

# 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 *embedded media extensions*.

## Using Specific HTML Tags

So far the only application-specific scheme seems to be Netscape's (nowadays adopted by W3C's HTML5 WG) `<keygen>` HTML tag. This method has quite a few limitations but cannot be extended except by a major browser upgrade (and associated HTML redefinition) that only browser vendors can realistically succeed with.

## Using XHTML/XML Extensions

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

*CardSpace could without any functional degradation use the universal XML invocation method presented in this document.*

## Using JavaScript Objects

In Firefox there are two security-related extensions implemented as built-in JavaScript methods, `generateCRMFrequest()` and `signText()`. They work fine but mixing XML schema processing with JavaScript isn't particularly pleasant (usually requires BASE64-encoded XML). However, it *is* workable.

*A specific JavaScript extension invocation object though appears to be a possible “Plan B” if the proposed MIME-scheme is found too hard to support.*

## Generic Problem #2 – Execution of Multi-phase Protocols

Not all schemes can be covered by a single HTTP request/response pair which greatly limits the applicability of existing extension schemes. As an example key generation schemes typically require 4-10 message exchanges. Microsoft's CertEnroll [CERTENROLL] shows that running multi-phase security protocols from *untrusted downloaded code* tends to introduce additional security settings as well as making privacy-related questions pop-up (due to an increased exposure of sensitive data) that users may not always understand how to deal with.

## Using Netscape's Plugin API

This *native-mode* extension mechanism offers almost unlimited functionality. However, Netscape's Plugin API is not supported by recent versions of Microsoft's Internet Explorer. That doesn't disqualify it but the *mainstream* programming world is rapidly leaving C/C++ in for JavaScript, C#, Java and similar so it would require a major upgrade to become useful as a standard.

*The ability creating platform-independent extensions is crucial for adoption, particularly for browsers supporting multiple operating systems and processor architectures.*

## 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 - BPP

## 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 new MIME-type + a URI namespace argument supporting any number of registered extensions. This serves a number of purposes:

- Requires a single IANA registration
- URIs have become the norm for identifying globally unique objects
- Does not require a “bootstrapping” HTML page and associated *input format constraints*
- Supports application extension *versioning* making stepwise migration possible
- Enables an extension query scheme
- Potentially opens a cleaner extension object mechanism not requiring C/C++
- Is independent of HTML developments

The MIME-type is tentatively called `application/vnd.webpki.bpp` which should be interpreted as Browser Protocol Plugin (BPP).

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.

To enable a specific namespace URI (=plugin) a new HTTP header MUST also be specified in the HTTP response body. This header is tentatively called `X-WebPKI-Bpp-Ns`.

## Extension Registry

Each extension consumer (plugin) 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 a namespace URI [URI] and a version uniquely identifying the extension.

## Extension Query Interface

To facilitate extension support queries, a compliant user-agent MUST support a predefined JavaScript extension query object that returns a list of matching extension URIs and versions based on an input using regular expressions. The exact details are yet TBD.

## Private Extensions

In case the creator of an extension does not want parties outside of the target audience be aware of if their users have the extension installed or not, an extension MAY be marked as private which will remove it from extension query results. See previous section.

## Support for non XML Formats

Although the extension is primarily intended for XML protocols, the extension can *without modifications* also be applied to ASN.1 and JSON-based schemes.

## Handling Unsupported Extensions

If the actual extension namespace is unsupported, the browser should abort the processing of the extension and return a message to the user.

## 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* return a (WASP-specific) `SignatureRequest` object like below as the response to an arbitrary HTTP GET or POST operation invoked by the *user*.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.webpki.bpp
Content-Length: 490
X-WebPKI-Bpp-Ns: http://xmlns.webpki.org/wasp/1.0/core#

<?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>
```

If the browser supports this extension it should read the payload and invoke the extension which should *automatically* respond with a signature dialog or similar showing the text to be signed:



If the user carries out the signature process (including signature key selection not shown here), 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 would typically respond with an HTML page showing a note to the user that the signed object has been successfully received (and presumably also validated).

*From a standards point of view only the invocation part of this sample is really applicable, although 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 0.41, June 18, 2011.

## References

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