Integrating OAuth with Information Card Systems

Haitham S. Al-Sinani
Information Security Group
Royal Holloway, University of London
Haitham.Al-Sinani.2009@rhul.ac.uk

Abstract—**We propose a novel scheme to provide client-based interoperation between OAuth and an Information Card system such as CardSpace or Higgins. In this scheme, Information Card users are able to obtain a security token from an OAuth-enabled system, the contents of which can be processed by an Information Card-enabled relying party. The scheme, based on a browser extension, is transparent to OAuth providers and to identity selectors, and only requires minor changes to the operation of an Information Card-enabled relying party. We specify its operation and describe an implementation of a proof-of-concept prototype. Security and operational analyses are also provided.**

Keywords: Information Cards, CardSpace, OAuth

I. Introduction

To mitigate identity-oriented attacks, a number of identity systems (e.g. CardSpace, OAuth, OpenID, etc.) have been proposed [1]. An identity provider (IdP) in such systems supplies a user agent (UA) with a security token that can be consumed by a relying party (RP). Whilst one RP might support an Information Card system, another might only support OAuth [2]. To make these systems available to the largest possible group of users, interoperability between such systems is needed. We propose a scheme to provide interoperation between OAuth and an Information Card-based system.

The scheme operates with a variety of Information Card-based systems, including CardSpace and Higgins. For simplicity of presentation, we describe its operation with CardSpace, a widely-discussed example of an Information Card system.

We consider CardSpace-OAuth interoperation because of OAuth's fast-growing adoption by widely-used Internet service providers such as Facebook and Twitter. Complementing this, the wide use of Windows, recent versions of which incorporate CardSpace [3], means that enabling integration between the two systems is likely to be beneficial for large numbers of users. CardSpace-OAuth integration is attractive since both schemes support the exchange of user attributes.

The remainder of the paper is organised as follows. Section II reviews related work. Section III gives an overview of CardSpace and OAuth, and section IV presents the integration scheme. In section V we provide an operational analysis and, in section VI, we describe a prototype implementation. Finally, section VII concludes the paper.

II. Related Work

We first review related work. A similar scheme [4] has previously been proposed to support CardSpace-Liberty interoperation. However, unlike the scheme proposed here, the CardSpace-Liberty integration scheme does not support the exchange of user attributes and does not operate with HTTPS-enabled websites. Two further similar schemes have recently been proposed, allowing interoperation between a CardSpace-enabled RP and a Shibboleth IdP [5] or an OpenID IdP [6].

A Liberty-CardSpace integration scheme has also been proposed by Jørstad et al. [7], in which the IdP is responsible for supporting interoperation. The IdP must therefore perform the potentially onerous task of maintaining two different identity management systems. The IdP must always perform the same user authentication technique, regardless of the used identity system. This scheme also requires the user to possess an SMS-capable mobile phone. By contrast, the scheme proposed in this paper supports client-side interoperation between CardSpace and OAuth, does not require use of a handheld device, and does not enforce a specific authentication method.

III. CardSpace and OAuth

A. CardSpace

1) Introduction: CardSpace [3] is Microsoft's implementation of a digital identity metasystem, in which digital identities are represented to users as Information Cards (or InfoCards). There are two types of InfoCards: personal (self-issued) cards and managed cards, issued by remote IdPs.

CardSpace is supported in Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, e.g. Firefox and Safari, also exist. An updated version, CardSpace 2.0 Beta 2, was released, although Microsoft announced in early 2011 that it will not ship; instead Microsoft released a technology preview of U-Prove (http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx). We refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, that is available as a free download for XP and Server 2003, and which has been approved as an OASIS standard [8].

2) Using Personal Cards: Personal cards are created by users themselves, and the claims listed in such cards are asserted by the self-issued identity provider (SIIP) that co-exists with the CardSpace identity selector. Personal cards can contain claims of 14 editable types, including *First Name*, *Last Name*, and *Web Page*. The scheme proposed here uses personal cards to make information provided by an OAuth IdP (Resource Server) available to CardSpace RPs via the selector.

