MAuth: A Fine-grained and User-Centric Permission Delegation Framework for Multi-Mashup Web Services

Masoom Alam*, Xinwen Zhang†, Muhammad Nauman*, Sohail Khan* and Quratulain Alam*
*Security Engineering Research Group
Institute of Management Sciences, Peshawar, Pakistan
Email: {masoom,nauman,sohail.khan,quratulain}@imsciences.edu.pk
†Samsung Information Systems America, San José,USA
Email: xinwen.z@samsung.com

***Abstract*—Mashups are a new breed of interactive web applications that aggregate and stitch together data retrieved from one or more sources to create an entirely new and innovative set of services. The paradigm is not limited to social networks and many enterprises are redesigning their business processes to create interactive systems in the form of mashups. However, protecting users' private data from unauthorized access in mashups is a challenging security problem. Existing solutions for addressing the various authorization problems are limited due to all-or-nothing policy, third party dependence and scalability issues. In this paper, we present a general permission delegation model for mashups that is fine-grained, user centric and scalable. This contribution has the following objectives: We formally specify the dependency relationships among multiple web applications. Dependency relationships are categorized on the basis of specific data items. We present an extensible reference architecture for configuring multiple web applications and a session management protocol.**

***Keywords*-Mashup, Security, Access Control, Permission Delegation**

I. Introduction

Mashups are a new breed of web applications that aggregate and stitch together data retrieved from one or more sources to create an entirely new and innovation set of services [1]. This technology has its roots in the current state-of-the-art Internet technologies such as Asynchronous Javascript and XML (AJAX), and Service Oriented Architectures and is informally known as Web 2.0 [2]. It is fast becoming a de-facto standard for web applications ranging for connecting web services together in the from of social networks to e-Government to enterprise resource management [3], [4], [5], [6]. Initially the concept of mashups was limited to social networks. However, many enterprises are redesigning their business processes to create interactive systems in the form of mashups [7]. For example, cartographic data from Google Maps can be combined to real-estate data [8]. Another example is the collection of crime information from a law enforcement agency and depicting it on Google Maps [9]. In this way, a new and distinct web service is created that was not originally provided by either source.

Today, majority of mashups are focused on gathering data from backend services that do not require user authentication. However, with the increasing use of enterprise mashups, private data of users is involved that requires authentication and authorization mechanisms to be enforced. This means that the user must *delegate* a mashup the right to access data from a web application that hosts her private data. This is the problem of delegation in mashups and have many aspects. Firstly, the user must have provision to choose specific data items among others for delegation. Further, the use of third-parties is a common solution for complex delegation scenarios. But, if more than one web applications or mashups – called multi-mashups – are involved, user data has to pass from multiple hops and thus requires permission management at each hop. Therefore, user issued delegations can also be misused if a web application or mashup is malicious. To summarize, current authorization techniques used in mashups have the following problems.

Problem Statement:

1) Based on all-or-nothing policy: This means that either complete delegation occurs or none at all. For example, a user cannot restrict the list of friends accessible to a web application. She has to release all the information or no information at all;
2) Dependence on third parties: Upon user request a third party must issue a delegation permit to the mashup for accessing user data. This leads to synchronization overhead between third parties and backend services;
3) Are not scalable: This means that mashups authorization solutions are limited to scenarios in which a user gets data from multiple backend services through a *single mashup*. But as the number of web applications increases authorization becomes more complex and infeasible. For example, combining a list of friends from a social network with Google maps service to create a friend finder service requires simple authorization. However, if the web application responsible for collecting friends list also take data from another web application, the authorization among multiple

978-0-7695-4129-7/10 $26.00 © 2010 IEEE
DOI 10.1109/SERVICES.2010.112

web applications requires thorough analysis.

Contributions: In this paper, we present a general delegation model for mashups that addresses the aforementioned limitations. In particular, it provides a delegation mechanism for web applications that is fine-grained, requires zero-dependence on third-parties and is scalable for multi-mashup scenarios where multiple web applications are connected with each other in order to accomplish a task. Our contribution in this paper are three-fold:

1) We formally specify the relationship among multiple web applications on the basis of specific data items. This enables a fine-grained user centric permission delegation in mashups.
2) Dependent data items are formalized in the form of a graph. A session management mechanism is derived from the dependency relationships among multiple web applications regarding specific data items.
3) Finally, an extensible reference architecture coupled with a session management protocol among multiple web applications for specific data items is presented.

Outline: The rest of the paper is organized as follows: Section II presents a motivating example which will be used through out the paper to describe our approach and related work. Section III formally describes our approach. The reference architecture and the protocol is detailed in Section IV. Section V concludes and presents an outlook of the paper.

II. Background

A. Motivating Example

To demonstrate the limitation of the existing approaches towards authorization in mashups and to present our own approach, we take an example application scenario (cf. Figure 1). User Bob wants to calculate the `tax returns` by applying his local government policy available at `gov.com`[1] to his finances. There are a number of web applications available for this purpose but Bob chooses `financial.com`. However, `financial.com` needs Bob's `expenditure report` and `payment information`, which are available at `mintoo.com`. Bob's `expenditure report` is derived from the `utility bills` and other payments he has made through `mintoo.com`. For this purpose, `mintoo.com` has access to Bob's `bank statement` through `mybank.com` and `utility bills` through `utilitiybills.com`.

We use this example application scenario throughout the paper to explain our approach and to consider the limitations of existing approaches.

[1]All of the web addresses mentioned in this paper are for purposes of illustration only a re not related to the actual sites hosted at these addresses.

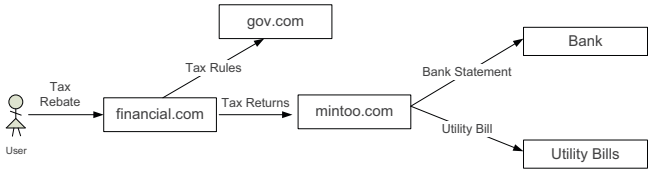

Figure 1. Use Case of a Multi-Mashup Scenario

B. Related Work

Due to the ever increasing popularity and utility of free and enterprise mashups, several problems related to these scenarios are being studied in the scientific community. In this section, we discuss the shortcomings of the various existing approaches for solving the authorization problems in mashups. We begin with a discussion on the current best practices available on the web for mashups authorization and their corresponding problems.

A fairly simple solution to the mashup authorization problem is that user is asked for login credentials such as username/password for the backend service. For example, `financial.com` asks user Bob for his login credentials at `mintoo.com`. These credentials are then used by `financial.com` for accessing user data at `mintoo.com`. Examples of such mashups are Tweetdeck.com [10] and yodlee.com [11] etc.

The obvious problem in this solution is the misuse of the login information of the user [12]. Since the mashup provides the username and password, the backend service considers the mashup as an owner and allows it to perform all actions that are permitted to the user. Also it is possible for the mashup to save the login information and misuse it in the future without the consent of the user. This approach utilizes an all-or-nothing policy. Moreover, using this simple authorization solution may increase the security risk in multi-mashup scenarios as user credentials may be floating around the web due to being shared by several web applications.

OAuth [13] provides an open set of specifications for authorization in a mashup scenario. Web application in an OAuth specs requests an unauthenticated token from a backend service. This unauthenticated token is forwarded to the corresponding user which authenticates it by providing login credentials at the backend service. The user is redirected to the backend service for this purpose. For example, `financial.com` redirects the user to `mintoo.com` with an unauthenticated token. Bob authenticates this token by logging in to `mintoo.com` using this security token. The `financial.com` can then access user data from `mintoo.com`. This protocol provides very limited forms of permission choice to derive flexlible and/or user centric arguments. Moreover, an important issue is the usability of this approach. In the presence of a single backend

service and a mashup, the approach works fine. However, when mashups are further connected to other mashups, the handling of enormous number of tokens quickly becomes infeasible at the user end. Moreover, more than one mashups between user and backend services can cause too many redirections between user and different mashups and thus, may confuse the user resulting in the selection of fake websites.

AuthSub [14] is a propreitary protocol for authorization for Google services. Third party applications can access user accounts on behalf of users using AuthSub call. The user is redirected to an access consent page where it logs in to the Google account and grants or denies access to the corresponding service. Likewise, this protocol too has very limited forms of permission choice. Upon grant access, the user is redirected to the web application site. Otherwise, the user stays at the google site. The authentication token is an opaque identifier and is used for references purposes only. A major disadvantage of the protocol is all-or-nothing policy.

