One of These Records Is Not Like the Others

Carrie Gates
CA Labs
520 Madison Avenue
New York, NY 10022

Matt Bishop
Dept. of Computer Science
University of California at Davis
Davis, CA 95616-8562

Abstract

This position paper argues the need to develop provenances, and provenance systems, in such a way that errors in the provenance (whether deliberate or not) can be detected and corrected. The requirement that a provenance have high assurance leads to some suggestions about the way a provenance should be constructed.

1 Introduction

The utility of data provenance has long been recognized as both a way to provide information ("metadata") to help users of the data evaluate its integrity and, possibly, confidentiality. Numerous papers discuss how to protect the provenance, consisting of a chain of *provenance records*, against tampering (see for example [6–8]. Underlying this is an assumption that the data making up the provenance is correct when the provenance entry is created and bound to the other entries in the provenance, for example by the use of provenance monitors [9].

We examine what happens when this assumption is incorrect—that is, the data in the provenance record is either incorrect or missing. Our question is, under what conditions is it possible to *detect* the problem, and what additional information must be present or available to *recover* or reconstruct the correct or missing provenance record? We examine the following cases:

1. One or more provenance records (called a "broken provenance" or "broken provenance records") are missing or incomplete. This may occur because the data transits systems that do not support gathering the requisite provenance data, or because a process that gathers information to add to the provenance, or that actually adds a provenance record, fails.
2. One or more of the providers provides incorrect, incomplete, or nonexistent information to add to the provenance. These providers do not modify any records illicitly. Instead, the provenance record created as a result of the data they provide lacks integrity—that is, it is not trustworthy. We call this an "untrustworthy provenance" or "untrustworthy provenance record" and it is a form of the type of attacks collectively called "insider attacks," because a trusted entity (the entity creating the provenance record) betrays that trust.

Collectively, we refer to these cases as a "bad provenance" or "bad provenance record."

The theme of this position paper is that provenance records must be designed to detect the above failures, to enable those who use the provenance records to assess their completeness and trustworthiness. We do not discuss making the records tamperproof, because the bogus information is inserted *before* any sealing of the record in question is done. We instead focus on inconsistencies, gaps, and bad actors.

Provenance requirements specify what each provenance record must contain. There are two types of requirements:

1. Requirements necessary for the use of the provenance ("use requirements"). These are application- and environment-specific.
2. Requirements necessary for the validation of the provenance itself ("test requirements"). These include extra data required for consistency checking.

For this note, we assume that the first set of requirements is known *a priori*. We focus on the second set.

We also refer to "generators" and "users" of provenances and provenance records. A generator adds a provenance record to the provenance; a user reads the records and analyzes them. A node may be both a user and a generator; without loss of generality, we will assume it consumes before it produces, so the provenance record may depend upon earlier records.

We first discuss our threat model. We then present two examples in which accuracy of provenance is important. Next, we examine detection and recovery and apply our suggestions to those two examples. We conclude with some suggestions for future work.

2 Threat Model

We use the representation of a provenance system developed by the Open Provenance Model [11]. The directed acyclic graph includes nodes that produce provenance records. The threats are:

1. A generator modifies a provenance record.
2. A generator deletes an earlier record.
3. A generator deletes *all* earlier provenance records, and begins a new provenance.
4. A generator fails to add a provenance record when it is expected to.
5. A generator enters incorrect information into the provenance record it is creating.

In general, threats can be *prevented*, *detected*, and *recovered from*. Unfortunately, a provenance mechanism cannot prevent broken or untrustworthy provenances or records in the absence of systems that are fully trusted by the users, because the generators control the systems on which the data to be added to the provenance is generated.[1] We thus focus on detection and recovery.

3 Examples

3.1 Real Estate Recordation

Whenever one purchases real estate (such as a house) in the state of California in the United States, a public record of the sale is filed at the county Recorder's Office. This record is added to a file that contains all previous records for that property. Thus, one can establish a record of property ownership (including liens and other property-related constraints) by examining this file.[2] In a paper-based world, the documents are filled out, and then a courier takes them to the county Recorder's Office. In California, these documents can be recorded over the Internet [3, 5].

Figure 1 shows the workflow of a document through the Electronic Recordation Delivery System (ERDS). The document is either scanned or created digitally, and then placed onto a workstation (ASW) controlled by an Authorized Submitter. The Authorized Submitter digitally signs the document and then encrypts it (this is the "payload"). The ASW then connects to the County Recorder's proxy server and authenticates itself to that server. If successful, the proxy server acts as a proxy for the County ERDS server, and the ASW transmits the payload to the ERDS server via this proxy. Subsequently, the workstation (CRDW) of the County Recorder Designee, who will examine the documents, connects to the ERDS server and retrieves the payload. The payload is decrypted and then validated. If the validation fails, the payload is rejected. Otherwise, the documents are extracted and checked for malware. If none are found, the documents are stored on the CRDW. The County Recorder then examines the documents and either rejects them or stamps them as accepted and files (records) them.

Represent each provenance record as $r_x = (a_{1,x}, \ldots, a_{n,x}, X)$, where $a_{i,x}$ is the ith attribute of interest entered in the provenance record, and X is the rest of the record. We assume that each provenance record is cryptographically signed, and that the records are cryptographically chained together, so we do not explicitly show this (we consider them a part of X).

1. The ASW generates the first provenance record r_1, containing the identity $a_{1,1}$ of the signer.
2. The ASW then generates the second provenance record r_2, containing the host $a_{2,1}$ and site identity $a_{2,2}$, and chains this record to the provenance chain.
3. The proxy server adds another provenance record r_3, containing its identity $a_{3,1}$ and where it received the payload from $a_{3,2}$, to the chain.
4. The ERDS server adds another provenance record r_4, containing its identity $a_{3,1}$ and where it received the payload from $a_{3,2}$, to the chain.
5. When the CRDW contacts the ERDS server, the ERDS server adds another provenance record r_5 to the chain. It contains information $a_{5,1}$ about the requester (CRDW), the ERDS server identity $a_{5,2}$, and where it received the payload from $a_{5,3}$.
6. The CRDW generates its own provenance record r_6, containing its identity $a_{6,1}$ and information $a_{6,2}$ about where it retrieved the provenance record from.
7. When the County Recorder Designee examines the document, the CRDW generates another provenance record r_7 showing who is examining the records $a_{7,1}$ and the disposition $a_{7,2}$, and adds it to the chain.

Thus, the provenance chain is $C = r_7 r_6 r_5 r_4 r_3 r_2 r_1$.

3.2 Co-operating Competitors

Many competitors work together on joint projects. For example, airplanes developed for militaries may include parts engineered by many different commercial organizations that, in the civilian world, market aircraft that compete with one another. Consider a hypothetical next generation tactical strike fighter with components from Boeing, EADS/Airbus, Lockheed-Martin, and Northrop-Grumman. To ensure the parts work together, the companies have established a trusted repository for shared doc-

