Verifying Digital Provenance in Web Services

Ben Palmer, Kris Bubendorfer, and Ian Welch
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {ben,kris,ian}@ecs.vuw.ac.nz

***Abstract*—Web services often combine or "mashup" a collection of heterogeneous data sources. Service providers take data from various sources, including other service providers, and perform some computation or combination of the results and present it to the user. This paper is concerned with the provenance of data provided by web services. Provenance for services includes where the information that is provided by the service originated and who has operated on it. We use a provenance tag that is passed along with the result of the service and contains enough information to recreate a provenance graph. We consider the methods a malicious participant could use to try and fake this provenance information and provide a threat model and security analysis to show our protocol prevents these attacks. We also discuss exclusion attacks where a service provider tries to exclude some input from the provenance information provided.**

***Keywords*-Provenance; Verification; Web Services**

I. Introduction

Web services provide new mechanisms for users to access services. In web services, service providers collate and present information from a collection of heterogeneous data sources. These sources can include databases, web applications, and other service providers.

The basic model for web services is shown in Figure 1. Consider an example of a web service that lets users display their photos along with a map showing the location the photo was taken. The service provider may be using a storage provider and a mapping provider. So in this example, the service provider is providing a service to the user that is a combination of the service providers own algorithms and the functionality provided by two external service providers.

A service may use information from several different service providers. Depending on the reputation of the service provider being used, customers may have differing expectations of the quality of the data as well as the amount they are willing to pay for the service. A service provider that is using a premium service will want to be able to charge more for this service. However, a malicious service provider could claim to be using a premium service when they have not actually purchased them and are using free services provided by a different service provider.

In this paper, we consider the concept of provenance for web services. The provenance of the data from the service provider includes the origin of the data, as well as information on any modifications that have been done to the data. In the ideal model, the customer would be able to check the provenance of all data that it receives from the service provider. The provenance of the data from the service provider includes the origin of the data, as well as information on any modifications that have been done to the data. As a web service provides an ongoing series of data provided to a customer, the provenance for a web service needs to be continually checked. The service provider may dynamically change the source of data when providing a web service and this change needs to be reflected in the provenance information. Provenance is a well established research area and we use the model of provenance developed by the Open Provenance Model [1]. We then create a protocol that can provide this provenance information correctly in the presence of malicious parties.

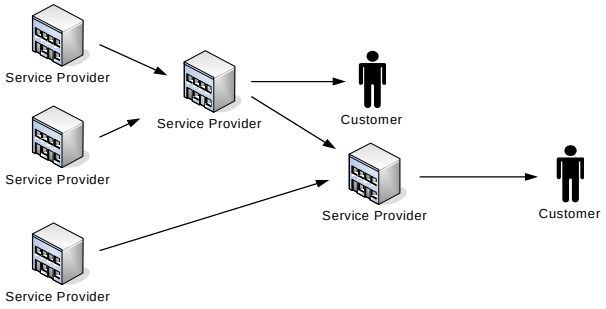

Figure 1. Web Services Model

II. Related Work

Provenance of data has been an active research area for some time with previous work concentrating on individual domains [2] and high level solutions intended to be able provide provenance in multiple domains [1], [3]. However, in our work we concentrate on security for provenance information in web services.

A good high level view of the security issues when providing provenance information in a distributed environment is presented by Tan et al [4]. The authors present a list of security issues and provide some initial ideas on solving these problems. To address the issue of trustworthiness of participants that are creating and providing provenance

information for data they suggest using digital signatures and a reputation based system.

Access control for provenance stores is addressed in a number of works [4], [5], [6]. In these works the problem of securing access to provenance information is addressed. A provenance store is a database of provenance information that can be queried to provide provenance information for artifacts. We do not consider access control for provenance information but it would be possible to apply similar techniques to our solution.

