsf{loc}_H, s \in \mathcal{S}\}$$

4 Provenance security

Given program c, initial store σ, and security context Γ, we assume that there is a *low observer*, an agent that can observe the contents of any input location $l \in \mathbf{DataLow}_\Gamma$. The low observer does not directly observe the evaluation of c, but given evaluation $\sigma, l \Leftarrow c \Downarrow \sigma'$, the low observer can observe $\sigma'(l)$, the content of output location l. The observer is then given a provenance trace T that is consistent with the evaluation $\sigma, l \Leftarrow c \Downarrow \sigma'$. (The observer could be given the entire provenance trace T, or T could be used to determine the access control policy for the observer's access to provenance.)

To ensure the security policies of input locations are respected, we must define security requirements that ensure neither the program result nor the provenance trace reveal anything about the values stored in locations $\mathbf{DataHigh}_\Gamma$ or which locations in $\mathbf{ProvHigh}_\Gamma$ affect the result of the program. Ideally, the trace T should be maximal in the information ordering $\leq$ while satisfying the security requirements.

In this paper we propose extensional security requirements that restrict what information is revealed by the provenance trace, and that ensure the program result respects the provenance security policies. We do not consider what information the program result reveals about confidential input values; much work in language-based security considers this problem [5].

A naive attempt to ensure that the trace T does not reveal which locations in $\mathbf{ProvHigh}_\Gamma$ affect the result is to require that T does not contain any location in $\mathbf{ProvHigh}_\Gamma$, but allow it to contain any other location, i.e., to require that $\mathrm{loc}(T) \cap \mathbf{ProvHigh}_\Gamma = \emptyset$, where $\mathrm{loc}(T)$ is the set of locations that appear in trace T. This is analogous to using access control to prevent access to parts of the provenance marked as sensitive, but allowing all other accesses.

However, consider the following program.

$$\begin{array}{l}\mathsf{let}\ x = l_0\ \mathsf{in}\\ \mathsf{if}\ x\ \mathsf{then}\ (l_1\ \mathsf{xor}\ l_2)\ \mathsf{else}\ (l_1\ \mathsf{xor}\ l_3)\end{array}$$

where the policies of locations are

$$\Gamma(l_0) = L\ \mathsf{loc}_L \qquad \Gamma(l_1) = H\ \mathsf{loc}_L$$
$$\Gamma(l_2) = H\ \mathsf{loc}_H \qquad \Gamma(l_3) = H\ \mathsf{loc}_H.$$

Note that the low observer is allowed to know the value stored in location l_0, but not the values stored in other locations. The low observer is permitted to learn whether l_1 affects the result, but not whether l_2 or l_3 do. Suppose that the value in location l_0 is 42. Then trace $T = l_4 \leftarrow l_0; \mathsf{cond}(l_4, 0, \star)$ is consistent with the program evaluation, and does not contain any locations from $\mathbf{ProvHigh}_\Gamma$. However, given trace T (and assuming the observer knows the program text) the low observer knows which branch of the if command was taken, and thus can infer that location l_2 affects the result, violating the security policy of l_2. Thus we reject $\mathrm{loc}(T) \cap \mathbf{ProvHigh}_\Gamma = \emptyset$ as a suitable security requirement.

We propose instead *provenance security* which requires that for any location $l \in \mathbf{ProvHigh}_\Gamma$, the provenance trace T is consistent with two evaluations, one in which l affects the result, and one in which l does not affect the result. The two evaluations must have the same values in the low-observable input locations, and must produce the same result. This ensures that neither the provenance trace nor the result reveals which locations in $\mathbf{ProvHigh}_\Gamma$ affect the result.

Definition 2 (Provenance security) *Provenance trace T satisfies* provenance security *for $\sigma_0, l \Leftarrow c \Downarrow \sigma_0'$ exactly when:*

$\sigma_0, l \Leftarrow c \Downarrow \sigma_0' \ \vDash\ T$ *and*
for all $l' \in \mathbf{ProvHigh}_\Gamma$ *there is a store* σ_1 *such that*
  $\sigma_0(l'') = \sigma_1(l'')$ *for all* $l \in \mathbf{DataLow}_\Gamma$ *and*
  $\sigma_1, l \Leftarrow c \Downarrow \sigma_1'$ *and* $\sigma_0'(l) = \sigma_1'(l)$ *and*
  $\sigma_1, l \Leftarrow c \Downarrow \sigma_1' \ \vDash\ T$ *and*
  $l' \in \mathrm{depend}(c, \sigma_0) \iff l' \notin \mathrm{depend}(c, \sigma_1)$.

For any location $l \in \mathbf{ProvHigh}_\Gamma$, provenance security provides plausible deniability: if l affects the result, then there is an evaluation in which l does not affect the result, and that evaluation produces the same result, is consistent with trace T, and has the same values in the low-observable input locations. Similarly, if l does not not affect the result, then there is an evaluation in which l affects the result, and that evaluation produces the same result, is consistent with trace T, and has the same values in the low-observable input locations.

Returning to the example above, we see that the trace $T = l_4 \leftarrow l_0; \mathsf{cond}(l_4, 0, \star)$ does not satisfy provenance security for the program $\mathsf{let}\ x = l_0\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ (l_1\ \mathsf{xor}\ l_2)\ \mathsf{else}\ (l_1\ \mathsf{xor}\ l_3)$, since only evaluations that execute the $l_1\ \mathsf{xor}\ l_2$ branch are compatible with T, and thus T reveals that the result depends on l_2. Indeed, there is no trace that satisfies provenance security for any evaluation of this command, since $l_0 \in \mathbf{DataLow}_\Gamma$, and so the low observer always knows which branch is executed.

Implicit in provenance security is the conservative assumption that the low observer knows the program c that was evaluated. If we make the stronger (and potentially dangerous[2]) assumption that the low observer does not know some parts of the program then we can use a weaker security condition: for every location in

[2] *Security by obscurity* is generally regarded as an inadequate mechanism to ensure security.

$\mathbf{ProvHigh}_\Gamma$ there is both an initial store σ_1 and a program c' such that the evaluation of c' from initial store σ_1 satisfies the appropriate conditions, and c' is appropriately similar to c. Depending on the application, this weaker security condition may be appropriate. For example, if the program is executing on a trusted server, and the observer does not have access to either the source code or binary executable, then it may be reasonable to assume that the observer does not know the program being evaluated, and cannot distinguish programs that take approximately the same amount of time to execute.

Provenance security ensures that neither the result nor the provenance trace reveal information about which locations in $\mathbf{ProvHigh}_\Gamma$ affect the result. Additionally, the provenance trace should not reveal secret input data inappropriately. For example, consider the program

$$\begin{array}{l}\mathsf{let}\ x = l_0 \bmod 2\ \mathsf{in}\\ \mathsf{if}\ x\ \mathsf{then}\ 0\ \mathsf{else}\ 0\end{array}$$

where $\Gamma(l_0) = H\ \mathsf{loc}_L$. Note that the output of the program does not reveal the secret input data stored in location l_0. However, the trace $T = l_1 \leftarrow l_0 \bmod 2; l_2 \leftarrow \mathsf{cond}(l, 1, \star)$ is consistent with an evaluation of the program, satisfies provenance security, but inappropriately reveals that the value stored in l_0 is even.

We propose the security condition *data security* to enforce that provenance does not reveal confidential data. Data security requires that a provenance trace does not reveal anything about values stored in $\mathbf{DataHigh}_\Gamma$ locations: trace T must be consistent with all evaluations from initial states with identical values in low-observable input locations.

Definition 3 (Data security) *Provenance trace T satisfies* data security *for $\sigma_0, l \Leftarrow c \Downarrow \sigma_0'$ exactly when:*

$$\begin{array}{l}\sigma_0, l \Leftarrow c \Downarrow \sigma_0' \ \vDash\ T \textit{ and}\\ \textit{for all } \sigma_1 \textit{ such that}\\ \qquad \sigma_0(l') = \sigma_1(l') \textit{ for all } l' \in \mathbf{DataLow}_\Gamma\\ \quad \sigma_1, l \Leftarrow c \Downarrow \sigma_1' \textit{ implies } \sigma_1, l \Leftarrow c \Downarrow \sigma_1' \ \vDash\ T.\end{array}$$

Like provenance security, data security provides plausible deniability: for any evaluation of the command that is consistent with trace T, there is another evaluation that is also consistent with T where the public input values are the same, but the secret input values are completely different. Returning to the example above, the trace $T = l_1 \leftarrow l_0 \bmod 2; l_2 \leftarrow \mathsf{cond}(l, \star, \star)$ satisfies data provenance for any evaluation of the command $\mathsf{let}\ x = l_0 \bmod 2\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ 0\ \mathsf{else}\ 0$.

5 Future work

The ultimate goal of this work is to provide precise, useful, and intuitive, semantic definitions of provenance security, and efficient mechanisms for enforcing them. This paper proposes some semantic definitions for provenance security. Much work remains, including constructive techniques for producing maximal provenance traces that satisfy the security conditions, and proof techniques for proving provenance security and data security for all possible evaluations of a program. Extensions to this work include defining other elisions of provenance traces, and considering the impact of treating provenance as first-class entities, allowing the program to inspect provenance as it executes.

This work is compatible with the use of (discretionary) access control to restrict users' access to provenance: it provides a semantic basis for determining the access control policy for provenance information.

In many provenance-aware applications intermediate results will persist after program execution terminates. As such, it is important to determine appropriate security policies and semantic security requirements for intermediate results. Currently, this work considers specification and enforcement of security only for input locations, and assumes the value in the output location is observable by the low observer.

Acknowledgments

Thanks to Kim Bruce, Melissa O'Neill, and Chris Stone for giving useful feedback on an earlier version of this work. This paper was written while visiting the Computer Science Departments at Pomona College and Harvey Mudd College. Thanks also to the anonymous reviewers for their useful comments.

References

[1] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security*, 2008.

[2] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Secure scientific workflow provenance querying with security views. In *Proceedings of the Ninth International Conference on Web-Age Information Management*, pages 349–356. IEEE Computer Society, 2008.

[3] J. Cheney, U. Acar, and A. Ahmed. Provenance traces. arXiv:0812.0564v1, December 2008.

[4] R. Hasan, R. Sion, and M. Winslett. The case of the fake Picasso: Preventing history forgery with secure provenance. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies*, June 2009.

[5] A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1): 5–19, Jan. 2003.

[6] C. Sar and P. Cao. Lineage file system. Online at `http://theory.stanford.edu/~cao/lineage`.

[7] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security issues in a SOA-based provenance system. *Lecture Notes in Computer Science*, 4145, 2006.