The personal card protocol operates as follows, in the case where the RP does not employ a security token service (STS).

1) UA → RP. A user visits a CardSpace-enabled RP page.

2) RP $\rightarrow$ UA. The login page is returned in which the RP security policy is embedded.
3) User $\rightarrow$ UA. The user clicks the CardSpace icon/option, and the selector (which is passed the RP policy) is activated. If the RP is visited for the first time, the selector will display the RP identity, giving the user the option to either proceed or abort the protocol.
4) Selector $\rightarrow$ InfoCards. The selector highlights InfoCards matching the RP policy.
5) User $\rightarrow$ InfoCards. The user chooses (or creates) a personal card. The user can preview the card to ensure that they are willing to release the claim values.
6) Selector $\rightleftharpoons$ SIIP. The selector sends a Request Security Token (RST) to the SIIP, which responds with a Request Security Token Response (RSTR).
7) UA $\rightarrow$ RP. The RSTR is passed to the UA, which forwards it to the RP.
8) RP $\rightarrow$ User. The RP verifies the token, and, if satisfied, grants access.

The PPID (private personal identifier) is an identifier linking an InfoCard to an RP. During card creation, a card ID and master key are created and stored. When a user first uses a personal card at an RP, CardSpace generates a card-site-specific PPID by combining the card ID with data taken from the RP certificate, and a card-site-specific signature key pair by combining the card master key with data taken from the RP certificate. The PPID could be used on its own as a shared secret to authenticate a user to an RP. However, it is recommended that the associated (public) signature key should be used to verify the signed security token to provide a more robust authentication method [3].

B. OAuth

1) Introduction: OAuth (Open Authorisation) is an emerging identity management standard, enabling an end-user to grant an application controlled access to personal information (e.g. user attributes) stored at a third-party site, without divulging long-term credentials such as passwords. The four entities involved in the protocol are: the *Resource Owner*, typically an end-user (or their UA); the *Client*, an application requesting access to user resources; the *Resource Server*, a server hosting user resources; and the *Authorisation Server*, a server that issues access tokens to clients after first authenticating the Resource Owner and obtaining its authorisation; the latter two roles are typically performed by a single entity.

2) Operational Protocol: Two major (incompatible) versions of OAuth have been released: OAuth 1.0 [2] and 2.0 [9], [10]. We describe the latest version, OAuth 2.0. OAuth 2.0 support four authorisation grant types, including *authorisation code* and *implicit*, which are described next.

- An 'authorisation code' is typically a short-lived random string. Such a value is supplied by an Authorisation Server to a Client if a Resource Owner authorises a request made by the Client to access (a) specific user resource(s). The Client uses the authorisation code to request an access token (typically an opaque, user/issuer-revokable string that indicates permission to access specific information for a defined time period) from the Authorisation Server. Before issuing such a token, the Authorisation Server first authenticates the Client by checking that the credentials provided by the Client match those issued to it when it registered with the Authorisation Server. If successful, the Authorisation Server generates an access token and sends it to the Client via a secure back-channel; the use of which means that this grant type cannot be supported by the scheme proposed here.
- An authorisation grant is said to be 'implicit' if the access token is issued to the Client as a direct result of Resource Owner authorisation. Since it requires fewer round trips to obtain an access token than for the authorisation code type, the implicit grant type improves the responsiveness and efficiency of certain clients, including browser-hosted clients. This grant type is supported in our scheme.

Prior to use, the Client must register with the OAuth Authorisation Server; this could involve the use of an HTML registration form provided by the Authorisation Server. During registration, the Authorisation Server collects certain data about the Client, including the Client type, its redirection URL, and any other server-required data, e.g. the Client name. The Authorisation Server issues the registered Client with a unique identifier and a secret used for Client authentication when using the authorisation code grant type. We next describe the OAuth 2.0 protocol when using the implicit grant type.