The open provenance model is applied to distributed systems in work by Groth et al [7]. They apply the open provenance model to distributed systems but do not consider attacks from active adversaries on provenance information.

Hasan et al have constructed a protocol for securely collecting provenance information from distributed sources with very similar goals to our work [8], [9]. They consider a very similar domain model to the one we have chosen. They also implement the provenance information as a chain of provenance records in the same way we do. Our approach differs to theirs in the exact representation of the provenance information that is passed along the chain. We also do not focus on confidentiality of provenance records as they do, although we could apply the same techniques to our work as they do to provide confidentiality. The main differences in our work are that we discuss how to prevent exclusion attacks on provenance information provided by honest participants and make use of the open provenance model.

Zhang et al have applied similar techniques as the work by Hasen et al to databases [10]. They have a more complex data model and support non linear provenance objects. They do not include exclusion attacks in their threat model.

While the open provenance model has been applied to distributed systems in previous work, no work looks at the application of the open provenance model in the presence of active adversaries. We also provide a more in depth discussion on methods to prevent exclusion attacks than previous work.

III. Domain Model

We define two roles that a participant in a web service can take: service providers and users. A participant may change roles, for example, a user may become a service provider by taking the service provided to it and combining it with some other service or computation to produce a new service.

- Service Providers. Service Providers make use of a heterogeneous collection of data sources to provide a service. A service provider is uniquely identified by a Uniform Resource Identifier (URI).
- Users. Users are the end user of the service provided by a service provider.

The Open Provenance Model group have defined a set of terms for describing the provenance of items [1]. We make use of these terms to define our model of provenance for web services. In particular we use the following terms:

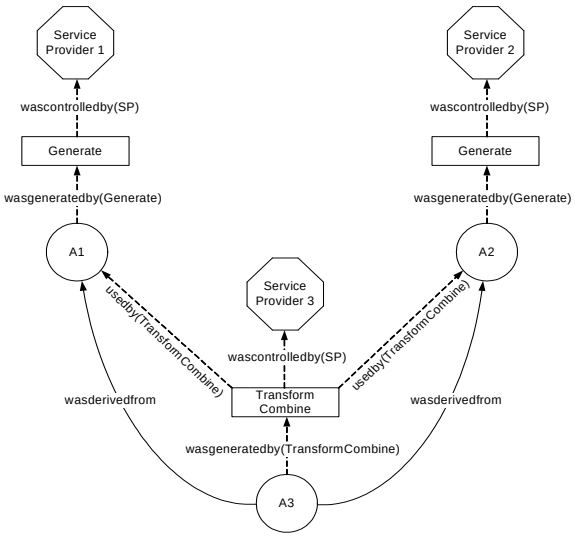

Figure 2. Web Services Open Provenance Model

- Artifact: An immutable piece of state. In our model an artifact is the result of a service. Artifacts are created by service providers, passed to other service providers or customers.
- Process: An action or series of actions performed on or caused by an artifact. In our model a process is either Generate where a service provider generates some data or TransformCombine where a service provider takes some artifacts and performs some actions on them to create a new artifact.
- Agents: A contextual entity controlling a process. In our model all agents are service providers as users do not create provenance information.

The Open Provenance Model also defines some actions, most of which are parametrised by the role of the process or agent that is performing the action. We make use of the following actions:

- wascontrolledby(Role): A process is controlled by an agent. The Role parameter is the role the agent is taking in the protocol. We define the roles as SP for service providers.
- wasgeneratedby(Role): An artifact is generated by a process. The Role parameter is the process that is generating the artifact, in our model this is Generate or TransformCombine.
- usedby(Role): An artifact is used by a process. The Role parameter is the process that uses the artifact, in our model this is the TransformCombine process.
- wasderivedfrom: Artifacts can be derived from other artifacts. There is no role associated with this action.

In the Open Provenance Model provenance is shown in a provenance graph which is a directed graph where agents are shown as octagons, processes as rectangles, and artifacts as circles. Directed edges in the graph represent an action with the source of the action being the end point of the arrow and the start of the arrow showing the result of the action. Figure 2 shows a provenance graph for a service with two service providers acting as input to a third service provider. In this work, we construct a protocol to generate and verify this provenance information in the presence of untrusted service providers.

IV. Threat Model

We have grouped the ways a malicious service provider could try and defraud a user with incorrect provenance information in to the following categories:

1) Fabrication. The adversary tries to create provenance information for an honest participant in the protocol when it has never obtained the service from the participant.
2) Cloning. The adversary tries to provide multiple users the same provenance information they have obtained from an honest participant in the protocol.
3) Network Sniffing. The adversary replays legitimate provenance information it has seen on the network (possibly intended for a different service provider).
4) Exclusion. The adversary tries to provide information to a user from an honest participant in the protocol without providing provenance information.

The service providers are untrusted and may be active adversaries, attempting to create false provenance data, or modify and delete true provenance data. If a service provider is honest and correctly provides provenance information, then it should not be possible for an adversary to fabricate, clone, network sniff, or exclude the honest service providers provenance information. A service provider may also provide incorrect provenance information to try and discredit another service provider.

While a service provider should provide provenance information for every step in the chain to the user, if a set of service providers on the chain colludes together they can make it appear as though they are acting as one service provider. We do not consider this an attack on our protocol as the group is colluding and essentially acting as one party.

V. Provenance Chains

Each time information is created or used by a service provider a provenance tag is created and passed along with the result. The tag is created by the source of the provenance information and signed using the secret key of the source. The tag should contain enough information to recreate the provenance information as set out in the model in Section III.

A tag is a tuple: $tag = \{A, B, C, D, E\}_{sk_{source}}$. The definitions of the parameters in the tag are:

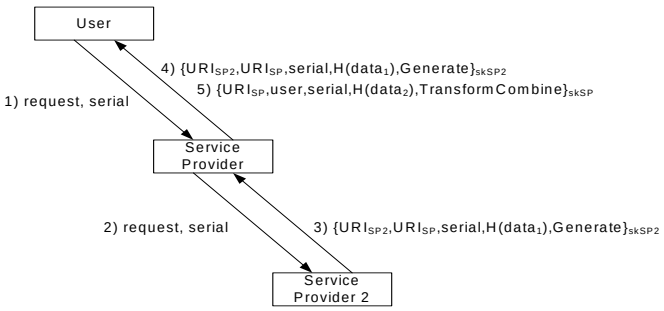

Figure 3. Provenance Chains

- $A = source$: the Uniform Resource Identifier (URI) of the source of this provenance tag.
- $B = destination$: the Uniform Resource Identifier (URI) of the destination of this provenance tag.
- $C = serial$: The randomly chosen serial number of this service request. This is created by the user and sent along with the request for service.
- $D = H(data)$: A hash of the service result that is associated with this provenance tag.
- $E = action$: What process was performed on this tag, either Generate or TransformCombine.

The tags are signed using the secret key of the source. We use the notation of $\{A\}_{sk_B}$ to denote the message A signed using the key sk_B. We assume that all public keys for service providers are well known or discoverable from a certificate authority.

Figure 3 shows the process for passing tags between a user and two service providers.

1) The user randomly generates a serial number for this request and sends the request for service to its service provider (Service Provider) along with the serial number. The serial number is to prevent service providers from being able to replay provenance information.
2) The service provider (Service Provider) generates the request for service from the second service provider (Service Provider 2) and sends it along with the serial number to the second service provider (Service Provider 2). The service provider (Service Provider) will send a request to every service provider it uses as input in this step, we have only shown one for clarity.
3) The second service provider (Service Provider 2) generates a provenance tag to return to the first service provider. The tag contains the source and destination, the serial number, a hash of the data returned, and the action taken (Generate). The tag is then signed using the secret key of the service provider sk_{SP2} and returned to the first service provider.
4) The service provider (Service Provider) sends the customer the tag from the second service provider.
5) The service provider creates a provenance tag to send

to the customer. The tag contains the source and destination, the serial number, a hash of the data to return to the customer, and the action (TransformCombine). The service provider then returns this tag to the customer.

The customer will check that all provenance information contains the correct serial number, a tag from all registered inputs, and an unbroken chain of provenance information.

VI. Analysis

We now analyse our protocol for providing provenance for web services compared to our requirements we have discussed in Section III and Section IV. We start by checking that the protocol meets completeness requirements and contains enough provenance information to recreate the provenance graph for a service. We then complete a security analysis to check that incorrect provenance information can be detected in the presence of malicious service providers.

A. Completeness

A protocol that has complete provenance information should allow the creation of a provenance graph from the provenance information provided by the protocol. To show completeness we consider the relationships shown in the open provenance model and how the provenance tags show these relationships.

- Artifact: identified by the hash value of the artifact in the provenance tag.
- Process: identified by the process name (either Generate or TransformCombine) in the provenance tag.
- Agent: identified by the source of the provenance tag.
- wascontrolledby(role): is represented by the source of the provenance tag and the role is identified by the process name.
- wasgeneratedby(role): is represented by the hash of the artifact and the role by the process name.
- usedby(role): is represented by the set of tags with the destination set as the controller for the operation of the role.
- wasderivedfrom: can be generated using the source and destination values of the provenance tags and the hash values of the artifacts. wasderivedfrom may be over approximated when there are multiple inputs and outputs for a service provider for a single service request. In this case, the tags show that the output artifacts were derived from the input artifacts, but not what individual input artifacts an output artifact was derived from.

B. Security Analysis

Service providers may provide incorrect provenance information or respond incorrectly to audit requests. We assume we are using a signature scheme that has provable security against existential forgeries under adaptive chosen message attacks in the random oracle model such as PSS RSA [11] or the triplet El-Gamal scheme [12]. If the signature scheme is secure against existential forgeries, then given a public key pk it is infeasible to forge a pair (m, σ) where σ is a valid signature on m using the secret key corresponding to the public key pk. We assume the serial numbers are chosen at random from a large set and that the chance of two users choosing the same serial number is negligible. We do not consider side channel attacks or the possibility of an adversary gaining access to the secret key of an honest participant.

1) Fabrication: If the adversary is able to construct provenance information from an honest participant in the protocol that they have never used then they must create a tag signed using the secret key for the participant. We can then use this adversary to break the signature scheme. The adversary is given as input the public key of the participant pk. The adversary will then produce a valid provenance tag $\{tag\}_{sk}$. We then have a valid message signature pair for pk, $(m = tag, \sigma = \{tag\}_{sk})$.

2) Cloning: If the adversary is able to clone provenance information from an honest participant then the adversary must modify a valid tag signed with a secret key with a new serial number. If the adversary can complete this modification, we can use the adversary to break the signature scheme. The adversary is given as input the public key of the honest participant and the tag to clone signed with the secret key, $\{tag_{original}\}_{sk}$. The adversary will then produce a new tag that has a different serial number denoted $\{tag_{new}\}_{sk}$. We then have a valid message signature pair for pk, $(m = tag_{new}, \sigma = \{tag_{new}\}_{sk})$.

3) Network Sniffing: If the adversary is able to use network sniffing to claim ownership of some provenance information then, similar to a cloning attack, the adversary will have to alter the tag to have the correct serial number. As shown in Section VI-B2 we can then use this adversary to break the signature scheme.

VII. Preventing Exclusion Attacks

To prevent exclusion attacks, a service provider registers the URIs of service providers it uses as input with a third party called a Registration Server which records these details and is queried by the customers to discover the provenance information it should be receiving from the service.

