A Design of Usable and Secure Access-Control APIs for Mashup Applications

Ryota Hashimoto  
NTT Information Sharing Platform Laboratories  
3-9-11 Midori-cho, Musashino-shi  
Tokyo 180-8585, Japan  
+81-422-59-3950  
hashimoto.ryota@lab.ntt.co.jp

Nachi Ueno  
NTT Information Sharing Platform Laboratories  
3-9-11 Midori-cho, Musashino-shi  
Tokyo 180-8585, Japan  
+81-422-59-3591  
ueno.nachi@lab.ntt.co.jp

Michio Shimomura  
NTT Information Sharing Platform Laboratories  
3-9-11 Midori-cho, Musashino-shi  
Tokyo 180-8585, Japan  
+81-422-59-5750  
shimomura.michio@lab.ntt.co.jp

Figure 1 Example of mashup
(http://www.housingmaps.com/)

ABSTRACT
Mashups, which are applications that are developed rapidly by combining multiple Web applications, are currently gathering much attention. One issue arising when creating mashups using data that is subject to access control is the difficulty in adding authentication and access control functions without losing the advantage of rapid development. We discuss the design and prototype implementation of an access control platform called SAXAE. SAXAE supports more secure and easy development of mashups. Its API enables both more secure and easier to use data protected by access control protocols.

Categories and Subject Descriptors
D.2.0 [General]: Protection mechanisms, K.6.5 [Security and Protection]: Authentication, H.3.5 [Online Information Services]: Web-based services

General Terms
Design, Security

Keywords
Mashup, web services, authentication, authorization, access control.

1. INTRODUCTION
Development of “mashups” [1], Web applications that contain a combination of data from multiple Web applications from different providers, is rapidly becoming widespread. An example of a mashup that combines imagery from a mapping service with housing information from a regional advertising service is shown in Fig. 1.

One of the main advantages of mashup development is high efficiency. There are other technologies, such as service-oriented architecture [2] or Web services collaboration [3], but these have complex system architectures compared to mashups, meaning higher costs for developer education and development itself [4]. In contrast, the costs of these are low for mashups because mashups use basic Web technologies (HTTP, URI, JavaScript, etc.).

The low efficiency in development of mashups using non-public data (protected by access control measures) is an issue. Data used in mashups can be divided into public and non-public. Examples of public data include stock prices, weather information, and news. Examples of non-public information include personal information held by social networking services or business information held within corporate systems.

The cost of using an access control protocol to retrieve non-public data reduces the development efficiency of mashups. The cost includes that of learning protocols and their implementations and obtaining knowledge in a wide range of areas, such as web security, digital signatures, and encryption. The learning cost is even higher if multiple access control protocols are involved. Even after the developer understands the protocols, integrating them more securely into an application is also expensive, as is maintenance of the application once it is complete. An example is the efforts to address vulnerabilities found in the implementation of the protocols.

We propose the SAXAE access control platform, which raises the efficiency of developing mashups that use non-public data. The secure and easy-to-use API design is a special feature of SAXAE. Through its use, mashup developers can use non-public data with the same level of ease as using APIs for public data.

In Section 2, we discuss mashup models considered in this research. In Section 3, we discuss the design of the SAXAE access-control platform. We list related research in Section 4 and present the conclusion in Section 5.

2. Access control models for mashups
Here we present various models used for access control in mashups.
First, we define some terms used in this paper.

- **Authentication.** The process of checking whether a user is authorized to use a given service.
- **Authorization.** The process of checking whether to allow a particular operation on a given service.
- **Policy.** Information, including authentication and authorization settings.
- **Access control.** The process of controlling access to a service, from authentication to authorization.
- **Resource.** Data obtained from an external Web application that is combined to form the mashup.
- **Mashup Source (MS).** A Web application providing a resource.
- **Mashup Program (MP).** A Web application that provides a new function by combining resources.
- **Authentication Server (IdP).** A server with a function to authenticate a user and provide the result of the authentication to another service.
- **Service Provider (SP).** A server providing services other than those of an IdP.
- **Access control platform (PF).** A platform supporting access control for mashups.
- **User (U).** Someone using an MP through a browser and with authorization to retrieve the resources used by the MP.
- **ID.** An identifier that uniquely identifies a user.

Authentication models for mashups can be categorized based on the ID relationships between the MS and MP. Hereafter, we define "X → Y" as "Authorize X to pass data to Y".

- **[Auth1] The MS and MP have separate authentication mechanisms**
  
  In this model, the MP provides an ID to the user and the MS and MP are different. There is no mapping of IDs between MS and MP.

- **[Auth2] The MP acts as the IdP**
  
  In this model, the MP provides an ID to the MS.

- **[Auth3] The MS acts as the IdP**
  
  In this model, the MS provides an ID to the MP.

- **[Auth4] Both MS and MP are SPs**
  
  In this model, a third-party IdP provides ID(s) to the MS and MP. For example, the MP could act as the IdP. Authorization models for mashups that include an intermediate platform can be categorized as follows.

3. **SAXAE Access Control Platform**

We introduce the design and implementation of the SAXAE access control platform in the following.

3.1 **SAXAE Platform**

SAXAE provides an access control function for mashups. Web application developers can use the functions provided by SAXAE to develop mashups that make use of non-public data securely and easily.