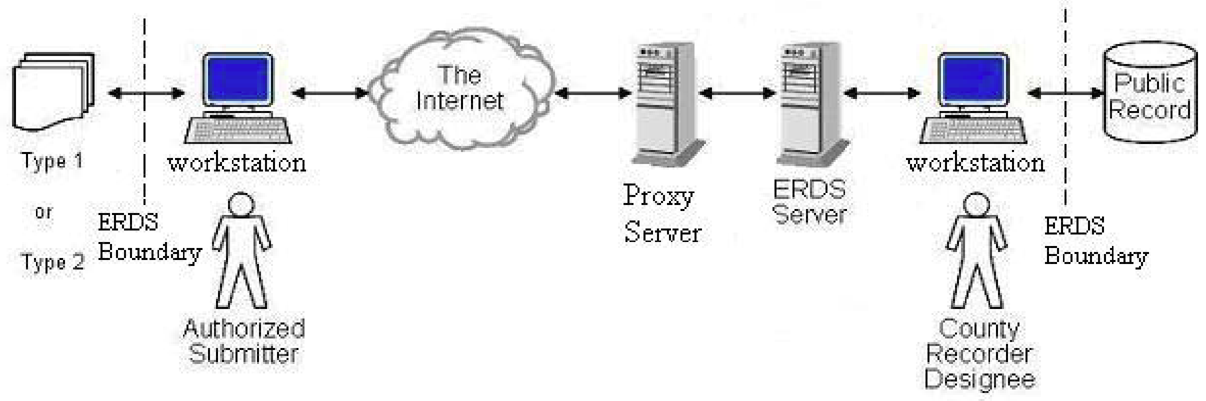

Figure 1: Overview of an ERDS system. The document is put onto the workstation at the office, and transmitted to the workstation at the County Recorder office, where it is vetted and, if found valid, made a part of the public record. [4, p. 10]

uments. These documents may be shared with all partners, or among a subset of partners in case one company does not want all partners to see it.

The shared repository generates provenance records for retrieval and insertion (or replacement) of documents, and the provenance chain is attached to the document. But the repository does not check provenances for consistency. The organizations must do that themselves.

When a document is retrieved, it is to be edited on a company workstation or laptop that uses third-party software to generate and insert provenance records. Consider what happens when the vendor of those operating systems patches software, such as the editing software on these systems. The third-party provenance software may no longer work with the patched programs, requiring the use of a non-augmented system until the third party upgrades its software (or another patch fixes the problem). The non-augmented system will not append a provenance record.

Note that the non-augmented system may not have access to the provenance records, so if the document must be uploaded before the problems mentioned above are fixed, the entire provenance record chain will be deleted.

4 Detection

According to the classic definition, detection mechanisms monitor a system and look for indications of attacks [1]. For this note, the provenance is the "system," so we seek indications that the provenance is broken or untrustworthy. We consider the threats identified above.

Cryptographic techniques can handle attacks that instantiate the first two threats. In particular, if a generator digitally signs the generated provenance record, any subsequent modification of that record will be detected. Similarly, a chaining scheme using cryptographic hashing will bind each record to the next, so the deletion of any previous record can be detected. Our examples both assume the use of these techniques.

If a generator deletes an entire existing provenance and begins a new one, in effect the generator has simply created a new document and begun a provenance with that document. As there are no internal indications of the deletion of the provenance in either the document or the new provenance, the detection of this action must rely on external data. This situation is similar to someone deleting an author's name from a paper and adding her own.

The detection of attacks instantiating the fourth threat depends in part on the nature of the attack. If the attack modifies the data, then the discrepancy between the hash of the data in the last provenance record and the computed hash of the current data will show that an unexpected change occurred. But if the attack does not involve modification of the data, and is merely a failure to add the record, then cryptography does not detect that the document has been accessed (or that a provenance record is missing).

This situation reduces to the fifth threat, in which a generator enters incorrect information into the provenance record it is creating (in the degenerate case, the record is empty). We can use two techniques: consistency checking and examination of external information.

We do consistency checking by looking within the provenance itself, for inconsistencies within a record (for example, syntactic invalidity) and inconsistencies between records (for example, semantic discrepancies). These inconsistencies may not be the result of an attack, but they do raise questions about the trustworthiness of at least one creator of provenance records. A trusted third party would be able to do this impartially.

Consider a provenance record that, as part of the use requirements, contains information about the host through which the data passes. Then the provenance record will contain the identity of the host—for our purposes, say the fully qualified domain name. A second (test) requirement might be that the provenance also contain the IP address of the host from which the data was received. Then a simple consistency check is to verify that the fully qualified domain name in each provenance record corresponds to the IP address of the "received-from" host in the next record.

EXAMPLE: The recordation provenance requires checking that $a_{2,1} = a_{3,2}$, $a_{3,1} = a_{5,3}$, $a_{5,1} = a_{6,1}$, and $a_{5,2} = a_{6,2}$.

EXAMPLE: The document provenance record shows that the document was taken out by Boeing and then returned by Lockheed-Martin. This discrepancy may be legitimate, but it may also indicate a loss of provenance records or something more sinister.

Another example arises from a use requirement that the provenance record contain a time stamp saying when the data was last modified. Without loss of generality, we assume all such timestamps are in UTC. The test for consistency is simply that the timestamps be increasing[3] because clocks do not run backwards. This example also shows that an inconsistency may be innocent. Here, a system clock may be incorrect due to clock drift, or a system administrator may reset a clock incorrectly.

EXAMPLE: Suppose Eleanor is to finish her revisions of a document by 3PM on Wednesday. Due to a bad case of writer's block [12], she does not download it until 4PM on Wednesday. She finishes it at noon on Thursday. She then adjusts her laptop's time back to Wednesday. When the provenance record is created, it will show the document completed at noon on Wednesday. But the prior record will have been created at 3PM on Wednesday.

A third example occurs when a provenance is added to data that follows a workflow. A workflow implies a model of computation, and the goal of the provenance associated with the data is to demonstrate that the workflow is indeed followed. This means that one can derive properties that the provenance, and provenance records, should satisfy [10]. If they do not, there is an inconsistency implying that the provenance is bad.

EXAMPLE: The recordation system requires that the signer be authorized to use ERDS, that the ASW be a certified workstation, and that the company be properly registered to use ERDS [3, 5]. So the validation would check that $a_{1,1}$ is in the list of authorized signers, that $a_{2,1}$ is in the list of certified workstations, and that $a_{2,2}$ is a company authorized to use ERDS. Also, the document examiner must be someone who is a County Recorder Designee, and this can be verified by checking $a_{7,1}$.

Thus, designing provenance systems—and in particular the contents of provenance records—must include test requirements as well as use requirements. Two complications arise.

The first is the generators who have direct access to the provenance data they create. The key observation with respect to generators of untrustworthy records is that their goals must align in some way with defeating the use to which the provenance will be put. That they have access to this data means that they can alter it, and undermine the provenance. The "insider" can be modeled as a continuum of access to a resource and the criticality of the use to which that resource will be put [2]. Under this model, one must consider the ability of each generator to help realize the specific threats against which the provenance guards as well as the access to the relevant fields of the provenance record that that provider has.

The second is the issue of concealment. In particular, a rogue generator who can examine other records in the provenance may be able to supply incorrect information for the new record, and do so in such a way as to defeat detection mechanisms. This means that protecting the confidentiality of each provenance record from providers other than the creator of that record may enhance the detection capability.

Weighing against the confidentiality of the provenance is the desire for an "open provenance," in which each of the intermediate actors can use the previous provenance records to validate the data for their own use. Information in general will be used once it is known, and so it is probable that the data being described by the provenance will play a role in the actions of the nodes adding new provenance records. In such a case, concealing those records in order to meet a security goal will detract from the intended functionality of the system. This emphasizes that security is a trade-off with functionality.

Information used to detect broken or untrustworthy provenances can come from either internal or external sources. Thus far, we have focused on internal sources. Data from external sources must be evaluated carefully. If the assurance of the external data is greater than that of the provenance, and the two are inconsistent, the provenance is suspect. When external data indicates problems with the provenance, the specific indication, and the information involved, must be recorded. This means either the record must contain a field for "comments" or "additional information," or a new type of provenance record (a "recovery record") must be added. This record documents the problem, and any assumptions and recovery actions undertaken (see Section 5).

When validation of a provenance, or provenance record, fails, the entity doing the validation has three options. It can discard (ignore) the provenance, it can continue to propagate the provenance with an annotation that the validation failed (as noted above), or it can attempt to recover the correct data for the provenance.

EXAMPLE: If the provenance of the document to be recorded fails to validate, the ERDS system is almost certainly not compliant with the governing regulations. The document would be rejected, with a new provenance record being added to reflect the rejection. Thus, the provenance would continue to propagate, with the record showing rejection as part of the chain.

EXAMPLE: If the shared document's provenance failed to validate properly, then what happens is based on policy. The record could be ignored, but most likely would not be. Certainly a record for the failed validation would be created. The company may also make inquiries among its collaborators to uncover the reason for the discrepancy.

