Invoking Browser-based Application Extensions

Background: There is a fairly large set of security-related applications that use a browser as an initiation vehicle. In this category of applications we find form-based signatures, on-line key generation schemes, and various kinds of authentication mechanisms. However, the way such applications are invoked from a browser is all over the map making the process creating interoperable solutions extremely slow. This document describes some of the more well-known methods including their pros and cons. Finally, a generic solution is proposed as a possible item for future standardization.

Generic Problem #1 – Finding out Browser Extension Capabilities

Not all browsers understand/implement a certain application extension. A future-proof solution should allow *query* of supported extensions *including application extension versioning*. An example of a problematic scheme is the ECP (Enhanced Client Profile) featured in OASIS' SAML 2.0 [SAML2] where a conforming client is supposed to request a resource like the following:

```
GET /index HTTP/1.1
Host: identity-service.example.com
Accept: text/html; application/vnd.paos+xml
PAOS: ver='urn:liberty:paos:2003-08' ; 'urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp'
```

The consequence of this scheme is that *all* requests must be augmented with the PAOS attribute which unfortunately also is only valid for the ECP extension.

Using the HTML "Object" Tag

In theory the HTML Object tag would be ideal for introducing new functionality in browsers but for various reasons ranging from an excessively complex definition, to the fact that *most of the sought applications do not have any relation to a particular area on an HTML form*, it seems that this method is mainly suited for "Flash" and similar *media extensions*.

Using XHTML/XML Extensions

In Microsoft's recently introduced CardSpace® [CARDSPACE] authentication scheme, MSIE has been augmented with a set of XHTML/XML extension objects. An issue with embedded XHTML/XML extensions is that these presumably only can be implemented by browser vendors.

Using JavaScript Objects

In Mozilla there are two security-related extensions implemented as built-in JavaScript methods, `generateCRMFrequest()` and `signText()`. They work fine but *as far as I know there is no way to add intrinsic JavaScript objects for mere mortals, making this extension scheme fairly non-generic*. In addition, the today almost ubiquitous way of defining data (using XML), isn't particularly pleasant in conjunction with JavaScript objects although it is certainly workable.

Using Netscape's Plugin API

Netscape's Plugin API is not supported by recent versions of Microsoft's Internet Explorer. That doesn't disqualify it but the world is slowly leaving C in favor of .NET, Java and similar so it would require a major upgrade to become useful as a standard.

Conclusion

The web clearly deserves a new *application-oriented* extension scheme that is aligned with the actual needs and may be supported in such a way that it doesn't hamper new developments.

Browser Application Extension Proposal - XBPP

Resurrection of the MIME-type Application Extension

Internet browsers have since the very beginning been able to associate MIME-types with applications. This proposal is based on the idea of introducing a single new MIME-type supporting any number of registered extensions. This serves a number of purposes:

- Enables an optional extension query scheme
- Does not require a “bootstrapping” HTML page and associated *input format constraints*
- Supports application extension versioning making *stepwise migration possible*
- Potentially opens a cleaner extension object mechanism not requiring C/C++
- Is independent of possible HTML developments

The additional MIME-type is tentatively called `application/xbpp+xml` which should be interpreted as Xml Browser Protocol Plugin (XBPP).

A compliant user-agent SHOULD include this MIME-type in the “Accept” header of all standard HTTP requests to indicate its support for the application extension scheme.

Extension Registry

Each extension consumer (browser application extension) MUST register itself to the proposed mechanism in some way (outside the scope of this document) in order to be recognized by the browser. The registry should hold an XML namespace URI [URI] and a version uniquely identifying the extension.

Pre-registered Enquiry Extension

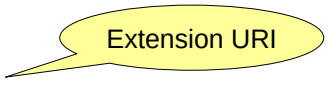
To facilitate extension support queries, a compliant user-agent SHOULD support a pre-defined enquiry extension (URI: TBD) that would return a list of matching extension URIs and versions based on an input using regular expressions. The exact details are yet TBD. *The query interface should preferable also be available from JavaScript*.

Sample Application

The following section shows how a *sample* browser application extension called WASP (Web Activated Signature Protocol) could be implemented using the proposed extension scheme.

Assume that you want a user to sign the text "Hello signature world!". To do that the requesting service (a web-server application), may as a minimum create a (WASP-specific) `SignatureRequest` object like below:

```
<?xml version="1.0" encoding="UTF-8"?>
<SignatureRequest ID="_10d16a170d97ddd1a7024f7d9ee"
                  SubmitURL="https://example.com/submit"
                  xmlns="http://xmlns.webpki.org/wasp/1.0/core#">
  <DocumentReferences>
    <MainDocument ContentID="cid:d0@example.com" MimeType="text/plain"/>
  </DocumentReferences>
  <DocumentData>
    <Text ContentID="cid:d0@example.com">Hello signature world!</Text>
  </DocumentData>
</SignatureRequest>
```



When this structure is returned (as the response of an arbitrary HTTP GET or POST operation invoked by the *user*), to a compliant browser using the XBPP MIME-type `application/xbpp+xml`, the browser should *automatically* respond with a signature dialog or similar showing the text to be signed:

If the user carries out the signature process, a `SignatureResponse` object like the following is POSTed to the `SubmitURL` specified in the `SignatureRequest` object:

```
<?xml version="1.0" encoding="UTF-8"?>
<SignatureResponse xmlns="http://xmlns.webpki.org/wasp/1.0/core#">

        …Other elements removed for brevity

</SignatureResponse>
```

Note that the POSTed signature object is like any other user-originated browser-to-server invocation, which means that the server should preferably respond with a page showing a note to the user that the signed object has been successfully received (and presumably also validated).

From standards point of view only the invocation part of this sample is really applicable, although I believe there are a number of support functions required as well in order to create actual applications. Typically you want to spawn dialog windows, POST data, as well as parsing and validating XML.

Anders Rundgren
WebPKI.org

Version 0.12, August 10, 2008

References

[SAML2] http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf