# Secure Delegation for Web 2.0 and Mashups

*A Position Statement for the*
*2008 Workshop on Web 2.0 Security And Privacy*

P.Austel S. Bhola, S. Chari, L. Koved, M. McIntosh, M. Steiner, S. Weber
*IBM T.J. Watson Research Center*
*P.O. Box 704, Yorktown Heights, New York 10570*
*{pka, sbhola, schari, msteiner}@us.ibm.com, {koved, samweber}@watson.ibm.com*

## Abstract

Service providers are letting users completely control and use data through proxies, such as Web. 2.0 mashups. This trend is bringing renewed interest in the problem of secure distributed delegation. We highlight and discuss a number of new challenges for service providers and developers in creating a secure and usable delegation framework:

- Wire protocols securely authenticating the consumer without compromising the consumer's credentials
- Usability of delegation protocols for non-trivial mashups
- Protection from phishing and related attacks
- Access control based on fine grained delegation of access rights specified by users
- Enabling legacy systems, including existing RSS and ATOM servers, to support limited delegation, and
- A web programming model that supports a usable delegation model.

## Introduction

Mashups are one of the pillars of Web 2.0 technology. They take existing data/services and "mix" them to create new data/services. A classic example, Zillow.com, combines real estate tax assessments, tax equalization rates, and maps in order to plot estimated home values throughout the United States. Flikrvision.com takes photos submitted to flickr.com and overlays them on a map of the world, showing the photograph and the location of the photographer. While they are relatively simple, they show some of the expressive power of mashup technologies.

Mashups, by their very definition, involve a man-in-the-middle. While Web Services[1] using SOAP[2] as a transport can provide end-to-end security services, typical Web 2.0 applications use the simpler REST-based communication approach[3] that lacks a rich security infrastructure to draw upon. SOAP based messages include security headers which embed or reference security tokens which are used to encrypt or sign parts of the message for a particular service. This protects the message from being read or changed by untrusted intermediary services.. Browsers must be extended with a plugin or script code to create such headers and perform the necessary cryptography necessary to secure these types of messages. In particular, web clients (browsers) do not typically implement Web Services security. As a result, the current, yet insecure, best practice is to delegate[4] full rights to the mashup server (man-in-the-middle) and hope that the user's rights to data and services are not abused. This entails the end user providing the mashup server with their security credentials for the back-end services in a format whereby they can be exploited.

The basic communication pattern for mashups has a client (e.g. web browser) authenticating to a mashup server, which in turn authenticates to one or more data sources. When the mashup server and data sources are in the same security domain (or have a pre-existing trust relationship), the mashup server can reuse the authentication credentials to authenticate to the data source. The result is, however, unrestricted delegation. Figure 1 – Mashup Authentication shows a similar scenario, except that the data sources are in disjoint security domains. The credentials needed to authenticate are shown by circles with different patterns, representing different credentials. Because the data servers are in different trust domains, the mashup server needs to possess sufficient credentials to authenticate to both back-end servers, and therefore could perform operations on behalf of the user without their consent. The problem gets magnified when data sources

 

aggregate data from multiple sources, therefore additional credentials are needed to access those resources. As credentials are accumulated and communicated to more mashup servers, there is greater opportunity to expose security credentials to untrusted servers, giving them full delegation rights.

Another application paradigm motivating secure distributed delegation is one in which the end-user delegates to a third party service limited rights to content at a service provider. The classic example is that of a user delegating to a photo printing service a set of rights to read photos hosted at a service provider.[5] The user wants the printing service to read a subset of photos, but not modify them, then read other photos,

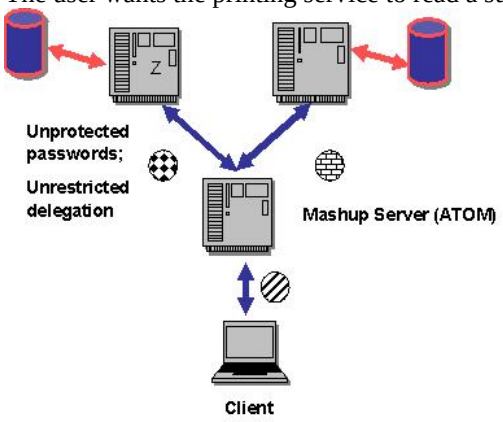


Figure 1 – Mashup Authentication

modify profiles etc. This is in contrast to providing end-user credentials to the photo printing service which would result in giving them full access rights to all of the accessible resources.