A. Service Provider Registration

Figure 4 shows the process used by a service provider to register their service with the registration server. The service provider submits a record with their URI URI_{SP},

Figure 4. Service Provider Registration

Figure 5. Customer Requesting Service Provider Data

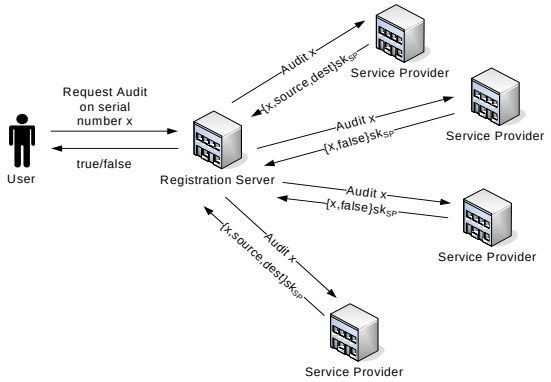

Figure 6. Auditing Registration Information

their public key pk_{SP} and the list of URIs of all service providers they use as inputs $URI_1, ..., URI_N$. The inputs that a service provider uses may also change over time. In this case the service provider should be re-register the service with the new inputs.

B. Customer Requesting Service Provider Data

Figure 5 shows a user requesting the data on a specific service provider from the registration server. The user will not need to perform this action every time it uses the service, it will only need to periodically check for changes to the inputs to the service provider.

C. Auditing Registration Information

A service provider may submit incorrect information by excluding some of the service providers it uses and hiding a source of data. We include a method for a user to request an audit for a particular query it has done. The user will submit the serial number it used in the request to the registration server which carries out checks to confirm the registration information provided by the service provider.

Figure 6 shows the process for requesting the audit. The registration server broadcasts an audit request with the serial number. Service providers then return a signed message containing either false and the serial number if they did not take part in the request, or the source and destination of the request if they did. These messages are put together by the registration server to check the information it has registered for a service provider.

A broadcast request has to be sent to all service providers, but we do not envisage the users requesting an audit for every service request. The registration server could also return a time stamp of the last audit time for the service provider with the information it returns to the user.

If we assume that the registration server is acting honestly, then the registration server will send the audit message to all service providers that have registered. All honest participants will have registered with the registration server and so will receive the audit message and respond to it. The registration server can then check the information that is returned by the honest participants to confirm the registration information that has been provided by the service provider.

If we assume that the registration server is not trusted and may act maliciously, then we need some mechanism to verify the actions of the registration server. We briefly discuss two possible mechanisms for implementing a verifiable registration server: a public bulletin board and the use of a group of registration servers. The registration server could publish all its actions to a public bulletin board. Customers could then check that the audits they requested have been completed and responders to the audit message can check that their response is correctly recorded. A second option is to use a group of registration servers where a threshold value of the group is required to sign values sent to the customer. As long as fewer than this threshold value of registration servers are acting maliciously the customer can be confident of receiving correct responses to its queries and audits.

A malicious web service can change the serial number during the service to carry out an exclusion attack. The user sends a web service request with a serial number $S1$ to a service provider. The service provider then creates a new web service request with a new serial number $S2$ to send to the service provider. When the audit request is made with the serial number $S1$, the service provider that received the request with the serial number $S2$ will return false.

To prevent this kind of exclusion attack, we need to prevent or discourage a service provider from being able to generate a serial number. We let the registration server generate the serial numbers, so an honest service provider will only provide the service and provenance information if the service request has a valid serial number signed by the registration server and users will need to apply to the registration server for valid serial numbers.

To discourage dishonest service providers from generating a serial number we can require a participant that requests a serial number to solve a computational challenge or captcha. A user will then need to solve this computational challenge for every service request they make. This punishes service providers that try to provide web services that exclude their input services by the loss of computation that they require to generate serial numbers. While users will also need complete the computational challenge, it is reasonable to assume that a service provider will have many customers and will need to contribute more computational time for an exclusion attack.

