Oasis Security Services Bindings Model

Prateek Mishra
Chris Ferris
Jeff Hodges
Evan Prodromou

draft-sstc-bindings-model-04.doc
14-June-2001

comments to: security-bindings@lists.oasis-open.org

This document is an OASIS-Draft and is [largely] in conformance with relevant OASIS SSTC document standards as described in draft-sstc-doc-guidelines-00.txt.

1 Introduction

1.1 Scope

Other Oasis Security Services TC subcommittees (e.g. Core Assertions and Protocol) are producing a specification of SAML security assertions and one or more SAML request-response message exchanges.

The high-level goal of this document is to specify how:

(1) SAML request-response message exchanges are mapped into standard messaging or communication protocols. Such mappings are called SAML *protocol bindings.* An instance of mapping SAML request-response message exchanges into a specific protocol <FOO> is termed a *SAML <FOO> binding*.

Example: A SAML HTTP binding describes how SAML Query and Response message exchanges are mapped into HTTP message exchanges. A SAML SOAP binding describes how SAML Query and Response message exchanges are mapped into SOAP message exchanges.

(2) SAML security assertions are embedded in or combined with other objects (e.g. files of various types, protocol data units of communication protocols) by an originating party, communicated from the originating site to a destination, and subsequently processed at the destination. A set of rules describing how to embed and extract SAML assertions into a framework or protocol is termed a *profile* for SAML. A set of rules for embedding and extracting SAML assertions into a specific class of <FOO> objects is termed a *<FOO> profile* for SAML.

Example: A SOAP profile for SAML describes how SAML assertions may be added to SOAP messages, the interaction between SOAP headers and SAML assertions, list of SAML-related error states at the destination.

(1) and (2) MUST be specified in sufficient detail to yield interoperability when independently implemented.

1.2 Contents

The remainder of this document is in four sections:

- General guidelines for the specification of protocol bindings and profiles. The intent here is to provide a checklist that MUST or SHOULD be filled out when developing a protocol binding or profile for a specific protocol or framework.

- A process framework for describing and registering proposed and future protocol bindings and profiles.

- Protocol bindings for selected protocols. Bindings MUST be specified in enough detail to satisfy the inter-operability requirement.

- Profiles for selected protocols and frameworks. Profiles MUST be specified in enough detail to satisfy the inter-operability requirement.

1.3 Guidelines for Specifying Protocol Bindings and Profiles

Issues that MUST be identified in each protocol binding and profile:

(1) Each binding or profile must be characterized as set of interactions between parties. Any restriction on applications used by each party and the protocols involved in each interaction must be explicitly called out.

(2) Identification of parties involved in each interaction: how many parties are involved in the interaction? Can intermediaries be involved?

(3) Authentication of parties involved in each interaction: Is authentication required? What types of authentication are acceptable?

(4) Support for message integrity: what mechanisms are used to ensure message integrity?

(5) Support for Confidentiality: can a third party view the contents of SAML messages and assertions? Does the binding or profile require confidentiality? What mechanisms are recommended for securing confidentiality?

(6) Error states: characterization of error states at each participant, especially those that receive and process SAML assertions or messages.

(7) Support for *integrity of assertion attachment*. Many profiles consist of a set of rules for adding assertions to an existing protocol or packaging framework. These rules will be used by an originating party (e.g., user, server) to create a *composite package* consisting of assertions and a business payload for delivery to a destination. When the composite package arrives at the destination, the recipient will require proof (1) the originating party is the subject of the assertions contained within the composite package, (2) neither the assertion nor business payload have been altered.

The term *integrity of assertion attachment* refers to the linkage between the originating party, assertions and business payload, created when an originating party constructs the composite package. Integrity of assertion attachment MUST be verifiable by a recipient. Typically, mechanisms provided to support attachment integrity will be based on some cryptographic techniques (hash or digital signature).