We argue that the following challenges need to be addressed in order to create secure and usable mashups:

**Wire protocols**: Interoperable wire protocols to securely authenticate the user and the principals involved in this delegation statement. Proprietary protocols such as AuthSub[6], OpenAuth[7], BBAuth[8] have been proposed. The OAuth[9] protocol has been published as an open protocol and WS-Trust[10]/WS-Federation[11] provide standards which can be used as part of the solution. These frameworks can support multiple authentication challenge protocols, including OpenID[12], as well as legacy userid/password. Additional work is needed to support chains of delegation – chains of mashup servers in a service request. Although OAuth provides a good start for the wire protocol it does not specify any framework or ontologies for rights delegation, identity management, authentication and user interaction. This protocol relies on many browser redirects which make it more prone to phishing attacks.

**Server to User Protocols and Usability**: A key challenge to be addressed is the user experience, where the mashup server specifies the rights that the end-user needs to delegate, and the subsequent interaction with the service provider. In the context of an interaction with a single mashup, the user may need to authenticate to multiple back-end services. This poses interesting usability challenges, including: How does the end-user determine the minimal rights that are reasonable to delegate? If multiple mashups (e.g., in separate <*iframe*>s[13]) are present on a web page, how does the end-user recognize which iframe (mashup) is receiving the delegation? When a back-end service is used by multiple iframes, how will the end-user determine to which iframe the rights are being delegated? In the presence of multiple mashups and authentication challenges, and a lack of visual cues, there is greater potential for phishing attacks and other misuses of user credentials. We propose using Higgins i-cards here to present the user with a consistent user experience that will help them better manage the delegation of rights to the mashup servers. A card selector interface is displayed and the user will be presented with a list of i-cards that match a delegation policy for the back-end service.

**Access Control Model**: Implicit in the notion of secure delegation of rights is the description of resources and services to which the end-user can delegate rights and the operations that are

 

permitted on these resources. What are appropriate ways for a service provider or a data source to specify user delegatable rights? What is an appropriate (and usable) framework for service providers to support fine-grained access rights, rather than one-size-fits-all approach? What is the incentive for service providers to enable fine-grained access rights?

**Legacy Systems**: To be successful, "legacy" systems need to be able to participate in secure delegation. The obvious choice is to interpose a proxy that participates in the new delegation protocols. This serves as a stop-gap until services start adopting the delegation protocol. We will discuss how to build such a proxy.

**Programming Mode**l: When should the end-user authenticate to the mashup server/data sources? Two simple models are *greedy* and *lazy*. A greedy model requests authentication for all of the mashup servers / data sources when the end-user starts using the mashup server initiating the service requests (before any requests are made to back-end services/data). The lazy model waits until authentication is required, and then presents security challenges and specification of delegation rights to the end-user. The greedy and lazy models present different programming model challenges.

The rest of the paper touches upon these issues and outlines possible technical approaches and solutions.

## *Wire Protocols*

A number of wire protocols have been proposed to support delegation. After a number of proprietary protocols such as OpenAuth, BBAuth and AuthSub, an open protocol, OAuth, has emerged as a possible standard. We also consider the use of Higgins as delegation framework.

### Delegation Using OAuth

The OAuth Protocol involves the following steps:

1. The mashup application (or consumer in the OAuth terminology) interacts with the service provider to obtain a Request token.
2. The browser or other user-agent is redirected to the service provider with this request token so the user can authenticate to the service provider and return a User Authorized Request Token, which implicitly encodes the Signed Delegation Statement.
3. The service provider redirects the user-agent back to the consumer with the User Authorized Request Token. The consumer then exchanges the Request Token for an Access Token.

One of the issues for using this protocol is usability. In the presence of a simple delegation scheme, where there is a simple client (e.g., one window or <iframe>), producer and consumer, the end-user will be able to recognize that the delegation rights are for the current interaction present on window (web page). When the scheme becomes more complicated (consumer is collecting data many from many providers or the provider is itself a consumer that connects to many providers) the user can easily be confused when keeping track of which rights are delegated to which services.

Another issue is phishing attacks. The protocol defines many redirects between the browser, consumer (or mashup) and service provider. These redirects make the user more vulnerable to phishing attacks. The user may be redirected to a web site that spoofs the service provider's site. The malicious site can ask the user to authenticate and then use the credentials to impersonate the user.

We propose using the Higgins framework to simplify the user interaction for specifying delegation rights and to provide a recognizable interface to authenticate to service providers to prevent phishing attacks. The users will only provide credentials to a trusted identity provider that is also trusted by the service provider.

### Delegation Using Higgins Framework

Windows *Cardspace*[14] and Higgins[15] *Identity Selectors* are means for achieving a secure authentication. *i-cards* are a graphical presentation of a collection of user identity information. They are provisioned by Identity Providers and can be consumed by Service Provider / Relying Party sites that trust the Identity

 

Provider. In the case of mashups, the Higgins infrastructure can authenticate the user to an Identity Provider trusted by the Service Provider. The user interacts with a trusted identity selector to choose an i-card with the appropriate credentials and delegation rights for the back-end service. The Identity Provider issues an encrypted access token for the Service Provider. This access token can be securely sent to the mashup server, and then obtain limited access rights to backend data/services. The flow is depicted in Figure 2.

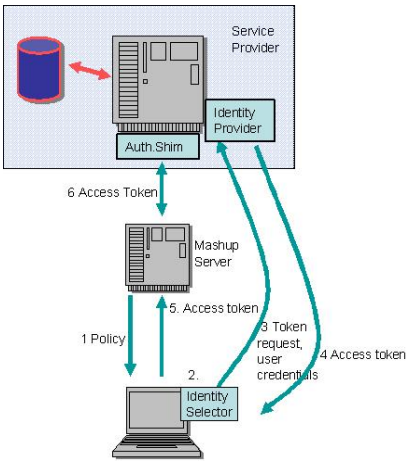


**Figure 2**

An i-card contains a set of claims about the user's identity. There are a set a predefined claims such as given name, surname, and email address. Identity Providers can also specify which claims they support, as well as define new claims in support of delegation.

Using i-cards involves the following steps:

1. The mashup server receives the service provider policy indicating which delegated rights and identity information/claims are requested by the service provider. The policy may be further modified to limit the delegation rights.
2. An identity selector is triggered at the client, which identifies and displays i-cards that match the policy sent by the mashup server. The user can review all claims that the mashup server is requesting, including which rights should be delegated to the mashup server. If the mashup server is asking for rights the user is not comfortable delegating then the user ends the interaction and the mashup server does not receive an access token.
3. The identity selector sends a request to the identity provider requesting a token with the claims specified in the policy from the mashup server. The user must include authentication information for the identity provider inside the request since the identity provider can only issue tokens for authenticated users. The identity provider may reside within the security domain of the service provider or it may be outside the service provider's security domain. The user can also request that the access token be restricted so that it can only be used by a specific service at the service provider.
4. The identity provider will create an access token with the required claims designated for a particular service at the service provider.
5. The user then sends the encrypted access token to the mashup server.
6. The mashup server sends the access token to the service provider to access resources. In the case of legacy system, there may be a shim that accepts / validates the token and then calls the legacy application with appropriate authentication credentials.

Identity selectors provide a more secure and usable way for the user to select identity claims to be shared with service providers.

 

## Access Control Model

Independent of which wire protocols are used in indicating delegations, usability considerations make it desirable for there to be uniformity in specifying a set of rights that can be delegated by the user. If an data/service provider would like to offer and participate in secure delegation protocols, there needs to be standardization of a number of details. Ideally, from an application developer perspective, one would like a declarative model, describing services / content to which rights may or may not be delegated by the user, the associated (implicit) permissions for operations on the content that can be delegated, and the expected trust model associated with the published content.

## Legacy Systems

Many mashups and data sources, including those based on RSS[16] and ATOM[17], already exist. For a delegation protocol to succeed, it will be necessary to support these "legacy" systems without modification. At the same time, we want a protocol that can be easily built into new services. For these reasons, we focus on protocols that can address these two constraints. As noted earlier, OAuth and Higgins provide delegation frameworks. We will not dwell in constructing new services based on these frameworks. To address legacy systems, we propose using a thin proxy, or shim layer, to sit between a consumer and producer.

Commonly used wire protocols include HTTP 1.1, LDAP and database access (e.g., JDBC). We will focus on HTTP 1.1 since it is the wire format for RSS and ATOM, two common protocols used by mashup servers. Figure 3 shows the insertion of a HTTP proxy, or shim, between the producer and consumer. The primary purpose of the proxy is to run the delegation protocol (e.g., OAuth), authenticating the end-user to the producer and allowing the end-user to specify the delegation rights. Using the delegation protocol, the rights are then returned to the mashup server for subsequent use.

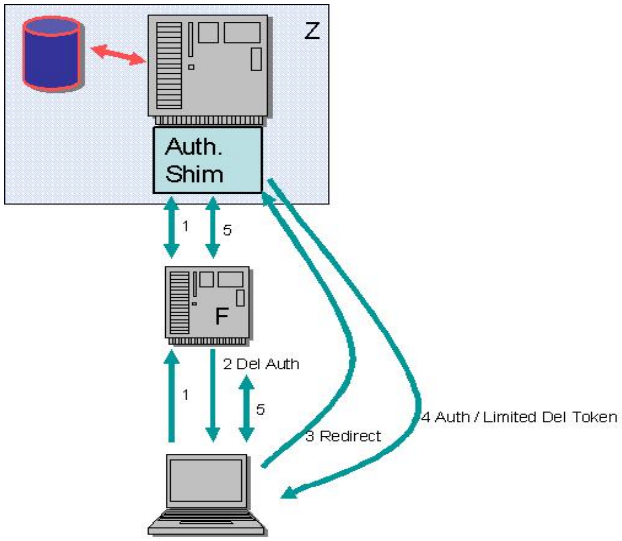


**Figure 3**

Since the mashup server makes requests to the producer, via the proxy, using REST calls, the proxy will be able to enforce delegation by inspection of the URI, and blocking requests for which the caller does not

 

have the associated rights. The techniques for other message formats (e.g., SOAP) wire protocols (e.g., LDAP) are outside the scope of this paper.

## Programming Model

Authenticating the end-user and specifying delegation rights is important from a programming perspective. Seemingly random requests for authentication will become annoying to the user, as well as have a cognitive impact[18]. From a security perspective, obtaining all delegation rights before running the mashup / application may result in incorrect specification of delegation rights[19] due to the lack of prior knowledge of the control path through the applications. While these are important usability and security issues, there are several practical programming considerations as well.

A greedy algorithm requires authentication to all of the services prior to executing the mashup. This necessitates determining all of the authentication / delegation requirements for all dependent services prior to execution. Two possible ways to gain this knowledge are: (1) metadata about the services being consumed – e.g., through some sort of catalog, in which the services are registered, or (2) through empirical testing – attempts to access the service that results in an authentication challenge (e.g., a HTTP status code *401* returned by the service). If a service is protected by a form-based authentication challenge, it may be more difficult to discover that the service requires authentication. For applications with cascading mashup servers, the challenge is in defining how the authentication metadata gets propagated, aggregated and managed by the mashup servers. For example, how would a mashup server discover that the authentication requirements of a dependent mashup server have changed (e.g., new and eliminated authentication requirements)?

In contrast, a lazy algorithm waits until a server/feed presents an authentication challenge to a mashup server. In general, there may be more than one mashup server in a chain of service requests. Each of these mashup servers will need to propagate the authentication challenge back to the browser, and then return the response (delegation rights) to the appropriate mashup server. In addition, there is the challenge of dealing with the HTTP programming model. Once a (mashup) service starts to generate a response (HTML, XML, JSON, etc.), and discovers that there is an authentication challenge, there is no simple way to suspend or roll back the output already generated and send an authentication / delegation request to the client (browser). At first glance, it seems that programmers will need to determine if there will be an authentication request before starting to generate output.

## Summary

In this position statement, we have laid out a number of challenges in the context of authentication and delegation for mashups. From a security perspective, we often are concerned with the wire protocols and associated cryptography. However, we have identified a set of critical issues that also need to be addressed:

- Usability of the protocol will be challenging, and will need to address phishing and specification of rights delegation.
- The protocols used will affect how developers write their code in the presence of authentication challenges from back-end services / data sources.
- The specification of the rights is important since it affects the authorization model at the service provider.
- A delegation protocol must be able to support legacy systems. The use of a proxy, or shim layer, might address this requirement.

We believe that this is fertile ground for research and welcome contributes to addressing these issues. We are beginning to explore these issues and hope to report on our results next year.

 

## References


[1] Web Services Architecture. W3C, February 2004. http://www.w3.org/TR/ws-arch/
[2] SOAP Version 1.2. W3C, April 2007. http://www.w3.org/TR/soap/
[3] Roy Thomas Fielding, "Architectural Styles and The Design of Network-based Software Architectures" PhD dissertation, University of California, Irvine, 2000.
[4] M. Abadi, M. Burrows, B. Lampson and G. Plotkin. "A Calculus for Access Control in Distributed Systems" ACM Transactions on Programming Languages and Systems, 1993.
[5] OAuth. October 2007. http://oauth.net/core/1.0/
[6] Authentication for Web Applications. http://code.google.com/apis/accounts/docs/AuthForWebApps.html
[7] AOL Open Authentication (OpenAuth) API. April 2007. http://dev.aol.com/openauth
[8] Browser-Based Authentication. http://developer.yahoo.com/auth/
[9] OAuth. October 2007. http://oauth.net/
[10] WS-Trust 1.3 OASIS Standard, March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html
[11] OASIS Web Services Federation (WSFED) TC. http://www.oasis-open.org/committees/wsfed/
[12] OpenID. http://openid.org
[13] W3C. HTML 4.01 Specification. December 1999. http://www.w3.org/TR/html401/present/frames.html
[14] Introducing Windows CardSpace/ http://msdn2.microsoft.com/en-us/library/aa480189.aspx
A Technical Reference for the Information Card Profile V1.0 - http://msdn2.microsoft.com/en-us/library/bb298802.aspx
[15] Eclipse Higgins Project - http://www.eclipse.org/higgins/
[16] RSS 2.0 Specification. July 2003. http://cyber.law.harvard.edu/rss/rss.html
[17] The Atom Syndication Format. IETF RFC 4287. December 2005. http://www.ietf.org/rfc/rfc4287.txt
[18] McFarlane, D. C., & Latorella, K. A. (2002). The Scope and Importance of Human Interruption in HCI Design. *Human-Computer Interraction, 17*(3), 1-62.
[19] J. H. Saltzer and M. D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE, September 1975.


 