The approach of PermitMe [12] suggests a Permit Grant Service (PGS) for handling the delegation issues. The PGS sits between the user and the mashup and issues a delegation permit to the web application. These encrypted delegation permits are stored with the web application and thus may be misused, which is a severe security problem. The approach only shifts the responsibility of authorization from backend service to the PGS with significant redirection overhead. PGS requires security requirements and/or policies of the backend service to delegate permission, which is only viable for a small set of backend services like Google services, not for general Internet-based web services.

Currently, browsers level security for mashups is being studied at length [15], [16], [17]. Our focus in this paper is on the delegation issues in server-side mashups.

We come to the conclusion that delegation in mashups is quite different from the existing delegation models [18], [19]. In such scenarios, user has to specify delegation policies in each step of delegation. For example, Bob has to specify rights for `mintoo.com` and other web applications. We believe that a general delegation model is needed that addresses the following concerns:

1) Fine-grained delegation: A delegation model that can specify the relationship among multiple web applications on the basis of specific data items.
2) Zero-dependence on third parties: A user-centric mechanism is needed that can reduce third party trust dependency management. Thus, a user is in charge of all his private information.
3) Multi-mashups: Finally, the delegation model for mashups shall be extensible to incorporate multiple web applications. The underlying protocol shall clearly make a separation between security and session management. This means that a user can configure multiple web applications that are dependent on each other regarding specific data items.

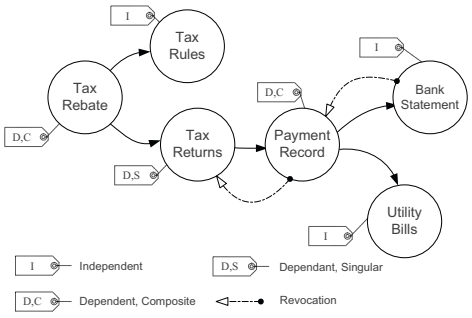

Figure 2. Dependency Graph

Below, we describe our delegation model formally that incorporates these differences to specify delegation in multi-mashup scenarios.

III. User-centric Permission Delegation in Mashups

Traditionally, mashups are considered as independent web applications that can only combine data from backend services. This categorization itself limits the potential value of web applications. Rather than dividing them into mashups or backend services, we term both types as *web applications*. In our settings, it is possible that a web application can purely act as a mashup – collects data from external sources only, as a backend service – where data is locally stored or as a hybrid in which case it can store as well as combine data from external sources. Thus, web applications may depend on other web applications regarding some data items. These dependencies can be resolved by assigning rights to the source web applications for accessing private data hosted at a target web application. For example, Bob can configure at `mintoo.com` the rights for accessing his `expenditure report` by `financial.com`. Section III-A presents the dependency resolution mechanism in our delegation model.

A. Dependency Resolution

We define a web application as a software that can be accessed by thin clients i.e. browsers over a network such as Internet or intranet. It is a set of services that a user can access with the help of her browser. These services provide access to a set of data items available locally or retrieved from external sources. Formally:

Definition 1: A web application provides a set of services $S = \{s_1, s_2, \ldots, s_n\}$ that provide access to a set of data items D. $\delta = \{(d_1, v_1), (d_2, v_2), \ldots, (d_n, v_n)\}$ is a set of pairs comprising of data items and their corresponding values where $v_i \in dom(d_i)$ and $d_i \in D$.

A web application may receive data from any number of external sources. For example, `financial.com` collects `tax returns` from `mintoo.com` and `tax rules` from `gov.com`. Thus, `tax rebate` at `financial.com`

depends on `tax rules` and `tax returns` from `gov.com` and `mintoo.com` respectively. The relationship between data items is defined as a *Dependency Graph*.

Definition 2: A Data Item Dependency Graph, or simply Dependency Graph, is a directed, acyclic graph (D, E) where D is a set of nodes representing data items and E is a set of edges representing the dependency relationship between data items. An edge directed from d_i to d_j represents that d_i is dependent on d_j. This dependency is denoted as: $d_i \rightsquigarrow \{d_j\}$. We write $d_j \in \eta(d_i)$ if d_i is dependent on d_j where $\eta : D \rightarrow 2^D$ is a function that takes a data item as input and returns a set of data items on which the input depends.