1.4 Process Framework for Describing and Registering Protocol Bindings and Profiles

When a profile or protocol binding is registered, the following information is supplied:

1. Identification: specify a URI that authoritatively identifies this profile or protocol binding.

2. Contact information: specify the postal and electronic contact information for the author of the profile or protocol binding.

3. Description: the description MUST follow the guidelines for profiles and protocol bindings given above.

4. Updates: references to previously registered profiles or bindings that the current entry improves or obsoletes.

*Issue: Where should this registry be maintained? It has been proposed that IANA (*http://www.iana.org*) might provide an appropriate forum. Further investigation is required..*

2 Protocol Bindings

2.1 HTTP

2.1.1 Introduction

HTTP is among the most commonly-used Internet application protocols today. There are any number of implementations of the protocol that allow rapid development of dynamic servers or clients. With the possible exception of SMTP mail servers, HTTP servers withstand the greatest collective load, in terms of performance, stability, and security, of any other class of software. For these reasons -- widespread use, robust implementations, and diverse development platforms -- it makes sense to leverage HTTP, and HTTP software, for the exchange of SAML messages.

The following binding description derives from the HTTP binding provided with [AuthXML]. Note that the current version of SAML [draft-sstc-core-07.doc] has two different message formats, which will probably change over time. For this reason, this section merely refers to them as "request messages" and "response messages" without particular information about the content or structure of the message.

Note that this section does not treat the issue of passing SAML assertions or assertion tokens from a standard Web browser to a Web server. Instead, it concentrates on using HTTP as a transport layer for SAML messages, without the restrictions that standard Web browsers impose. In most cases, this binding will be used as a service-to-service binding, rather than a user-to-service binding.

Some design goals of this binding are as follows:

* Enable using existing HTTP software (Web servers, client libraries) to create SAML services.
* Minimize requirements for supporting the somewhat complex HTTP protocol.
* Minimize the information carried in HTTP headers and other data. Except in extreme situations, information should be passed as SAML.

Readers of this document should be familiar with HTTP/1.1, which is described in [RFC2616].

2.1.2 Overview

The message protocol for SAML is based on a request-response metaphor. This naturally maps to the HTTP request-response method. So, for most types of interaction between systems, a request message is sent as an HTTP request, and a response message is sent as an HTTP response. There are two parties involved in the interaction: a requester and a responder. There is no provision for intermediaries in the current framework.

In the discussion that follows, the following terms are used:
* request message -or- request: A SAML request XML object.
* response message -or- response: A SAML response XML object.
* HTTP request: An HTTP request, as distinct from a SAML request.
* HTTP response: An HTTP response, as distinct from a SAML response.
* requester: The party sending the request.

* responder: The party sending the response.

2.1.3 HTTP Binding

2.1.3.1 Connections

As with all HTTP connections, the requester will initiate the connection. Connections MUST be one way. Multiple requests and corresponding responses MAY be sent over a single connection, per the HTTP 1.1 specification. The requester MUST only send requests through the connection, and the responder MUST only send responses through the connection.

The Connection header MAY be added to an HTTP request to request that the connection be closed after the response is given. "Connection: close" is the only allowed field in this header, in which case the responder MUST add the "Connection: close" header to the response and MUST close the connection after completing the response.

If the "Connection: close" header is not added to the request, the connection will be handled per the default for the HTTP version of the request. If the HTTP version of the request is 1.0, the connection will be automatically closed by the responder. If the HTTP version is 1.1, the connection will be maintained by the responder, unless a "Connection: close" header was added to the response (See section 2.1.3.3 below).

2.1.3.2 Request Messages

A request message is bound to an HTTP request.

The request MUST use the POST method. The HTTP version MUST be one of"1.0" or "1.1".

The request MUST have a Content-Type of "text/xml".

The content of the HTTP request MUST be exactly one request message. Additional content MUST NOT be included in the HTTP request.

The Host, Date, Content-Type and Content-Length headers MUST be provided in the HTTP request and be correct. A Connection header may be added as noted above in section 2.1.3.1.

