A Proposal and Evaluation of User Centric Trusted Log Archival Architecture

Takashi Shitamichi,
Tokyo Denki University, Japan
5 Senjyu-Asahicho Adachi-ku, Tokyo, 120-8551 Japan
shitamichi@isl.im.dendai.ac.jp

Ryoichi Sasaki,
Tokyo Denki University, Japan
5 Senjyu-Asahicho Adachi-ku, Tokyo, 120-8551 Japan
sasaki@im.dendai.ac.jp

ABSTRACT

Internet services of numerous types are widely implemented at countless sites in today’s worldwide computing environment, which is on-premises or cloud computing, and the generated system and application service logs they produce are important for assuring such systems work correctly. When the owners of such logs are auditors or system managers, it has been thought that from the standpoint of manageability, it is better to accumulate logs at one site rather than multiple sites. However, when the owner of a log generated by an application service is a system user, he or she might want to express a preference from the available log archival sites. Furthermore, there are often cases when a service site is located far away from the log archival site. It should also be mentioned that if sites providing services do so in a cloud computing environment, it is particularly necessary to use a secure and fast messaging method between the service and log archival sites. In this paper, we define a “user centric log archival architecture” concept, examine related works and technical specifications, and propose a new trusted model via both abstract and practical methods. By extending the Simple Object Access protocol (SOAP) based Security Assertion Markup Language (SAML), and using SAML assertions, we show how log messages can be exchanged with confidentiality, integrity, and availability, before they are written securely to storage devices. In order to verify the effectiveness of the proposed architecture, the latencies of a XML messages that contain a SAML assertion and a XML signature are measured and considered in a cloud computing environment.

KEYWORDS

Log, Security, Federated identity, SAML, Cloud computing

1 INTRODUCTION

The number of services that are deployed in cloud computing environment has been increasing in recent years. Such services are not only provided in single clouds, but also in multiple ones in order to federate the services that exchange numerous kinds of information such as personal identities on social networking services (SNS). Along with the wide deployment of services in the cloud, information security on such services has garnered significant amounts of increased attention because of recent incidents of identity fraud and related theft.

As countermeasures for security incidents, numerous studies have been conducted and technical solutions have been devised for both channel and message security. In unsecure environments, a log is very important for assuring that a system runs correctly and provides the agreed upon services. Generally speaking, it is thought that auditors or system managers use such logs for information system audits, and it is true that “system logs”, which are generated by one or more infrastructure systems that consist of the network, hardware,
software, middleware, etc., should be examined to ensure their confidentiality, integrity, and availability.

Meanwhile, “application log” clusters, which are generated by one or more application programs on provided services, are also generated and maintained by service sites. An application log is generated by the events that occur when a user uses the services. Therefore, it is reasonable that a user should have the option of selecting where the archival log is to be stored.

Furthermore, it is possible that each service that a user can select for use is based in a different region of the world. However, some users may want to store their logs at sites that are located in the country where they live.

Below, a number of examples are provided:

- Service-A: Shopping site in United States
- Service-B: Health service in France
- Service-C: SNS service in Malaysia
- Log Archival-X: located in United States
- Log Archival-Y: located in (rough country Y)
- Log Archival-Z: located in Japan
- User: lives in Japan

If the abovementioned user wants to preserve his or her personal information securely, there is a high probability that he or she would like to select the Log Archival-Z, which is located in Japan. Figure 1 shows the selectable services and log archival sites.

Currently, numerous services internally own and maintain their log systems, while users must use each service as shown in Figure 2. As can be seen in the figure, if a user desires to control all of his/her logs, he/she has to examine each log at each site individually. However, if a user wants to control and preserve all of the logs in a more manageable fashion, they should be concentrated onto a log archive at a single site, such as can be seen in Figure 3.

In identity management field studies, allowing a user to control his/her own identity, which consists of multiple attributes, is called “user centric identity” [1][2]. In this paper, we define the concept by which a user can select a preferred log archival site as “user centric log archival architecture”. In addition to the merit of manageability, if a user is concerned about privacy in cloud computing environments [3], he or she may desire to ensure all logs are securely stored in a single site that he or she selects. Although centralized logs were examined in various studies previously [4], there have been no studies that describe a centralized log storage system with the concept of the user-centric.

A user uses Services A, B, and C, but cannot control where the logs are stored

![Figure 1: Selectable log archival site](image1)

A user can select to use Service-A, B and C, as well as select where to store the logs

![Figure 2: Each service site owns its log archives](image2)

A user uses Services A, B, and C and wants to control where the logs are stored

![Figure 3: Logs are gathered and stored at one site](image3)
2 RELATED WORKS

2.1 Securing Logs on Untrusted Machines

In their paper, Bruce Schneier and John Kelsey described a computationally cheap method for making all log entries generated prior to the logging machine’s compromise impossible for attackers to read, as well as impossible to modify or destroy undetectably [5]. Their premise is that log entries are generated at untrusted machines, \( U_0, U_1, \ldots, U_n \), and sent to a trusted machine \( T \). Therefore, auditors may be able to perform information system auditing tasks completely and efficiently if they must only deal with the machine \( T \), which holds all of the \( U_0, U_1, \ldots, U_n \) log entries.

However, there is no description of \( U \) and \( T \) in the distributed system environment and it is considered likely that further requirements will be forthcoming in the cloud environment in the future. In particular, the complex distributed Web services that are deployed across multiple domains in cloud computing environments will make it very difficult for auditors to locate the log entries that are provided by each service site. To solve this problem, a system needs to satisfy the following requirements:

- **Centralized logs** - The archives of the log entries must be aggregated and operated securely at one site alone, even among the multiple domains of the cloud computing environment.
- **Secure log messaging** - All log entries generated by each service site across multiple domains must be sent securely to the archive site.
- **Fast log messaging** – If the service site is located at the far distant place from the archive site, the elapsed time of messaging is longer than the one between the short distance places. Fast processing of messaging in the both sides is required.

2.2 SAML Assertion

The Security Assertion Markup Language (SAML) [6] was developed in the early 2000s as a framework for exchanging security information using the Simple Object Access protocol (SOAP) mechanism through a network, and has since been widely deployed in numerous Internet and intranet services. The most famous usage, which is called use case, of SAML is single sign-on (SSO). In a cloud computing environment, numerous actual enterprise information systems such as Google Apps [7] and Salesforce.com’s Force.com [8] use SAML for federated identity. In addition, the number of studies using SAML such as identity federation service for consumers [9] and privilege federation [10] has been increasing as well.

In terms of software architecture, SAML has two types of system entities known as the identity provider (IdP) and service provider (SP).

An IdP is defined a provider type that creates, maintains, and manages identity information for principals while also providing principal authentication to other service providers within a federation, such as with Web browser profiles.

In contrast, a SP is a role donned by a system entity where the system entity provides services to principals or other system entities.

In order to enable the SAML SSO, the IdP provides a SAML assertion to the SP, which is an information set that supplies one or more statements made by an authority. An assertion consists of one or more statements, which are one of the three different types defined below [11]:

**Authentication:**

The specified subject is authenticated by a particular means at a particular time. This statement is typically generated by the IdP.

**Attributes:**

The specified subject is associated with the supplied attributes.
**Authorization decision:**
A request to allow the specified subject to access the specified resource has been granted or denied.

For an SSO, a typical SAML assertion will contain a single authentication statement and possibly a single attribute statement. The structure of a SAML assertion and its associated statements are shown in Figure 4.

![Figure 4: SAML assertion structure and statements](image)

The use of a SAML assertion enables SSO among one or more SPs and an IdP. Each site independently owns its user accounts, and each account for the same user in both sites is associated with an identifier that is called an “opaque handle”. This technology enables each site to independently maintain the user identity and protect the user’s privacy because no common identity information is shared. In the architecture example shown in Figure 5, two sites enable SAML SSO. One of them is the IdP and the others are the SPs.

![Figure 5: SAML SSO using opaque handle](image)

2.3 Cloud Log Archiver

In order to meet the requirements of “Centralized logs” and “Secure log messaging”, Takashi Shitamichi and Ryoichi Sasaki introduced the “Cloud Log Archiver” (CLA) architecture that extends SAML and the Identity Web Services Framework (ID-WSF) specifications [12]. In this architecture, individual sites do not retain any log entries that are locally generated by the application service programs running in the site. Instead of writing them on a local storage device, each site sends the SOAP messages of the log entries, which include the SAML assertion, to the CLA that holds all of the log entries. The CLA then verifies the sender’s authentication contexts with an extensible markup language (XML) signature, and acknowledges the sender with the public key.

The structure of the SOAP message, which is called a SOAP Envelope that consists of a SOAP header and a SOAP body, is drawn in Figure 6. The SAML assertion is described in the element of <Assertion> in the element of <Security> in the header of the SOAP Envelope. Another element in <Security> is the <Signature> that consists of the XML Signature for the SOAP header and the SOAP body. The combination of the SAML assertion and the XML signature for the SOAP header and the SOAP body prevent manipulation and assure secure messaging between the service provider and the archival sites.

The CLA architecture allows reliable logs to be successfully gathered from various cloud sites, but it depends on ID-WSF [13], which is an extremely complicated specification. Therefore, it is not easy to implement due to its complexity. Furthermore, the CLA architecture does not clearly define the trusted relations between the service provider and log archival sites.
As mentioned in Section 2, the previous studies and the original technical specifications are unable to sufficiently substantiate user-centric log archival architecture in terms of trust in order to meet the following requirements:

**Requirement (A): Trust relations**
All of the system entities, which include the service site, log archival sites and a user, must be in trusted relationships.

**Requirement (B): User centric**
Users must be able to select a log archival site, which then gather and store numerous logs from one or more service sites, in the trusted relations.

**Requirement (C): Secure messaging**
Any and all log records can be sent from a service site to a log archival site with confidentiality, integrity, and manageability.

**Requirement (D): Ease of implementation**
The architecture is easy to implement without using complex technology.

**Requirement (F): Fast messaging**
The architecture provides fast processing for secure messaging in both a service site and an archival site.

In this section, we propose both abstract and practical models that demonstrate how the above requirements can be met.

### 3.1 Abstract Models

#### 3.1.1 Considerations of System Entities and Principal
A system entity, $L_g$, provides a service to the principal $P$ and generates log records. A system entity, $L_a$, collects, archives and stores the log records.

When $P$ wants to use the service at $L_g$ and store the log records at $L_a$, $P$ should assure the log records created by $L_g$ for $L_a$. At this point, $P$ trusts both $L_g$ and $L_a$, whereas $L_g$ may not trust $L_a$. This means that $P$ has trusted relations with $L_g$ and $L_a$ individually, but that $L_g$ has no trusted relations with $L_a$.

Figure 7 shows the relationship between $P$, $L_g$ and $L_a$.

In order to make $L_a$ trust $L_g$, another system entity is necessary to facilitate trusted relations between the two. Therefore, we add another system entity called $L_i$, which establishes trusted relations between $L_g$ and $L_a$. Using this method, $L_a$ can trust $L_g$ as a system entity.
Figure 8 shows the trust relationship between \( P, Lg, La \) and \( Li \).

![Figure 8: Trusted relationship with \( Li \) added](image)

### 3.1.2 Circle of Trust

Once \( Li \) provides surety regarding trusted relations between \( Lg \) and \( La \), and the identity verification of \( P \), the three systems entities and \( P \) establish a complete set of trusted relations, which is called a “Circle of Trust” (CoT). Within a CoT, each system entity and participant may communicate with each other in trust. Figure 9 shows the CoT that consists of \( Lg, La, Li \) and \( P \).

![Figure 9: Circle of Trust with three system entities and one participant](image)

### 3.2 Practical Models

As \( Lg \) may be located far distant from \( La \), a secure messaging mechanism is necessary. SOAP with SAML can provide a practical solution for establishing a CoT and secure message exchange between system entities and participants.

#### 3.2.1 Preparation of CoT

Prior to the commencement of services at \( Lg \), a couple of SAML metadata, which state the information of each system entity, are exchanged between \( Lg \) and \( Li \), and between \( La \) and \( Li \). This result establishes a CoT for \( Lg, La \), and \( Li \).

#### 3.2.2 Declaration of selecting a log archival based on \( P \)'s will

In order to enable a service at \( Lg \) for \( P \), and in order to use the log archival mechanism with \( Lg \) and \( La \), each entity and \( P \) must complete the following steps and follow the message flow shown in Figure 10.

1. \( P \) tries to sign-on to start using a service at \( Lg \). \((t1, t2)\)
2. \( Lg \) redirects \( P \) to \( Li \), which then provides authentication for \( Lg \). \((t3, t4, t5, t6)\)
3. Once \( P \) successfully completes an \( Li \) sign-on, a SAML assertion is returned to \( P \) and sent back to \( Lg \). \((t7, t8, t9, t10)\)
4. \( Lg \) returns a success status to \( P \) and shows one or more \( La \)s that are user-selectable log archivals. \((t11, t12)\)
5. \( P \) selects one of the \( La \)s. \((t13, t14)\)
6. \( Lg \) redirects \( P \) to the \( La \). \((t15, t16, t17, t18)\)
7. \( La \) redirects \( P \) to \( Li \), which provides authentication for \( La \). \((t19, t20, t21, t22)\)
8. Once \( P \) successfully completes an \( Li \) sign-on, a SAML assertion including a bootstrap, which has another assertion for \( La \) at \( Li \), is returned to \( P \) and sent back to \( La \). \((t23, t24, t25, t26)\)
9. \( La \) redirects \( P \) to \( Lg \) to pull the bootstrap which has an assertion for \( La \). \((t27, t28, t29, t30)\)
10. \( Lg \) returns the success status to \( P \). \((t31, t32)\)

Once those steps are completed successfully, \( P \) joins and establishes a CoT that consists of \( P, Lg, La \) and \( Li \).
3.2.3 Service usage commencement

Now that \(Lg\) has a SAML assertion for \(La\), \(Lg\) can commence services by sending a reliable log message along with the assertion. Figure 11 shows the entity transition diagram.

1. \(P\) starts using a service at \(Lg\). \((p_1)\)
2. \(Lg\) verifies a SAML assertion for \(La\). \((v_1)\)
3. If the assertion is not verified (e.g. due to the expiration date), \(Lg\) requires \(Li\) to send a new assertion for \(La\). \((a_1)\)
4. \(Li\) creates the new assertion for \(La\) and sends it back to \(Lg\). \((a_2)\)
5. \(Lg\) generates a log record with an assertion in a buffer and sends it to \(La\). Figure 12 shows the message, which consists of the assertion and the log record. \((s_1)\)
6. \(La\) receives the log message and verifies the assertion. \((v_2)\)
7. If the assertion is valid, \(La\) writes the log record to a local storage device. \((w_1)\) If the assertion is invalid, \(La\) returns an error status to \(Lg\).

4 EXPERIMENT

The previous section describes the trusted log archival architecture that uses a SAML
assertion to send securely a message from a service site (Lg) to an archival site (La). In this architecture, the processing of signing and verification for XML signature uses a lot of computer resources and time in addition to latencies between Lg and La. Therefore, the elapsed time of signing, transferring and verification were measured under real-world conditions.

4.1 Configurations

Experiments were conducted to measure the performance of a XML signature and transferring of data to determine whether signing, transferring and verification could be performed quickly enough in a cloud computing environment. For the experiments, the Amazon Elastic Computing Cloud (EC2) of Amazon Web Services (AWS) was selected as the Infrastructure, as a Service (IaaS) for providing virtual instances in nine regions in the world as of December 2014.

The specifications of the AWS EC2 used in the experiments are shown in Table 1.

<table>
<thead>
<tr>
<th>Region : Lg</th>
<th>Virginia, Oregon, California, Ireland, Frankfurt, Singapore, Sydney, Sao Paulo, Tokyo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: La</td>
<td>Tokyo</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 6.5 (64bit)</td>
</tr>
<tr>
<td>Instance Type</td>
<td>m3.medium (vCPU:1, ECU:3, Memory: 3.75GB, Instance storage: 4xSSD)</td>
</tr>
</tbody>
</table>

4.2 RTT between different Regions

As the first experiment, the round trip time (RTT) between Lg and La were measured to examine real latencies in the real cloud computing environment over the world. As each VM instance for each region was set up to receive ICMP packets, La that was run in the Tokyo region sent the packets to the nine Lg. All of RTT were measured by ping and traceroute commands that are Linux standard programs. The results are shown in Table 2. The summary of the result is as follows.

- The destination with the maximum RTT from La was Sao Paulo and almost 4 times longer than the RTT from Singapore.
- The average RTT within the same Tokyo region was only 0.43 although the hop count is 11.
- The maximum hop count is 26 for Virginia. It was more than other two regions in the US. It is considered that packets were transferred across North America.

<table>
<thead>
<tr>
<th></th>
<th>Min(ms)</th>
<th>Ave(ms)</th>
<th>Max(ms)</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia</td>
<td>202.54</td>
<td>202.92</td>
<td>203.14</td>
<td>26</td>
</tr>
<tr>
<td>Oregon</td>
<td>124.89</td>
<td>125.04</td>
<td>125.25</td>
<td>19</td>
</tr>
<tr>
<td>California</td>
<td>122.26</td>
<td>122.57</td>
<td>122.80</td>
<td>19</td>
</tr>
<tr>
<td>Ireland</td>
<td>273.12</td>
<td>273.47</td>
<td>273.74</td>
<td>24</td>
</tr>
<tr>
<td>Frankfurt</td>
<td>278.99</td>
<td>279.26</td>
<td>279.94</td>
<td>20</td>
</tr>
<tr>
<td>Singapore</td>
<td>75.35</td>
<td>75.63</td>
<td>75.93</td>
<td>18</td>
</tr>
<tr>
<td>Sydney</td>
<td>118.52</td>
<td>118.79</td>
<td>119.20</td>
<td>17</td>
</tr>
<tr>
<td>Tokyo</td>
<td>0.30</td>
<td>0.43</td>
<td>4.97</td>
<td>11</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>300.57</td>
<td>301.20</td>
<td>301.85</td>
<td>21</td>
</tr>
</tbody>
</table>

4.3 Signing, Verification and RTT

In order to prove the effectiveness of the proposed architecture, it should represent that the fast messaging is possible between Lg and La even if two sites are located in the far distant places. While the latencies of RTT were shown in the previous section 4.2, elapse time of internal processing in each site should be examined. As the configuration consists of fast CPU and SSD that is much faster than HDD, most of the application program logics are processed very fast. However, the proposed architecture uses the SOAP message that consists of a SAML assertion and a digital signature, therefore, the processing time of a signing of the SOAP header and body at Lg and a verification of the signature at La is the critical factor although it is generally considered that signing uses a lot of computer resources and time.
In order to examine the processing time for signing and verification, a couple of experimental programs at \( Lg \) and \( La \) were written as follows.

- All of the programs are written in C language.
- In order to handle XML signature, XML Security Library, which is a C library based on LibXML2, was used for signings and verifications.
- As a public-key cryptosystem, RSA 2048 is used. The private key is encrypted with AES 256, then installed onto \( Lg \). The public key is done onto \( La \).
- \( LgP \) is a program at \( Lg \). \( LgP \) creates a log record in eight different sizes of 64KB, 128KB, 192KB, 256KB, 320KB, 384KB, 448KB, and 512KB.
- \( LgP \) signs each log record and creates a SOAP header, which includes SAML assertion, and body, then sends it to \( La \) through the dedicated port and the established TCP connection.
- \( LaP \) is a program at \( La \). \( LaP \) receives a message that is eight different length from \( Lg \).
- \( LaP \) verifies the signature of the message and writes the log data into a storage (SSD).
- The experiments of signing were held in each five region with 50 times of trial. As all of the instances were equally configured, the average of signings at the all of the \( Lg \) was calculated.
• The experiments of verification were held at \( La \) in Tokyo with 50 times of trial. The average of them was calculated.

\( LgP \) was run at each eight \( Lg \) and the elapsed time of signing and transferring was measured. \( LaP \) was run at \( La \) in Tokyo region and the elapsed time of verification was measured. The result of the experiment is shown in Table 3 and Table 4. RTT, which includes signing, transferring and verification is shown in Figure 13. The summary of the result is as follows.

• The processing time of signing is 3-5 times longer than verification in the case of using RSA 2048 with AES-256.
• The maximum time of signing was only 14 ms even for 512KB data.
• In contrast, transferring time was much longer than internal processing for signing and verification.
• The maximum RTT was measured for Frankfurt region. In case of the same region where both \( Lg \) and \( La \) were in Tokyo, RTT was quite short even for transferring 512KB data.

4.4 Consideration of the Experiment

Experiments were successfully conducted to measure the performance of signing and verification for XML signature and transferring of a log record using proposed architecture. The speed of data communication has been faster than before even if it is cloud computing environment, however, Table 2 in the section 4.2 shows that there are still big latencies between the far distant places. In contrast, processing time of signing and verification of a log record is much shorter than transferring time. Therefore, usage of a SAML assertion that includes XML signature does not impact on RTT but effective to send a log record securely.

5 EVALUATIONS

As mentioned in the beginning of Section 3, there are five requirements for the architecture. Evaluations for each requirement are provided below:

• Requirement (A): Trust relations
  CoT creates relationships of trust between a service site that generates a log, a log archival site, a user as a principal, and the IdP. Thus, requirement (A) is met.

• Requirement (B): User centric
  After a user signs onto a service site, he or she is allowed to select a preferred log archival site. Thus, requirement (B) is met.

• Requirement (C): Secure messaging
  Using a SAML assertion, messaging between a service site and a log archival site are handled with confidentiality, integrity, and manageability. The architecture is based on SOAP, therefore, network issue is expected to solve by the SOAP mechanism such as WS-Reliability. Thus, requirement (C) is met.

• Requirement (D): Deployment ease
  Instead of using complex specifications such as ID-WSF, the architecture can be implemented easily with SAML and its extensions. If the log data is encoded to readable format such as base 64, any log can be sent. Thus, requirement (D) is met.

• Requirement (F): Fast messaging
  The experiment of the proposed architecture shows that the processing time of signing and verification even for a log record is not considerable. The proposed method does not impact on fast messaging.

As all of the evaluations above show, our proposal meets all of the provided requirements, and it can be expected that our user centric trusted log archival architecture will prove to be very useful in the future.

6 CONCLUSIONS

In this paper, we proposed a new user centric trusted log archival architecture, along with
both abstract and practical models, that demonstrate how log messages can be sent from a log generator to a log archival site while being maintained with confidentiality, integrity, and availability using SAML. We also provided figures that show trust relations among system entities, message flows, and transitions.

Our successful experiment demonstrated that the proposed architecture can be implemented in actual cloud environments at sufficient performance levels.

Since our proposed architecture allows users to select log archival services that meet their needs, we believe our user centric trusted log archival architecture will also further facilitate the establishment of service providers in the cloud computing environment.

REFERENCES


[13] Liberty Alliance, Liberty Alliance ID-WSF 2.0 Specifications including Errata v1.0 Updates, Available at http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_wsf_2_0_specifications_including_errata_v1_0_updates?f=resource_center/specifications/liberty_alliance_id_wsf_2_0_specifications_including_errata_v1_0_updates.