Each data item in the dependency graph has a *category* and a *type*. The category is either *independent* or *dependent*. An independent data item has no dependency relationship with other data items i.e. an independent data item has no outgoing edge in the dependency graph. A dependent data item, on the other hand, may depend on one or more data items.

Definition 3: Category function $\varsigma : D \rightarrow \{dependent, independent\}$ maps each data item to one of two categories: dependent or independent. The Type Function $\theta : D \rightarrow \{singular, composite\}$ maps each data item node in the dependency graph to either singular or composite data type. Data items belonging to the singular type must depend only on one data item. Formally:

$$\varsigma(d_i) = independent \iff \eta(d_i) = \phi$$
$$\varsigma(d_i) = dependent \iff \eta(d_i) \neq \phi$$
$$|\eta(d_i)| = 1 \iff \theta(d_i) = singular$$
$$|\eta(d_i)| > 1 \iff \theta(d_i) = composite$$

For example, in Figure 2, `tax rebate` is dependent on `tax rules` and `tax returns`. Data item `tax returns` is dependent on `payment record`, which is constituted from `bank statement` and `utility bills`. Formally, $paymentrecord \rightsquigarrow \{bankstatement, utilitybills\}$ or $\eta(paymentrecord) = \{bankstatement, utilitybills\}$. Similarly, $\theta(paymentrecord) = composite$, $\varsigma(paymentrecord) = dependent$, $\varsigma(bankstatement) = independent$ and so on.

A web application asks permission to access specific data items from other web applications through a Policy Authorization Request (PAR).

Definition 4: Policy Authorization Request (PAR) is defined as a set of data item names and the rights requested on these data items. $PAR = \{(d_1, r_1), (d_2, r_2), \ldots, (d_k, r_k)\}$ is a request and $PAR \subseteq D \times R$, where R is the set of rights.

Rights are not just limited to `read/write`, but it is left on the discretion of the target web application. For example, `MoveCalendar`, `ChangeDate` etc., all are rights, that can be defined by a backend service. Based on the user preferences, a web application can define a set of policies. These policies are defined as a set of requests that are permitted and the set of requests that are denied by the policy. A 2-valued access policy P is a function mapping each PAR to a value in $\{Y, N\}$. R^P_Y, and R^P_N denote the set of requests permitted and denied by the policy P respectively and R_{eq} is a set of PARs where $R_{eq} = R^P_Y \cup R^P_N$ and $R^P_Y \cap R^P_N = \emptyset$. As an in charge, a user has to decide for each corresponding PAR. For example, once a PAR has been generated by `financial.com` for accessing `expenditure report` at `mintoo.com`, Bob gives his consent regarding this PAR. We define user consent as follows:

Definition 5: User Consent (UC) is defined as a set of attribute names and their corresponding rights. $UC \subseteq PAR \cup C$ where C is set of constraints such that the user consent restricts the set of data items requested through the PAR.

This user consent is called the delegation policy token, which defines the rights that the user allows for each data item requested by the web application.

Revocation on the other hand is more simpler in such delegation scenarios. For example, a user can restrict the access of a specific data item (such as the duration after which revocation should take place) through her consent. Another possibility is that user can directly revoke the rights from a web application. Figure 2 shows that if the right for accessing `bank statement` has been revoked by `bank.com`, the payment record can no longer be constituted and thus the delegation chain is not valid.

B. Session Derivation

After a user has resolved the dependencies related to all data items, the corresponding data items can be retrieved by the configured web application. In order to avoid misuse of the delegation policy token, a common session must exist between dependent hosting web applications. A common session ensures that a web application can only access the data items when the user is currently logged in. Sessions can be depicted through graphs, where each node represents a web application and each edge represents an active session between their hosting application for a specific data item. Formally:

Definition 6: A Session Graph is a directed, acyclic graph (W, F) where W is a set of nodes representing web applications hosting the data items and F is a set of edges representing the corresponding data items for which web applications are dependent on each other. An edge directed from w_i to w_j represents that there is an active session between w_i and w_j regarding d and is written as $w_i \xrightarrow{d} w_j$. Further, data items are mapped to their hosting web applications using the data item association function $\pi : D \rightarrow W$. If a data item d is hosted by a web application w, we write $\pi(d) = w$.