Additional HTTP headers MAY be provided, but parties in the conversation MUST ignore those other headers.
[Rationale: many existing HTTP libraries will add additional headers to an HTTP request. The intent is to ensure a minimal number of headers required to handle the binding, without requiring that implementations write their own HTTP code.]

Content-Encoding or Transfer-Encoding schemes MUST NOT be used.
[Rationale: SAML messages are relatively small and should not require chunked encoding or compression. Forbidding Content- or Transfer-Encoding will allow implementers to safely ignore these fairly advanced and costly HTTP features.]

2.1.3.3 Response Messages

If a request can be handled and generates a response, the response will be bound to an HTTP response message. If the responder cannot or will not generate a SAML response, the responder MUST send one of the HTTP error responses defined in section 2.1.3.6. The rest of this section will treat only successful responses.

[Note that success, in this context, means that a SAML response was generated. It does not mean that the request was fulfilled or have domain level meaning, such as that authorization was granted, etc. The SAML response may have failure notifications per the SAML protocol.]

The HTTP response MUST have a status code of 200. The HTTP version MUST be one of "1.0", "1.1".

The response MUST have a Content-Type of "text/xml".

The content of the HTTP response MUST be exactly one response message. Additional content MUST NOT be included in the HTTP response.

The Host, Date, Content-Type and Content-Length headers MUST be provided in the HTTP response and be correct. A Connection header may be added as noted above in section 4.1.

Additional HTTP headers MAY be provided, but parties in the conversation MUST ignore those other headers.

Content-Encoding or Transfer-Encoding schemes MUST NOT be used.

2.1.3.4 Authentication and Message Integrity

Authentication of parties and message integrity of both requests and responses MUST be handled in one of two ways.

2.1.3.4.1 XML Signature

If this technique is used, an XML digital signature MUST be added to the entire request or response. The digital signature MAY be embedded in the message, or the message MAY be embedded in the signature.

2.1.3.4.2 HTTP/S with Certificates

Alternately, the HTTP conversation may be conducted over a Secure Sockets Layer (SSL) connection. In this case, both parties (requester and responder) MUST provide digital certificates for the SSL layer.

2.1.3.5 Message Confidentiality

HTTP/S MAY be used preserve message confidentiality. If authentication and message integrity is protected using XML Signatures, neither party is required to provide a digital certificate.

2.1.3.6 Errors

The following error messages may be sent by the responder for a SAML message. [Note that in the following section, the error text is not normative, but gives an indication of what the error code means. Only the error number is normative.]

For all status values besides "200", the "Connection: close" header MUST be sent, and the connection between requester and responder MUST be closed.

2.1.3.6.1 200 OK

The responder received the request and successfully generated a response. The response may contain a SAML error code or further SAML information. The meaning of the 200 message is "more info in SAML content."

2.1.3.6.2 400 Bad Request

The responder received the request, but the request was ill-formed in some way. The content of the Response is undefined, but it SHOULD NOT be a SAML message. The content of the Response MAY be a stock piece of HTML or plain text explaining the nature of the error.
[Rationale: Some HTTP server software will add stock explanations for error status codes.]
This result code is appropriate for requests with bad HTTP headers, HTTP methods other than "POST", or with syntactically incorrect SAML content.

2.1.3.6.3 403 Forbidden

The responder has received the request, but refuses to perform a SAML message exchange with the requestor. The content of the Response is undefined, but it SHOULD NOT be a SAML message. The content of the Response MAY be a stock piece of HTML or plain text explaining the nature of the request.

2.1.3.6.4 500 Internal Server Error

The responder has received the request but has failed to produce a response, due to internal error. The content of the Response is undefined, but it SHOULD NOT be a SAML message. The content of the Response MAY be a stock piece of HTML or plain text explaining the nature of the request.