1) The Client redirects the Resource Owner to the Authorisation Server, requesting access to private data. The redirect includes the Client identifier, the scope of the requested access, an optional state parameter, and a redirection URL to which the Authorisation Server will redirect the Owner once access is granted (or denied). The state parameter is set to an unguessable value [9].
2) The Authorization Server validates the received request, ensuring that all the required parameters are present and valid. The Authorisation Server verifies the Client identity by comparing the provided redirection URL with the Client URL previously registered. If valid, the Authorisation Server (if necessary) authenticates the Resource Owner over a TLS-protected channel; the authentication method used is outside the scope of OAuth. It then asks the Resource Owner whether the Client's access request is authorised (where this query includes the Client identifier and the scope of the requested access). If all the checks succeed, the protocol continues.
3) The Authorisation Server redirects the Resource Owner's UA back to the Client using the redirection URL provided earlier (see step 1). The redirection URL includes the access token in a URL fragment. If the state parameter was present in step 1, the Authorisation Server must return it unmodified (in the URL) to the Client to protect against attacks involving manipulated URIs and malicious redirections, notably CSRF (Cross-Site

Request Forgery) attacks [11]. The UA follows the redirection instructions by making a request to the Client, excluding the access token-bearing fragment, although the UA retains the fragment information locally.

4) The Client returns a web page containing an embedded script capable of accessing the full redirection URL, including the fragment retained by the UA (see step 3). The UA executes the embedded script, which extracts the access token from the URL fragment and passes it to the Client over a secure, TLS-protected channel.
5) Finally, the Client uses the access token to securely retrieve the required resource(s) from the Resource Server.

3) Facebook Connect: Facebook Connect [12] implements the OAuth 2.0 standard, providing a single sign-on service. Facebook Connect (http://developers.facebook.com/docs/authentication/) allows end-users to sign on to applications (e.g. Facebook-affiliated websites) using their Facebook account, and also enables such applications to access Facebook-hosted user data, subject to user authorisation.

IV. The Integration Scheme

We now describe the novel scheme. The parties involved are a CardSpace-enabled RP, a CardSpace-enabled UA (e.g. a web browser), a browser extension implementing the protocol described below, and an OAuth Resource and Authorisation server; for simplicity, we assume that the roles of both the Resource and the Authorisation Servers are performed by a single entity, which we refer to as the 'OAuth IdP'.

The scheme has the following operational requirements.

- The user must have an existing relationship with both a CardSpace RP and an OAuth IdP (and so the IdP will have a means of authenticating the user).
- The user must register the RP with the IdP, in a user-specific manner. This involves the user interacting (via the UA) with an HTML registration page hosted by the IdP, and using this page to send the IdP the RP's name, URL, and (optionally) locale. The IdP then issues an RP-user-specific identifier, where the 'identifier' is used by the browser extension to identify the RP to the IdP.
- Prior to, or during, use of the protocol, the user must create a personal card, referred to here as an *OAuthCard*. This OAuthCard must contain the following data items in specific fields (the choice of which is implementation-specific): the URL of the IdP; the RP's identifier (as issued by the IdP); and a predefined sequence of characters (e.g. 'OAuth'), used to trigger the browser extension.
- The RP must not employ an STS. Instead, the RP must express its policy using HTML/XHTML, and interactions between the selector and the RP must be based on HTTP/S via a browser (a simpler and probably more common scenario for selector-RP interactions). This is because the scheme uses a browser extension.
- The RP must accept an unsigned (CardSpace-like) SAML token which includes OAuth IdP-supplied attributes and the signed RSTR containing the card-RP-specific PPID.

The protocol operates as follows. Steps 1, 2, and 4–7 are the same as steps 1, 2, and 3–6, respectively, of the personal card protocol given in section III-A2.

3) Extension $\rightarrow$ UA. The extension performs the following.
   a) It scans the login page to detect whether the RP website supports CardSpace; if so, it proceeds.
   b) It examines the RP policy to check whether the use of personal cards is acceptable. If so, it proceeds; otherwise it terminates, allowing CardSpace to operate normally.
   c) It keeps a local copy of any RP-requested claims.
   d) It determines the communication protocol (HTTP or HTTPS) in use with the RP[1].
   e) If necessary and if HTTP is in use, it modifies the RP policy to include the types of claim employed in the OAuthCard. For example, if the URL of the IdP is stored in the *web page* field of the OAuthCard, then it must ensure that the RP policy includes the *web page* claim. Adding the claim types to the policy ensures that the RSTR supplied by the SIIP contains the values of these claims, which can then be processed by the extension.
8) Selector$\rightarrow$ Extension/UA. Following the user submission of a suitable OAuthCard, the RSTR, unlike in the 'standard' case, does not reach the RP; instead the extension intercepts it and temporarily stores it. If the RP uses HTTP, the extension uses the contents of the RSTR to construct an OAuth request which it forwards to the IdP, having discovered its address from the RSTR. If the RP uses HTTPS, the extension first asks the user whether the use of the integration protocol is required. If not, it terminates, thereby allowing CardSpace to operate normally. If so, the extension prompts the user to enter the IdP's URL and the RP's identifier. The extension could offer the option to store the user-supplied values for future logins at this RP. Precisely as in the HTTP case, the extension then constructs an OAuth request.
   In both cases (i.e. HTTP and HTTPS), the 'implicit' grant type is adopted. Also, in both cases the OAuth request includes: the 'redirect_uri' parameter, to which the IdP must later redirect the UA; the 'scope' parameter, showing the scope requested[2]; and the 'state' parameter.
9) OAuth IdP $\rightleftharpoons$ User. This step is the same as step 2 of the OAuth 2.0 protocol given in section III-B2.
10) OAuth IdP $\rightleftharpoons$ Extension/UA. The IdP redirects the UA

[1]Note that the protocol operates slightly differently depending on whether the RP uses HTTP or HTTPS. This is because, if HTTPS is used, the selector will encrypt the RSTR message using the site's public key, and the browser extension does not have access to the corresponding private key. Hence, it will not know whether to trigger the integration protocol, and will be unable to obtain the OAuth IdP's URL and the RP's identifier; such issues do not occur if HTTP is used, since the selector will not encrypt the RSTR.

[2]This scope parameter indicates the RP-requested user attribute types (if any) which are to be provided by the OAuth IdP. The browser extension will know what they are since they were stored by it in step 3c. The 'scope' parameter helps the IdP determine the scope of the access request; the IdP must ask the user to authorise the release of the requested attribute values.

back to the provided RP URL, including the access token (in the URL fragment) and the 'state' parameter[3]. The extension reads and uses the provided access token to request and retrieve the RP-required user attribute values from the IdP via online communication with the IdP. The UA-IdP communication channel is TLS-protected.

11) Extension/UA $\rightarrow$ RP. Having retrieved the required user attribute values from the IdP, the browser extension constructs a 'CardSpace-like' SAML token and submits it to the RP. Such a token includes the IdP-supplied user attributes and the digitally-signed, SIIP-issued RSTR (which contains the PPID), allowing the RP to verify the SIIP signature (see also section V-A).
12) RP $\rightarrow$ User. The RP verifies the SAML token (including verifying the RSTR signature, PPID, nonce, timestamps, etc.), and, if satisfied, grants access.

V. Discussion and Analysis

A. Security Considerations

1) Properties: The scheme mitigates the risk of phishing. This is because the redirect to the OAuth IdP is initiated by the browser extension and not by the RP, i.e. the RP cannot redirect the user to an OAuth IdP of its choosing. By contrast, in OAuth, as it is typically used, a malicious site could redirect a user to a fake IdP, which could capture user credentials [13].