A second option would be to charge a participant that generates a serial number a micropayment. This will mean

that an honest user will need to pay a micropayment every time they request a service. Again it is reasonable to assume that a service provider will need to generate more serial numbers than the users resulting in a significant financial penalty. These micropayments can also be used to fund the resources required for the registration servers.

VIII. Performance

We examine both the computational and communication complexity of our protocol for providing provenance information for service providers. We show the upper bounds for the complexity. We let m denote the complexity of modular exponentiation and n denote the complexity of modular division. We denote a as the number of service providers. The value p is a public parameter for the signature scheme used and tag is the size of the provenance tag. In these tables we consider the complexity of creating and sending the provenance tags and not the auditing or checking of inputs for a service provider or any computational cost of generating a serial number.

	Service Provider	User	Total
Computational	$a(m+n)$	$3ma$	$a(4m+n)$
Communication	$a(tag+3p)$		$a^2(tag+3p)$

Table I
Complexity

Table I shows the computational and communication complexity of our protocol. Service providers have to construct one signature on their provenance tag per output value. If the service provider checks the provenance information provided to it it will have the combined complexity of the user and the service provider. This complexity scales linearly with the number of service providers.

For the communication complexity, we only consider the amount of data each party must send as part of the protocol. The user does not send any data. The communication complexity scales quadratically with the number of service providers. This is due to the service providers having to forward on not only their provenance information but also all previous provenance information from its inputs.

IX. Conclusion

In this paper we have constructed a protocol for providing provenance information for web services. As web services increase in popularity and importance for both personal and commercial use, customers will want to verify the provenance of data provided by web services. We have shown that our protocol prevents fabrication, cloning, and network sniffing attacks. We have also provided a detailed discussion about preventing exclusion attacks. We use a third party called a registration server to prevent exclusion attacks and we also suggest methods to verify the actions of this third party. We have also shown the computational and communication complexity of the protocol with the computational complexity growing linearly with the number of service providers, and the communication complexity growing quadratically with the number of service providers.

References

[1] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche, "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, July 2010. [Online]. Available: http://eprints.ecs.soton.ac.uk/21449/

[2] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Rec.*, vol. 34, pp. 31–36, September 2005.

[3] P. Groth, S. Miles, and L. Moreau, "A model of process documentation to determine provenance in mash-ups," *ACM Trans. Internet Technol.*, vol. 9, pp. 3:1–3:31, February 2009.

[4] V. Tan, P. T. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security issues in a soa-based provenance system," in *IPAW*, 2006, pp. 203–211.

[5] U. Braun, A. Shinnar, and M. Seltzer, "Securing provenance," in *The 3rd USENIX Workshop on Hot Topics in Security*, ser. USENIX HotSec. Berkeley, CA, USA: USENIX Association, July 2008, pp. 1–5.

[6] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang, "Scientific workflow provenance querying with security views," in *Web-Age Information Management, International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 349–356.

[7] P. Groth and L. Moreau, "Representing distributed systems using the open provenance model," *Future Gener. Comput. Syst.*, vol. 27, pp. 757–765, June 2011.

[8] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: preventing history forgery with secure provenance," in *Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–14.

[9] ——, "Preventing history forgery with secure provenance," *Trans. Storage*, vol. 5, pp. 12:1–12:43, December 2009.

[10] J. Zhang, A. Chapman, and K. Lefevre, "Do you know where your data's been? — tamper-evident database provenance," in *Proceedings of the 6th VLDB Workshop on Secure Data Management*, ser. SDM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 17–32.

[11] M. Bellare and P. Rogaway, "The exact security of digital signatures-how to sign with rsa and rabin," in *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques*, ser. EUROCRYPT'96. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 399–416.

[12] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *EUROCRYPT '96: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*. Springer-Verlag, 1996, pp. 387–398.