2.2 SOAP 1.1

2.2.1 Introduction

SOAP (Simple Object Access Protocol) 1.1 is a standard proposed by Microsoft, IBM, and other contributors for RPC-like interactions using XML. It defines a mechanism for defining messages in XML, and for sending them through HTTP. Since its introduction, it has had increased attention, and it is expected to provide the foundation for many future Web-based services.

SOAP 1.1 has three main parts. One is a message format that uses an envelope and body metaphor to wrap XML data for transmission between parties. The second is a restricted definition of XML data for making strict RPC-like calls through SOAP, without using a predefined XML schema. Finally, it provides a binding for SOAP messages to HTTP and enhanced HTTP.

This document describes how to use SOAP to send and receive SAML messages. An additional section of the SAML specification ("SOAP Profile") defines how to use SAML as an authentication mechanism for SOAP. In other words, this section describes using SAML over SOAP, and that section describes using SAML for SOAP.

Like SAML, SOAP can be used over multiple underlying transports. This document does not address the use of underlying transports directly, although it makes recommendations for some transports in addressing message integrity and confidentiality concerns.

Note that this protocol binding is relatively short. This is because SOAP is a relatively simple protocol, and because most of the difficult details of connections, routing, etc. are defined in the SOAP 1.1 standard.

2.2.2 Overview

SOAP messages consist of three elements: an envelope, header data, and a message body. SAML messages (queries and responses) are enclosed in the SOAP message body.

SOAP 1.1 also defines an optional data encoding system. This system is not used for the SOAP protocol binding for SAML. This means that SAML messages can be transported using SOAP without re-encoding from "standard" SAML to a SAML-like SOAP encoding.

The system model used for SAML conversations over SOAP is a simple request-response model. A sending party sends a SAML query in the body of a SOAP message. The receiving party processes the SAML query and returns a SAML query response in the body of another SOAP message.

A brief glossary:

SAML conversation: an exchange of a SAML query and a SAML response.
sending party: The party sending a message.
receiving party: The party receiving a message.
querying party: The party sending a query message.
responding party: The party sending a response.

2.2.3 SOAP Binding

2.2.3.1 Namespaces

All SAML messages encoded in SOAP MUST include XML namespace qualifiers, as specified by the core assertions and messages definition.

[Rationale: Some SOAP message processors require a namespace. Also, the namespace prevents conflicts with other standards and schemata.]

2.2.3.2 Headers

The sending party in a SAML conversation over SOAP MAY add arbitrary headers to the SOAP message.

[Rationale: some SOAP software and libraries may add headers to a SOAP message that are out of the control of the SAML-aware process. Also, some headers may be needed for underlying protocols that require routing of messages.]

The receiving party MAY NOT require any headers for the SOAP message.

[Rationale: requiring extra headers will cause fragmenting of the standard and will hurt interoperability.]

2.2.3.3 SAML Queries

A SAML query is stored as the child of the <SOAP:body> element of a SOAP message. The querying party MUST send one SAML query. The querying party MUST NOT send more than one SAML query per SOAP message. The querying party MUST NOT include any additional XML elements in the SOAP body.

On receiving a SAML query as a SOAP message, the receiving party MUST return either a SAML query response (section 2.2.3.3) or a SOAP fault code (section 2.2.3.4).

2.2.3.4 SAML Query Responses

A SAML query response is stored as the child of the <SOAP:body> element of a SOAP message. The message MUST contain exactly one SAML query response. The querying party MUST NOT include any additional XML elements in the SOAP body.

On receiving a SAML query response in a SOAP message, the querying party MUST NOT send a fault code or other error messages to the sending party.

[Rationale: The format for the message interchange is a simple request-response. Adding additional error conditions, notifications, etc. would needlessly complicate the protocol.]

2.2.3.5 Fault Codes

If a responding party cannot, for some reason, process a SAML query, it should return a SOAP fault code. Fault codes MUST NOT be sent for errors within the SAML problem domain, e.g. as a signal that the subject is not authorized to access an object in an authorization query.