The unsigned (extension-generated) SAML token in step 11 of section IV (referred to here as the 'user token') includes the PPID, the OAuth IdP-supplied user attributes, and the digitally-signed, SIIP-issued, RSTR. The RP compares the SIIP-asserted PPID (and the public key) in the user token with its stored values and verifies the digital signature (see section III-A). If the RSTR contains user attributes, the RP could compare these (locally-stored) attributes with the (remotely-stored) IdP-provided attributes; such a procedure could give the RP added guarantees about the accuracy of these attributes.

It is infeasible for a malicious entity to fabricate a user token to masquerade as a legitimate party since it will not have access to the PPID and the private key necessary to sign the RSTR, both of which are only available if the appropriate InfoCard is selected on the correct platform.

Note that, in protocol step 4, the selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user's permission to proceed (see section III-A2). This helps to support mutual authentication since the user and the RP are both identified to each other.

Finally, the scheme allows the user attributes to be stored remotely at the OAuth IdP; this has potential security advantages over storing the attributes locally on the user machine, as is currently the case with CardSpace SIIP-issued attributes.

[3]The extension checks that the value in the state parameter is the same as the one it generated in step 8, is sufficiently current, and has not been used. The extension adds the received value to a list for use in future verifications.

2) Concerns: If the web browser is compromised, then an adversary could steal the user token and use it to impersonate the user. If the RP does not use HTTPS, then the SIIP-issued RSTR will not be encrypted. Also, if we assume that the browser is not a secure environment, it may be possible for a malicious plug-in or other malware to get access to sensitive information present in the (plaintext) RSTR, the extension-generated SAML token, or the OAuth IdP-issued access token. However, the same risks apply when manually entering credentials (e.g. username-password) into a browser.

3) Additional Security Properties: In certain circumstances, the RP can gain additional assurance in the identity of the user through use of the scheme proposed in section IV. If the RP trusts that the correct browser extension is running unmodified on the user platform, then the RP will know that the user has been authenticated by an OAuth IdP, and that the attributes have been provided by a genuine IdP. In such a case, the scheme would provide a two-level user authentication, based on selection of the correct InfoCard and user authentication at the OAuth IdP. This would offer a security advantage by comparison with the 'native' CardSpace personal card protocol, where the user is only authenticated once.

However, this is a significant trust assumption. We next consider two ways in which this assumption might be met.

1) The extension could be installed in a managed environment in which a user is only granted limited privileges insufficient to modify or replace the extension. However, in order for the RP to have assurance that the user platform is in such a controlled environment (and to avoid making extra changes to the RP), the RP itself would probably need to belong to the managed environment.
2) A more widely-applicable solution would be to make use of the functionality of the Trusted Platform Module (TPM), present on a large proportion of recently manufactured PCs. Using the remote attestation mechanism, an RP could be provided with guarantees about the software state of the user platform, including the presence of the expected integration software.

The SAML token created by the browser extension in step 11 of section IV could be extended to contain an additional field to indicate that the user has been authenticated by a specified OAuth IdP (as well as when and how). The RP would need to be modified to be able to process such an extra field, although this is likely to be relatively straightforward.

B. Client-side Integration

IdPs/RPs might not accept the burden of supporting two identity management systems, unless there is a financial incentive. A client-based integration technique would therefore be practically useful and would not affect server-performance.

C. Attribute Mapping

CardSpace and Facebook Connect use two different sets of attribute types[4]; a mapping would therefore be required.

[4]As stated in section III-A2, CardSpace personal cards only support fourteen editable attributes, whereas Facebook Connect supports many more.

Table I gives an example mapping.