Session graphs can be derived from data item dependency graphs automatically. Figure 3 shows three cases of

Algorithm 1 Session Derivation Algorithm

Input: dependency graph
Output: set of session graphs

1) // set the current state to startNode at the beginning
2) current = getStartNode(dependency graph)
3) independentItems = $\{x|\varsigma(x) = independent\}$
4) allPaths = $\emptyset$
5) visited' = visited $\cup\{current\}$
6) **foreach** target in independentItems **do**
7) findPath(current, target, path, null, visited);
8) **end for**
9) transformDataItemsToWebApps(allPaths)
10) normalizeSessionGraph(allPaths)

11) **begin function** findPath(current, target, path, previous, visited)
12) **if** current = target **then**
13) path' = path $\cup\{current \rightarrow target\}$
14) allPaths' = allPaths $\cup\{path\}$
15) **return**
16) **end if**
17) **if** current $\in$ visited **then**
18) **return**
19) **end if**
20) path' = path $\cup\{previous \rightarrow current\}$
21) **if** $\varsigma(current)$ = dependent **then**
22) **if** $\theta(current) = singular$ **then**
23) findPath($\eta(current)$, target, path, current, visited)
24) **end if**
25) **if** $\theta(current)$ = composite **then**
26) **foreach** node in $\eta(current)$ **do**
27) findPath(node, target, path, current, visited)
28) **end for**
29) **end if**
30) **end if**
31) visited' = visited $\cup\{current\}$
32) allPaths' = allPaths $\cup\{path\}$
33) **end function**

how fragments of a dependency graphs can be converted to session graphs. In Figure 3.a the initial data item is dependent on a single data item that is independent. For this type of scenarios, the session needs to exist between web applications hosting d_2 and d_1 and the user u i.e. $w_1 \xrightarrow{d_2} w_2$. In the second case (Figure 3.b), the initial data item again depends on one data item (d_2) but d_2 again depends on d_3. Formally, $w_1 \xrightarrow{d_2} w_2 \wedge w_2 \xrightarrow{d_3} w_3$. The third case (Figure 3.c) is similar except that d_2 depends on d_3, which is itself dependent and composite. It depends on two data items d_4 and d_5. Since there are two terminal independent data items, two sessions must exist for the proper resolution of data

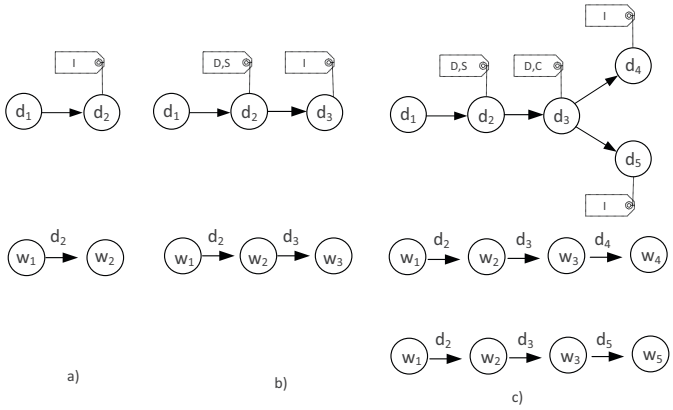

Figure 3. Three Cases in Session Derivation

items at runtime.

Based on these three cases, we have defined an algorithm that takes the dependency graph as input and derives the session graphs based on the dependencies of data items in the input graph. It first collects all independent items using the ς function (Line 3). It then loops over the set of these items collecting paths representing sessions over the dependency graph in `allPaths` variable (Line 4). The `findPath` function takes the current and the target node, the path variable, previous node visited and the set of all visited nodes as input. It calculates the path from the current to the target node (Lines 7, 11). If the current node is the same as the target node (Line 12), it returns adding the current node to the graph. Otherwise, it adds the step from the previous node to the current node to the path variable (Line 20).

If the current node is a dependent singular node, `findPath` calls itself recursively to calculate the path from the current node to the target node (Line 23). Otherwise, if the current node is dependent composite, it iterates over all the nodes that the current node is dependent on collecting the paths in the recursive call (Line 23).