The four fault codes (VersionMismatch, MustUnderstand, Client, Server) defined by SOAP 1.1 are sufficient to define any SOAP-related errors. Responding parties MUST NOT use any additional fault codes, or sub-defined fault codes, in a fault response.

Responding parties MAY provide additional fault information, such as descriptions and details, as defined by SOAP.

[Rationale: some SOAP processors may add fault information automatically.]

2.2.3.6 Authentication and Integrity

2.2.3.6.1 XML Digital Signature

To ensure message integrity, the parties in a SAML conversation MAY add a XML Digital Signature to the SAML query. The parties MUST NOT add signatures in either the headers or the envelope of the SOAP message.

2.2.3.6.2 HTTP/S with Certificates

Alternately, the parties MAY use the underlying transport of the SOAP conversation to ensure message integrity. For SOAP messages sent over HTTP, this would be HTTP/S with client certificates.

2.2.3.7 Confidentiality

To achieve message confidentiality, the parties in a SAML conversation MAY use the confidentiality protection mechanism in the underlying SOAP transport. For SOAP messages used over HTTP, this would be HTTP/S.

3 Profiles

3.1 Web Browser

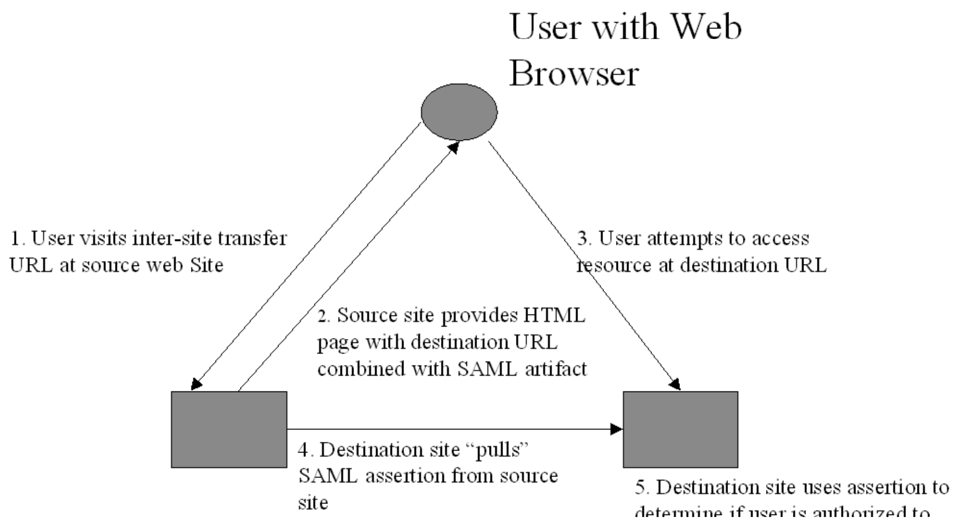
3.1.1 Overview

The user is utilizing a standard commercial browser and has logged onto the source web site. At some point, the user transitions to a destination site which supports single sign-on for users originating from the source site. In this situation, information about SAML assertions must be conveyed from one site to another through the browser.

The only general technique available is based on URL query strings; note that use of cookies requires that both the source and destination site belong to the same "cookie domain". While [RFC2616] does not specify any restrictions on URL length, in practice commercial web browsers and application servers impose constraints on URL size [Appendix A]. This suggests the use of some form of a "small" fixed-size SAML artifact, which can be reliably carried as part of the URL query string and thereby transferred from source to destination site. The destination site would then utilize information contained within the SAML artifact to "pull" a SAML assertion from the source site to the destination site.

3.1.2 Parties and Interactions

Figure 1: Single Sign-On (web browser)


User with Web Browser
1. User visits inter-site transfer URL at source web Site
2. Source site provides HTML page with destination URL combined with SAML artifact
3. User attempts to access resource at destination URL
4. Destination site “pulls” SAML assertion from source site
5. Destination site uses assertion to determine if user is authorized to

{PRIVATE "TYPE=PICT;ALT=Figure 1: SSO Diagram"}

The web browser profile involves a single interaction between three parties (source site, user equipped with a browser, destination site), with a nested sub-interaction between two parties (source site, destination site). We refer to the sub-interaction as an *assertion pull* interaction.

The user authenticates at the source web site and subsequently visits an inter-site transfer URL at the source web site (step (1)). As this step is over the open internet, confidentiality is required, and the inter-site transfer URL MUST be exposed over HTTPS (HTTP over server-side SSL).

The inter-site transfer URL at the source web site provides a HTML page (or re-direct link) to the user browser (step (2) which includes the destination URL combined with a SAML artifact. The SAML artifact is carried as part of the destination URL query string:

```
<destination> ::= https://destination_URL..?SAMLart=<artifact body>..
```

The first interaction completes when the user explicitly (or implicitly, if responding to a re-direct) attempts to access the destination URL (step (3)) and delivers both the destination URL and the SAML artifact to (a web server at) the destination site. As this step is over the open internet, confidentiality is required, and the destination URL MUST be exposed over HTTPS (HTTP over server-side SSL).

If the destination site is unable to process this information it MUST return a HTTP "400 Bad Request" error code to the browser. Otherwise, it MUST carry out the *assertion pull* interaction described below, obtain an assertion from the source site and make an access control judgement. If the user is refused access to the destination URL, it MUST return a HTTP “403 Forbidden” error code to the browser. Otherwise, the destination site should grant user access to the destination URL.

Summary of (User) Browser Interaction with Source and Destination Site		
Action	HTTP Request	HTTP Response
(1) User visits inter-site transfer URL	GET https://www.example.com/inter-site-transfer.html	Web page with destination site URL and artifact OR re-direct to destination site URL and artifact
(3) User accesses destination URL (or is re-directed to destination URL)	GET https://destination_URL…?SAMLart=<artifact body>…	Requested URL contents OR “400 Bad Request” OR “403 Forbidden”

The assertion pull interaction consists of a SAML message exchange between source and destination site (step (4)) utilizing a registered SAML protocol binding. The destination site sends a <SAMLQuery> message to the source site, which includes information adequate to identify a SAML assertion at the source site. If the source site can find the required assertion it responds with a <SAMLQueryResponse> message which includes the desired assertion within it. Otherwise, it returns an "assertion not found" error to destination site. The selected SAML protocol binding MUST support confidentiality.

3.1.3 SAML artifact structure

The exact format and size of the SAML artifact is somewhat implementation dependent.We would require the following properties from any implementation:

1. The SAML artifact must identify the source site to the destination site; the SAML artifact must identify the relevant assertion to the destination site.

2. The SAML artifact MUST be a "one-time use ticket"; once the user completes step (3) above, any repeated GET https://destination_URL…?SAMLart=<artifact body>…must fail and the destination site MUST return HTTP code “403 Forbidden”.

3. The SAML artifact MUST utilize adequate crypto so that it is difficult to forge.

4. The SAML artifact MAY be authenticated by the source web site.

We would expect there to be a large amount of variability in the design of artifact formats. This variability is accommodated by a mandatory two byte artifact type code in the proposed representation:

```
<SAML_artifact> :=
         B64 representation of <TypeCode> <Remaining artifact>
         <TypeCode> := Byte1Byte2
```

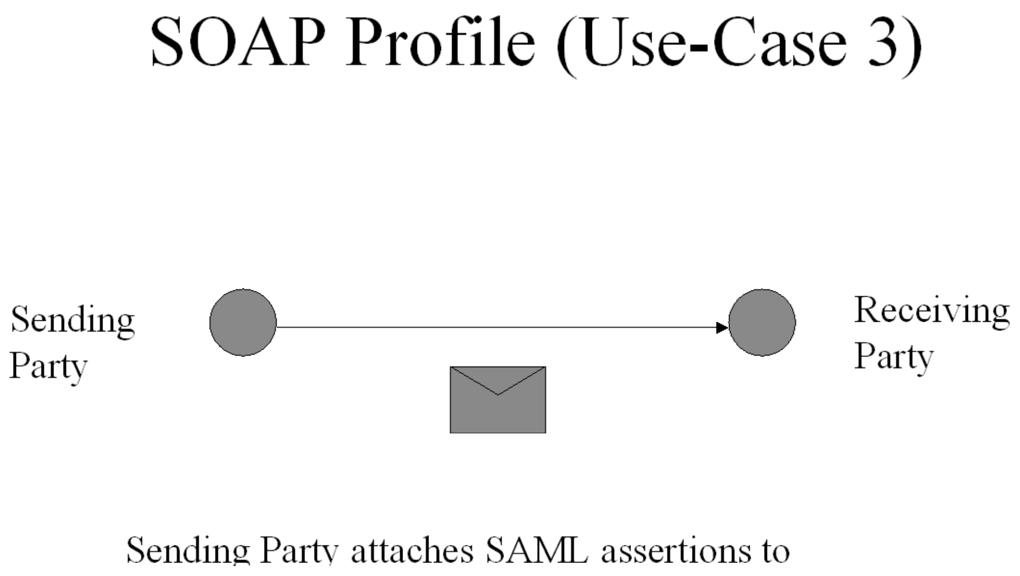
There are many possible implentations of **`<Remaining artifact>`** ([Core-Assertions-Examples, Shib-Impl]. Below, we describe an implementation called an elementary SAML artifact.

3.1.4 Elementary SAML artifact

```
<TypeCode> := 0x0001
<RemainingArtifact> := <PartnerID> <AssertionID>
<PartnerID> := byte1byte2byte3byte4
<AssertionID> := byte1byte2byte3byte4byte5byte6byte7byte8
```

<PartnerID> is a four byte value used by the destination site to determine source site identity. It is likely that such a value would have been agreed upon using some out-of-band technique between the source and destination site. <AssertionID> MUST be drawn from a random number sequence [RFC1750] generated by the source site and serves to identify the assertion at the source site. There is no authentication component to this profile.

3.2 SOAP


SOAP Profile (Use-Case 3)

Sending Party attaches SAML assertions to

{PRIVATE "TYPE=PICT;ALT=Figure 2: SOAP Message Transfer"}

3.2.1 Overview

The SOAP profile for SAML is based on a single interaction between a sending party and a receiving party. The sending party adds with one or more SAML assertions to a SOAP document and sends the message to the receiving party. The receiving party processes the SAML assertion and either returns an error or goes on to process the message in the standard way. The message may be sent over any protocol for which a SOAP protocol binding is available [SOAP].

SOAP provides a flexible header mechanism, which may be (optionally) used for extending SOAP payloads with additional information. A header entry is identified by its fully qualified element name, which consists of the namespace URI and the local name. All immediate child elements of the SOAP Header element MUST be namespace-qualified.

3.2.2 SOAP Headers and Error Processing

SAML assertions MUST be contained within the SOAP <Header> element contained within the SOAP <Envelope> element. Two standard SOAP attributes are available for use with header elements: actor and mustUnderstand. Use of the actor attribute is application dependent and no normative use is specified herein.