TABLE I
CARDSPACE-FACEBOOK CONNECT ATTRIBUTE MAPPING

CardSpace (Personal Cards)	OAuth (Facebook Connect)
givenname	first_name
surname	last_name
emailaddress	email
dateofbirth	birthday
gender	gender
country	locale
city	location
web page	website

VI. PROTOTYPE REALISATION

We now describe a prototype implementation. The description applies to Facebook Connect, an implementation of OAuth 2.0. The prototype uses Facebook Connect's client-side flow (i.e. the implicit 'grant' type). It is coded in JavaScript, chosen because its wide adoption should simplify the task of porting the prototype to a range of other browsers. It uses the Document Object Model (DOM) to inspect and manipulate HTML pages and XML documents. The JavaScript code is executed using a C#-driven browser helper object (BHO), a Dynamic-link library (DLL) module designed as a plug-in for IE; once installed, the BHO attaches itself to IE, thus gaining access to the page's DOM. The prototype can be enabled or disabled using the add-on manager in the IE's Tools menu. The prototype operates with the CardSpace and the Higgins (http://wiki.eclipse.org/GTK_Selector_1.1-Win) identity selectors without any modification. It has been tested with Facebook and an implementation of a CardSpace RP.

A. Prototype Operation

Prior to use, the user must have accounts with an RP and Facebook. The user must register the RP with Facebook (http://developers.facebook.com/setup/). The user must also create an OAuthCard, inserting the RP's identifier in the *first name* field, and the trigger word 'OAuth' in the *last name* field. The Facebook URL is contained in the source code of the browser extension. The user can give the OAuthCard a meaningful name and can also upload an image for the card. We next consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section IV.

In step 3 the plug-in uses the DOM to perform the following.

3.1 It scans the web page in the following way[5].

(a) It searches through the HTML elements of the web page to detect whether any HTML forms are present. If so, it searches each form, scanning through each of its child elements for an HTML object tag.

(b) If an object tag is found, it retrieves and examines its type. If it is of type 'application/x-informationCard' (which indicates website support for CardSpace), it continues; otherwise it aborts.

(c) It retrieves and stores in a cookie the name attribute of the CardSpace object tag.

(d) It searches through the param tags (child elements of the retrieved CardSpace object tag) for the 'issuer' tag and examines its value; if it is 'http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self', indicating that the use of personal cards is acceptable, it continues.

(e) It retrieves the 'requiredClaims' and 'OptionalClaims' tags from the param tags, and stores the mandatory/optional claim types listed in these tags.

(f) It uses the property 'document.location.protocol' to discover whether HTTP or HTTPS is in use.

(g) If necessary and if HTTP is in use, the plug-in, after keeping track of the original policy settings, modifies the RP policy so that the *first name* and *last name* claim types are specified in the 'requiredClaims' tag.

3.2 It adds a JavaScript function to the head section of the HTML page to intercept the RSTR.

3.3 It obtains the action attribute of the CardSpace HTML form and stores it in a cookie. This attribute specifies the URL address of a web page at the RP to which the security token must be forwarded for processing.

3.4 It changes the current action attribute of the CardSpace form to point to the newly created 'interception' function.

In step 8 the plug-in uses the DOM to perform the following.

8.1 It intercepts the RSTR using the added function.

8.2 It operates slightly differently depending on whether HTTP or HTTPS is in use.

- If HTTP is used, the plug-in parses and extracts certain RSTR contents. If the *last name* field contains the word 'OAuth', the plug-in proceeds; if not, normal operation of CardSpace resumes. It reads the *first name* field to discover the RP's identifier.
- If HTTPS is used, the plug-in (using a JavaScript pop-up box) asks the user whether the use of the integration protocol is required. If so, it proceeds; otherwise it terminates. On proceeding, it prompts the user to enter the RP's identifier (as issued by Facebook). The plug-in offers the option to store the supplied values in a persistent cookie for future logins at this RP, using a plug-in-embedded checkbox.