The *transformation function* (Line 9) applies the π function on all nodes of the $allPaths$ variable to convert nodes representing data items into those representing their corresponding web applications and dependent data item is represented over the edge.

The *normalization function* (Line 10) reduces the resultant session graphs by applying the following two cases: **Case I:** $w_i \xrightarrow{d_1} w_j \wedge w_i \xrightarrow{d_2} w_j$. In this case, w_i and w_j are connected through two edges representing d_1 and d_2 respectively that shows two different sessions. Since a session can be shared at a target web application for multiple data items, normlization function removes one edge and places the second data item over the first edge i.e. $w_i \xrightarrow{d_1,d_2} w_j$.
Case II: $w_i \xrightarrow{d_1} w_j \wedge w_j \xrightarrow{d_2} w_j$. In this case, one data item

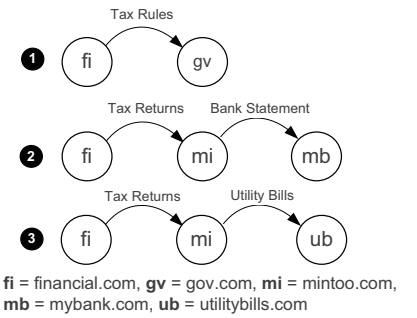

Figure 4. Session Graphs Derived for Usecase Shown in Figure 1

of a single web application depends on another data item of the same web application. In this case, we get an edge directed from a node to itself. Since sessions do not need to exist between a web application and itself, we normalize the session graph that removes all such edges. Figure 4 shows the sessions derived from the usecase given in Figure 1.

Based on the model described in Section III-A and III-B, the following section highlights the sequence of activities involved in both configure-time and runtime protocols.

IV. Proposed Architecture and Protocols

A user configures a web application such that other web applications can access her data. Policy Authorization Protocol is used to configure web applications with user preferences (cf. Section IV-A). After configuration, web applications can work with others in order to achieve a common task. The runtime management issues such as session management among multiple web applications while accessing dependent data items is the concern of session management (cf. Section IV-B).

A. Policy Authorization Protocol

The Policy Authorization Protocol takes place the first time a user tries to use a web application that requires data from other web applications. The steps involved in the protocol are as follows (cf. Figure 5):

1) Alice logs in at the web application using the link `http://www.financial.com/login`.
2) After authentication, Alice is shown a set of data items (e.g. tax returns) that the web application needs for its normal operation. These data items might belong to different web applications for each of which the user has to repeat the following steps.
3) Upon Alice's consent, she is redirected to `mintoo.com` for the protected data item (e.g. `tax returns`). It is important to note that some data items might be unprotected such as `tax rules` from `gov.com` and are thus available without authentication. For protected data items, Alice logs in at the corresponding web application (`mintoo.com`) and presents a token, which can be used by `mintoo.com` to fetch the Policy Authorization Request from `financial.com`.
4) As described in Section III, Policy Authorization Request contains a set of attributes for data item names and their corresponding rights. The Policy Authorization Request is shown to Alice in a listing. This visual depiction helps her in understanding which data items are requested and for what purpose.
5) If all the data items are available locally, the user preferences along with the Policy Authorization Request is transformed to a policy token called the Delegation Policy Token. This delegation token can be encrypted and is used by the source web application for accessing the specified resources at the target web application.
6) If some of the the data items are not available locally, then Step 2, 3 and 4 are repeated for each individual data item not available locally.

B. Session Management

After configuration using the Policy Authorization Protocol, the user can access the data items at the first web application. The first web application has been assigned a delegation token for accessing dependent data items. However, the target web application also needs a session token in order to avoid misuse of the delegation token by the source web application. This ensures that if a user is currently logged in, only then the web application will be able to release the corresponding data item. In our session management protocol, whenever a web application requests a data item from a target web application, the target web application can handle the request in the following manner. If the data item is locally available, the web application needs to verify that the user is currently logged in before releasing the data item. On the other hand, if the data item is dependent and the web application requires it from another web application, it asks for the dependent data item along with the delegation policy token. If the data item is composite, the same step is performed for each of the data items on which the requested data item depends.