We now examine the third alternative.

5 Recovery

Once a user has detected a broken provenance, it may try to recover. Recovery could take many forms.

For a broken provenance, the goal would be to take the point at which the provenance broke (which was identified when the broken provenance was detected), and determine whether the data left the systems in the provenance system or whether a generator omitted adding a record. From that, one could determine whether recovering the missing information is possible, and if not perhaps what happened to the data during the missing time.

For an untrustworthy provenance, the recovery would begin with the identification of the record(s) at which the untrusted data was inserted. Then the specific untrusted data would need to be determined, and replaced by the original or correct data.

We suggest two approaches to this task, and then a generalization of them. In many cases, the particular structure and uses of provenance records will enable other methods of recovery.

The Open Provenance model represents generators as nodes and provenance flows as directed edges between those nodes. Consider an "overlay" graph complementary to that DAG, constructed as follows. Let node **n** be a user. Add edges from **n** to each node representing a generator of a provenance record used by **n**. This represents the paths along which **n** may request additional information to validate the provenance. Edges from the generators to **n** indicate which generators will supply such information.[4] This model parallels recovery from network problems, where information is resent in response to a request from a recipient. Here, the edges from the user to the generator play the role of a path along which a request is sent, and the return path is that along which a response is sent. If a generator does not retain a copy of (or a pointer to) the provenance record it created, or cannot reconstruct it, the generator may simply say so, leaving the user no worse off than had it not asked.

If the generator provides incorrect information—in other words, a lie—there is little that can be done. The user may accept the incorrect information as authoritative or it may notice a discrepancy, in which case it can regard the record as unrecoverable.

An alternate approach draws from the notion of "shadow keys" in cryptography. There, one breaks a cryptographic key into n parts, any $t < n$ of which are sufficient to reconstruct the original key.

Construct the provenance in such a way that any m out of n generators whose records make up the current provenance can reconstruct the whole provenance. When an untrustworthy or broken provenance is received, the user must contact any m generators to fix it. As some of the generators may be malicious or non-responsive, the user might need to contact members of overlapping sets of m generators to determine the correct provenance. This problem is analogous to various problems in cryptography, and similar approaches may prove useful.

To handle a rogue generator, have n entities construct each provenance record, such that any m of them could reconstruct the record. This enables recovery of individual records. For this to work, one would need to design redundancy into the fields of the provenance record, or add "shadow fields" that conceal or bind parts of the correct data in a way such that the generator cannot tamper with it. This seems impossible given a corrupt generator.

Both of these ideas have two bases: the notion of *tamperproofness* and the notion of *redundancy*. Ultimately, to enable full recovery, either the provenance records must be stored in such a way that the provenance cannot be tampered with, or there must be enough redundancy in the provenance to enable reconstruction of the provenance. Note that cryptographic hashes do not provide this capability, because they are one-way functions. We need to find the record with that hash, but by definition of "one-way function," that is impossible.

This suggests storing each version of the provenance in the next record. Let p_i be the ith provenance record, and let d_i be the data to be stored in it. Then:

$$\begin{aligned} p_0 &= d_0 \\ p_i &= d_i \,||\, f(p_{i-1}) \end{aligned}$$

where $||$ means concatenation and f is some invertible transformation of the provenance record.

Assume the adversary alters the value of $f(p_{i-2})$ in the previous record. To avoid detection, she must then alter the record p_{i-2} to match the value stored in record p_{i-1}. But then she must alter p_{i-3}, because that record is stored in p_{i-2}. This progression continues throughout the provenance chain. The only way to alter $f(p_{i-2})$ undetectably is to alter all previous records. If each record also includes a digital signature, then detection is unavoidable, even if recovery is not possible.

Also, if f is an encryption function, a user will notice that the decrypted record is syntactically incorrect, and discard it.

For recovery, the provenance must be stored in a tamperproof area. This requires a trusted repository for provenance data being sent over a network; pointers to the data would be passed around. As the trusted third party prevents any data from being altered, this gives the tamperproof property. But the pointers themselves could be tampered with. So the trusted third party must also save the referenced data. Thus each generator commits its provenance record and changes to the data (if any) to the trusted third party, and then tells the next generator how to retrieve the data from the trusted third party.

6 Conclusion

This note examined assurance of the data *contained in* the provenance. While detecting that a provenance has been tampered with is easy, detecting that provenance data is missing, incomplete, or incorrect is harder, and reconstructing those records is even more difficult. Thus, the design of provenance records and a provenance system must have redundancy and tamperproof mechanisms to minimize the difficulty of detecting and correcting these incomplete, missing, or erroneous records.

Acknowledgements: Thanks to Bertram Ludäscher for helpful comments.

Matt Bishop was supported by NSF grants CCF-0905503 and CNS-1049738. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors.

References

[1] BISHOP, M. *Computer Security: Art and Science*. Addison-Wesley, Boston, MA, Dec. 2002.

[2] BISHOP, M., AND GATES, C. Defining the insider threat. In *Proceedings of the Fourth Annual Workshop on Cyber Security and Information Intelligence Research* (New York, NY, USA, May 2008), F. Sheldon, A. Krings, R. Abercrombie, and A. Mili, Eds., CSIIRW '08, ACM, pp. 15:1–15:3.

[3] CALIFORNIA CODE OF REGULATIONS. Title 11 (Law) Division 1 (Attorney General) Chapter 18 (Electronic Recording Delivery System), 2005.

[4] CALIFORNIA DEPARTMENT OF JUSTICE. *Electronic Recording Delivery System Baseline Requirements & Technology Standards Handbook*, Sep. 2007.

[5] CALIFORNIA GOVERNMENT CODE. Electronic Recording Delivery Act of 2004, 2004.

[6] FACTOR, M., HENIS, E., NAOR, D., RABINOVICI-COHEN, S., RESHEF, P., RONEN, S., MICHETTI, G., AND GUERCIO, M. Authenticity and provenance in long term digital preservation: Modeling and implementation in preservation aware storage. In *Proceedings of the First Workshop on the Theory and Practice of Provenance* (Feb. 2009).

[7] HASAN, R., SION, R., AND WINSLETT, M. Introducing secure provenance: Problems and challenges. In *Proceedings of the 2007 ACM Workshop on Storage Security and Survivability* (Oct. 2007), pp. 13–18.

[8] LYLE, J., AND MARTIN, A. Trusted computing and provenance: Better together. In *Proceedings of the Second Workshop on the Theory and Practics of Provenance* (Feb. 2010).

[9] MCDANIEL, P., BUTLER, K., MCLAUGHLIN, S., SION, R., ZADOK, E., AND WINSLETT, M. Towards a secure and efficient system for end-to-end provenance. In *Proceedings of the Second Workshop on the Theory and Practics of Provenance* (Feb. 2010).

[10] MISSIER, P., LUDÄSCHER, B., BOWERS, S., DEY, S., SARKAR, A., SHRESTHA, B., ALTINTAS, I., ANAND, M. K., AND GOBLE, C. Linking multiple workflow provenance traces for interoperable collaborative science. In *Proceedings of the 5th Workshop on Workflows in Support of Large-Scale Science* (Nov. 2010), pp. 1–8.

[11] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E., AND DEN BUSSCHE, J. V. The open provenance model core specification (v1.1). *Future Generation Computer Systems 27*, 6 (2010), 743–756.

[12] UPPER, D. The unsuccessful self-treatment of a case of "writer's block". *Journal of Applied Behavioral Analysis 7*, 3 (Fall 1974), 497.

Notes

[1]They may also control the system that actually adds the provenance record to the provenance.

[2]A copy of all documents is also kept in a state repository in a vault in mountains, to protect against catastrophic loss.

[3]Or equal, depending on the granularity of the clocks.

[4]For legal or business reasons, generators may decline to comment further on the provenance record they created, or to supply additional information to certain users.