The architecture of SAXAE is shown in Fig. 2. SAXAE has functions for authentication, authorization, and policy management.

- **Authentication function** This authenticates users of the MP and passes results to the MP.
- **Authorization function** This delivers authorization results and configuration data when an MP attempts to access a resource.
Policy management function Through its authentication and authorization policy settings, SAXAE can be used to implement these functions according to various access control models. The authentication policy settings include the following:

- SAML [5] IdP Server metadata (identifiers, supported bindings, endpoints, certificates, keys, etc.)
- SAML SP Server metadata (as above)
- Configuration data such as public keys for OpenID OP servers.

The authorization policy settings include the following:

- Configuration data for OAuth [6] SP Servers (URLs for each function provided, signature scheme, IDs submitted by the server for each application, secret values).
- IDs and secret values submitted by the OAuth SP server to each application.

3.2 SAXAE API

The SAXAE API provides functions to the mashup to retrieve protected resources, as listed in Table 1. In addition to providing authentication and authorization functions directly, the API also provides functions that use them to retrieve protected resources easily and securely. We now describe the SAXAE API with an example using safe_get, which is the latter type of function, to retrieve a protected resource. For such a purpose, configuration data including the resource URL, authentication, and authorization protocols for the server or application being used are required. Conventionally, these have been managed within each application, but with SAXAE, they are managed in the platform and resources can be obtained with safe_get by specifying a single URL.

The SAXAE API is currently implemented as a library. It is written in the PHP language, which is widely used for Web applications.

Message sequence The message sequence in a normal case is shown in Fig. 3. When a Web application calls safe_get ([A] in Fig. 3), the SAXAE PF checks the user's authentication state ([B]). If the authentication is successful, the SAXAE PF reads the configuration data for the resource corresponding to the URL given ([C]), requests authorization for the user according to the OAuth authorization procedure ([D]), and then obtains the resource from the OAuth SP. The SAXAE PF passes the retrieved resource to the Web application as the return value of safe_get ([E]).

The authorization process can fail if there is request congestion resulting in competition for resources. This can often happen when a Web application that includes several other Web applications makes several requests for resources that are received by a single SP. For example, it could happen when several web applications attempt to access a user's SNS contact list at the same time. To avoid this situation, the timing of requests must be controlled.

4. Related Works

Here we discuss related research. There have been no other proposals for methods supporting development of mashups handling non-public data that support multiple protocols for

![Figure 3 Messaging sequence between mashup application, SAXAE platform, and external servers (SAML IdP, OAuth SP)](image)

<table>
<thead>
<tr>
<th>Function</th>
<th>API</th>
<th>Description</th>
<th>Parameters</th>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain protected resource</td>
<td>safe_get</td>
<td>Obtain a protected resource based on SAXAE Platform settings and user authentication state.</td>
<td>URL for protected resource</td>
<td>Protected resource</td>
</tr>
<tr>
<td>Start a session</td>
<td>saxae_session_start</td>
<td>Generate a new session, or retrieve a session identifier from a cookie.</td>
<td>None</td>
<td>Success or failure from starting the session</td>
</tr>
</tbody>
</table>
authentication and authorization. Below, we summarize research related to secure mashups according to various elements in a mashup.

**Measures taken on mashup server** There has been one proposal for using policies on the mashup server for data users. Warner et al. have proposed an architecture that preserves privacy based on referral to a privacy policy and provides data only to users conforming to the policy [7]. With this approach, rights for obtaining data are centralized on the mashup server.

**Measures taken in mashup program** One method involving measures in the mashup program uses session keys generated by the program. Tyler has proposed a method using a session key generated at the client to protect a WebAPI [8]. This session key is used on the server side for authorization. A feature of this method is that the key is included in the URL, so it has good compatibility with RESTful Web applications [9]. Another method is often used when data needs to be exchanged securely among multiple mashup programs [10] [11]. This is because the data handled by each application can be separated, but a communications path for secure data exchange can also be established.

**Measures taken at mashup sources** Measures taken at mashup sources are implemented by applying the existing specifications of their various authorization functions. There are several different authentication protocols and specifications that can be used to control access at the mashup source [5, 12]. Liberty Alliance ID-WSF [13] is a specification that defines data models and processes for secure distribution of user attributes on Web services. This can be used to implement data retrieval with access control. OAuth is a protocol that regulates authorization using data output from a WebAPI. WS-Authorization [14] is a specification that regulates assignment of rights to data output from a Web service. There are also several proprietary specifications for WebAPI authentication and authorization from service providers [15] [16] [17].

Even making use of the above related research, it is not possible to increase both the security and efficiency of development of mashups that use non-public data. This is because each of the proposals assumes that a given access control structure will be used. If multiple MS’s each use different access control structures, the MP will have to handle each of these different approaches, leading to trade-offs between security and development efficiency. Because of this, we have proposed an access control infrastructure that establishes both security and development efficiency and have presented an evaluation of the proposal.

5. Conclusion
We designed and implemented a prototype of the SAXAE platform and its API, which supports the secure and easy use of protected resources. As was our objective, this API helps us to implement mashups that retrieve these resources. In the future, we plan to conduct an API user test with several developers developing sample applications in order to improve on the evaluation standards and utility of the API.

6. REFERENCES