8.3 It constructs an OAuth request, compatible with Facebook Connect. This involves generating a nonce and timestamp (used to build the 'state' parameter), and also determining the required and optional attribute types to be requested from Facebook. The plug-in retrieves all the CardSpace-supported claim types it stored earlier (see step 3.1 (e) above). It then maps between them and the Facebook-supported attribute types, using Table I. The mapping is done using JavaScript regular expressions. The plug-in sets the value of 'redirect_uri' parameter to the URL of the visited RP page. In addition, it sets the value of the 'response_type' parameter to 'token',

[5]Two HTML extension formats can be used to invoke the selector from a web page [14], both of which involve placing the CardSpace object tag inside an HTML form. This motivates the choice of the web page search method [15], [16], [17].

signifying the use of the 'implicit' grant type.

8.4 It encrypts (and stores in a cookie) the RSTR and the value of the 'state' parameter using AES in CBC mode, using a secret key known only to the plug-in.

8.5 It redirects the user to Facebook along with the OAuth request, using the JavaScript property 'window.location'.

In step 10 the plug-in performs the following steps.

10.1 It parses the Facebook (URL-embedded) response.

10.2 It (transparently) validates the response, including checking that the value of the 'state' parameter is the same as the one it generated in step 8.3, is sufficiently current, and has not been used. It adds the received value to an internally stored list for future verifications.

10.3 It uses the provided access token to request and retrieve the RP-requested user attribute values from Facebook open graph (http://en.wikipedia.org/wiki/Social_graph#Open_Graph) using a TLS-protected channel.

In step 11 the plug-in performs the following steps.

11.1 It constructs a CardSpace-like SAML security token, inserting the user attribute values received from Facebook into the token. It also embeds the signed SIIP-issued RSTR into the SAML token, after retrieving and decrypting the RSTR from the appropriate cookie (see step 8.4).

11.2 It creates and appends an 'invisible' form (with the method attribute set to 'POST') to the current page.

11.3 It writes the entire SAML token as a hidden variable into the invisible HTML form, with the name attribute of this variable set to the CardSpace object tag's name.

11.4 It writes the end-point URL of the RP into the action attribute of the HTML form.

11.5 Finally, it auto-submits the HTML form (transparently to the user), using the JavaScript method 'submit'.

B. Potential Issues

The plug-in must scan every HTML web page to see whether it supports CardSpace, and this may affect system performance. However, informal tests on the prototype suggest that this is not a serious issue. In addition, the plug-in can be configured so that it only operates with certain websites.

Some older browsers (or browsers with scripting disabled) may not be able to run the plug-in, as it was built using JavaScript. However, most modern browsers support JavaScript (or ECMAscript), and hence building the prototype in JavaScript is not a major usability obstacle.

VII. Conclusions and Future Work

We have proposed and prototyped a means of interoperation between two leading identity management systems, namely CardSpace and OAuth. CardSpace users (indeed, users of any Information Card system) are able to obtain an assertion token from an OAuth provider, the contents of which can be processed by a CardSpace-enabled relying party. The scheme is transparent to OAuth providers and identity selectors, uses a browser extension, and requires only minor changes to relying parties. It uses the identity selector interface and personal cards to enable the interoperation.

Planned future work includes investigating the possibility of extending the CardSpace identity selector to simultaneously support security tokens from a variety of identity providers, such as OpenID, OAuth, Liberty, Shibboleth, as well as CardSpace remote and self-issued identity providers. Finally, a full version of this paper is available as a technical report [18].

Acknowledgements

The helpful remarks provided by Chris Mitchell are gratefully acknowledged. The author is sponsored by the Diwan of Royal Court, Sultanate of Oman.

References

[1] E. Bertino and K. Takahashi, *Identity Management: Concepts, Technologies, and Systems*. Artech House Publishers, Norwood, MA, 2011.

[2] E. Hammer-Lahav, editor, *The OAuth 1.0 Protocol*, RFC 5849, 2010.

[3] M. Mercuri, *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, 2007.

[4] H. S. Al-Sinani, W. A. Alrodhan, and C. J. Mitchell, "CardSpace-Liberty integration for CardSpace users," in *Proceedings of IDtrust '10 — the 9th Symposium on Identity and Trust on the Internet, Gaithersburg, USA*, K. Klingenstein and C. M. Ellison, Eds. ACM, 2010, pp. 12–25.

[5] H. S. Al-Sinani and C. J. Mitchell, "CardSpace-Shibboleth integration for CardSpace users," in *ACNS '11 [industrial track proceedings], 9th International Conference on Applied Cryptography and Network Security, Nerja (Malaga), Spain, 7–10 June 2011*, 2011, pp. 49–66.

[6] ——, "Client-based CardSpace-OpenID interoperation," in *Proceedings of ISCIS'11 — the 26th International Symposium on Computer and Information Sciences, London, UK*, E. Gelenbe, R. Lent, and G. Sakellari, Eds. Springer (LNEE), 2011, pp. 387–393.

[7] I. Jørstad, et al., "Bridging CardSpace and Liberty Alliance with SIM authentication," in *Proceedings of ICIN '07, International Conference on Intelligence in Next Generation Networks*. Adera, 2007, pp. 8–13.

[8] M. B. Jones and M. McIntosh, editors, *Identity Metasystem Interoperability Version 1.0 (IMI 1.0)*, OASIS Standard, 2009.

[9] E. Hammer-Lahav, D. Recordon, and D. Hardt, editors, *The OAuth 2.0 Authorization Protocol — draft-ietf-oauth-v2-20*, 2011.

[10] S. Pai, Y. Sharma, S. Kumar, R. Pai, and S. Singh, "Formal verification of OAuth 2.0 using Alloy framework," in *Proceedings of CSNT '11 — the International Conference on Communication Systems and Network Technologies*. IEEE Computer Society, 2011, pp. 655–659.

[11] T. Lodderstedt, editor, *OAuth 2.0 Threat Model and Security Considerations — draft-ietf-oauth-v2-threatmodel-00*, 2011.

[12] M. Miculan and C. Urban, "Formal analysis of Facebook Connect single sign-on authentication protocol," in *SOFSEM '11 (Software Seminar): Theory and Practice of Computer Science — the 37th Conference on Current Trends in Theory and Practice of Computer Science. Proceedings of Student Research Forum*, 2011, pp. 99–116.

[13] H. S. Al-Sinani and C. J. Mitchell, "A universal client-based identity management tool," in *Proceedings of EuroPKI '11 — the 8th European Workshop on Public Key Infrastructures, Services and Applications, Leuven, Belgium*. Springer-Verlag (LNCS), (to appear), 2011.

[14] M. B. Jones, *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*, Microsoft, 2008.

[15] H. S. Al-Sinani and C. J. Mitchell, "Using CardSpace as a password manager," in *Proceedings of IFIP IDMAN '10, the 2nd IFIP Working Conference on Policies and Research in Identity Management*, E. de Leeuw, S. Fischer-Hübner, and L. Fritsch, Eds., vol. 343. Springer, Boston, 2010, pp. 18–30.

[16] ——, "Extending the scope of CardSpace," in *Proceedings of SIN '11, the 4th International Conference on Security of Information and Networks, Sydney, Australia*. ACM (to appear), 2011.

[17] ——, "Enhancing CardSpace authentication using a mobile device," in *Proceedings of DBSEC '11, the 25th IFIP Conference on Data and Applications Security and Privacy, Richmond, USA, 11-13 of July 2011*, Y. Li, Ed., vol. 6818. Springer (LNCS), 2011, pp. 201–216.

[18] H. S. Al-Sinani, *Browser Extension-based Interoperation Between OAuth and Information Card-based Systems*, Technical Report: RHUL–MA–2011–15 (Department of Mathematics, Royal Holloway, University of London), 2011, http://www.ma.rhul.ac.uk/static/techrep/2011/RHUL-MA-2011-15.pdf.