In order to have a collaboration between these web applications, session management is of utmost importance and requires that the user log in to each of the involved web applications before data items can be released. The number of logins may increase drastically with the increasing number of web application. To remedy this problem, we propose the use of a single sign-on mechanism for providing login status information to different web applications.

In our target architecture, we use OpenID [20] as the single sign-on mechanism. OpenID provides a mechanism for identifying users across multiple websites based on their login status at their identity provider web application. OpenID can be applied in Policy Authorization Protocol for reducing the security risks associated with multiple username/password pairs for multiple web applications.

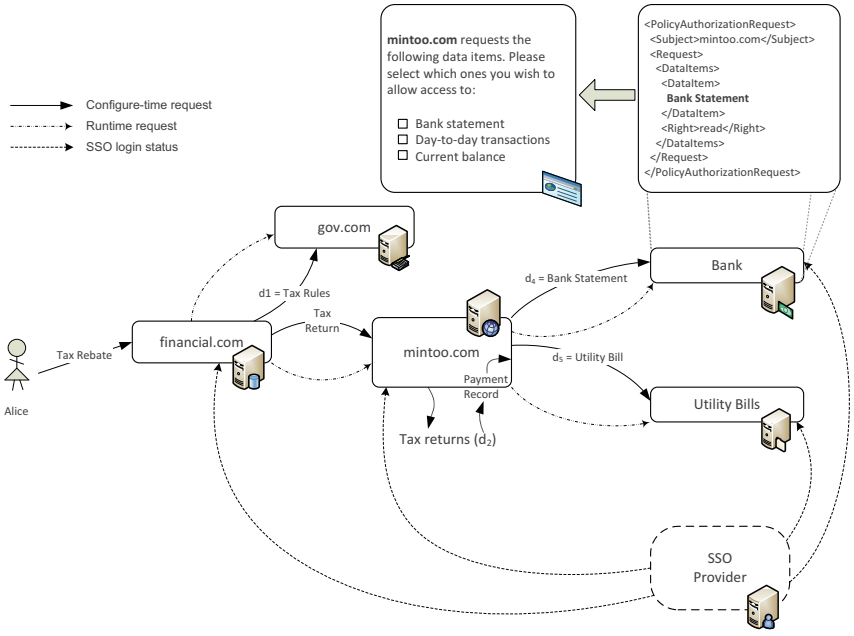

Figure 5. Target Architecture

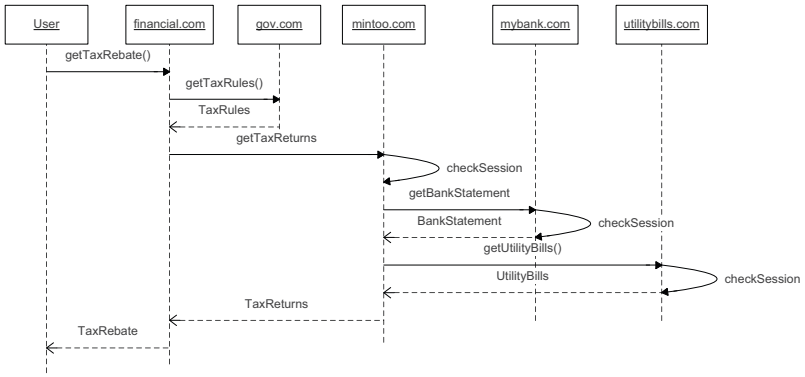

Figure 6. Sequence Diagram for Multi-mashup Data Item Retreival

The protocol for retrieval of data items follows these steps (cf. Figure IV-A):

1) Alice requests a service from `financial.com`. We assume that the data item and its dependencies have been resolved at configure-time using the Policy Authorization Protocol (cf. Section III-A).
2) `financial.com` accesses `mintoo.com` for acquiring `tax returns`.
3) `mintoo.com` checks to see if Alice is currently logged in through the OpenID provider.
4) The target web application (`mintoo.com`) checks to see that the session token is valid and that the delegation policy (generated as a result of PAR and user consent) allows the release of the requested data item.
5) Each web application after receiving the required data item releases the generated data item to the requesting web application; in case of independent data items, the only requirement is the validity of the session and delegation token.

Note that in the runtime portion of our protocol, the user does not have to perform any step other than logging in to the source OpenID provider and allowing the different web applications access to status information (which is a one-time operation). This allows for a usable protocol, which requires minimum user intervention, thus limiting the possibility of threats such as phishing attacks.

V. Outlook

Mashups are an architectural paradigm in which data is retrieved from one or more sources into a single inte-

grated whole to create entirely new and innovative services. However, allowing mashups to retrieve sensitive data on behalf of the user may lead to security concerns. In this paper, we presented a general delegation model for mashups, which is fine-grained as the web applications are dependent on each other on the basis of specific data items. Our approach is user-centric, which allows zero-dependence on third parties for issuing delegation permits and is capable of handling multiple web applications. We also presented a session management protocol based on single sign-on solutions in order to avoid multiple username/passwords. Our protocol is capable of configuring multiple web application through policy authorization requests. We have developed a prototype implementation using Eclipse in a Web Tools Platform (WTP) [21] for developing web applications. In our current implementation, each web application is composed of a set of services. These services are used to configure user preferences and access data items respectively. Each respective service can be either categorized as a configure service or runtime service. Configure services are used to configure the user preferences regarding accessing specific data items. Runtime services on the other hand allow the user or a web application to access the corresponding data items, upon presenting the issued delegation policy token. These services take XML documents as input and process them using FastXML parser [22]. Our evaluation shows that once, these services are configured, with the help of single sign-on solution such as OpenID, it is very easy to get multiple web applications work simultaneously. Currently, we are planning to develop open source API and libraries that can be used for interacting with configure-time and runtime services in our protocol.

REFERENCES

[1] D. Merrill, "Mashups: The new breed of Web app," *IBM Web Architecture Technical Library*, 2006.

[2] What Is Web 2.0 - O'Reilly Media, available at: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[3] M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y.-H. Ng, D. Simmen, and A. Singh, "Damia: a data mashup fabric for intranet applications," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 1370–1373.

[4] A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: interactive composition of data mashups," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 775–784.

[5] J. Wong and J. Hong, "Making mashups with marmite: towards end-user programming for the web," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM New York, NY, USA, 2007, pp. 1435–1444.

[6] R. Ennals and M. Garofalakis, "MashMaker: mashups for the masses," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM New York, NY, USA, 2007, pp. 1116–1118.

[7] A. Jhingran, "Enterprise information mashups: integrating information, simply," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 3–4.

[8] HousingMaps, available at: http://www.housingmaps.com.

[9] Chicago Crime Data, available at: http://chicago.everyblock.com/crime/.

[10] TweetDeck: a Simple and Fast Way to Experience Twitter, available at: http://www.tweetdeck.com.

[11] Yodlee - Innovative Bill Pay, Personal Finance and Online Account Opening Tools, available at: http://www.yodlee.com.

[12] R. Hasan, M. Winslett, R. Conlan, B. Slesinsky, and N. Ramani, "Please permit me: Stateless delegated authorization in mashups," in *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, 2008, pp. 173–182. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2008.24

[13] OAuth - An open protocol to allow secure API authorization, available at: http://www.oauth.net.

[14] Authentication for Web Applications - Account Authentication API, available at: http://code.google.com/apis/accounts/docs/AuthForWebApps.html.

[15] J. Howell, C. Jackson, H. Wang, and X. Fan, "MashupOS: Operating system abstractions for client mashups," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2007.

[16] F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama, "Smash: secure component model for cross-domain mashups on unmodified browsers," 2008.

[17] C. Jackson and H. Wang, "Subspace: secure cross-domain communication for web mashups," in *Proceedings of the 16th international conference on World Wide Web*. ACM New York, NY, USA, 2007, pp. 611–620.

[18] M. Blaze, J. Feigenbaum, and A. Keromytis, "KeyNote: Trust management for public-key infrastructures," *Lecture Notes in Computer Science*, vol. 1550, no. 59-63, pp. 33–60, 1999.

[19] N. Li, B. Grosof, and J. Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization," *ACM Transactions on Information and System Security*, vol. 6, no. 1, pp. 128–171, 2003.

[20] OpenID Developers, available at: http://openid.net/developers/.

[21] Eclipse Web Tools Platform, available at: http://www.eclipse.org/webtools/.

[22] FastXml.net: Fast XML Processing, available at: http://www.fastxml.net/.