The SOAP mustUnderstand global attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process. SAML assertions MUST have the mustUnderstand attribute set to 1; this ensures that a SOAP processor to which the message is directed must be able to successfully process the SAML assertions or return a SOAP message with <Fault> element as the message body. The returned <Fault> element takes the form:

```
<Fault>
   <Faultcode>mustUnderstand</Faultcode>
   <Faultstring>…</Faultstring>
</Fault>
```

If the receiving party is able to successfully process the attached SAML assertions, and based on their contents does not further process the body of the SOAP message, it MUST return a SOAP message with <Fault> element as the message body. The returned <Fault> element takes the form:

```
<Fault>
   <Faultcode>Client.SAML</Faultcode>
   <Faultstring>Subject not authorized</Faultstring>
</Fault>
```

3.2.3 Confidentiality

In the absence of a mature [XML-Encryption] specification, confidentiality has to be ensured by selection of a SOAP protocol binding which preserves confidentiality. This would include, for example, HTTPS, S/MIME or some proprietary encryption scheme understood by both sender and recipient.

3.2.4 Example

The following example illustrates the addition of SAML assertions to a SOAP message:
{PRIVATE "TYPE=PICT;ALT=Figure 3: SOAP document with inserted assertions"**}**

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=http://schema.xmlsoap.org/soap/envelope/>

<SOAP-ENV:Header xmlns:SAML="…">
        <SAML:Assertion  mustUnderstand=1>…</SAML:Assertion>
        <SAML:Assertion mustUnderstand=1>…</SAML:Assertion>
</SOAP-ENV:Header>
…
<SOAP-ENV:Header>

<SOAP-ENV:Body>
   <message_payload/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.2.5 Integrity of Assertion Attachment

OPEN ISSUE: We have not addressed the issue of the integrity of assertion attachment for the composite SOAP message. The step of adding SOAP assertions to a SOAP message must itself be secured. Once assertions are packaged together with a business payload, some form of integrity check is required to ensure that the linkage between the two has not been modified. Any solution would require some extension to the assertion element schema as described in [draft-sstc-core-0.7].

Two solutions have been proposed on the security services archive [attachment-integrity]:

(1) a hash of the business payload should be placed in the assertion,
(2) public key of the sending party is included in the assertion.

In case (2), the entire package (assertion + payload) must further be signed using the sending parties private key. It is important to distinguish between this signing act and that of an issuer signing an assertion.

Solution (1) has the advantage that it does not require a PKI but it does require that each assertion be obtained in the context of a specific business payload. It does not support the "re-use" of an assertion over multiple payloads.

4 References

[AuthXML] AuthXML: A Specification for Authentication Information in XML.
http://www.oasis-open.org/committees/security/docs/draft-authxml-v2.pdf

[BEEP] The Blocks Extensible Exchange Protocol Core
http://www.normos.org/ietf/draft/draft-ietf-beep-framework-11.txt

[Glossary] OASIS Security Services TC: Glossary.
http://www.oasis-open.org/committees/security/docs/draft-sstc-hodges-glossary-02.html

[S2ML] S2ML: Security Services Markup Language, Version 0.8a, January 8, 2001.
http://www.oasis-open.org/committees/security/docs/draft-s2ml-v08a.pdf

[draft-sstc-core-07.doc] Security Assertions Markup Language, Version 0.7, May 14th, 2001.
http://www.oasis-open.org/committees/security/docs/draft-sstc-core-07.pdf

[Shib] Shiboleth Overview and Requirements
http://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-requirements-00.htmlhttp://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-requirements-00.html

[Shib-Impl] Ariel Glenn, David L. Wasley, A Possible Model for a Shibboleth Implementation, Version 1.4,
http://middleware.internet2.edu/shibboleth/docs/draft-glenn-shibboleth-model-00.pdf

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1

[RFC1750] Randomness Recommendations for Security.

[SOAP] Simple Object Access Protocol (SOAP) 1.1 , W3C Note 08 May 2000

[Core-Assertions-Examples] Core Assertions Architecture, Examples and Explanations,
http://www.oasis-open.org/committees/security/docs/draft-sstc-core-phill-07.pdf

[attachment integrity]
http://lists.oasis-open.org/archives/security-services/200105/msg00028.html

5 Appendix A

http://support.microsoft.com/support/kb/articles/Q208/4/27.ASP

The information in this article applies to:
Microsoft Internet Explorer (Programming) versions 4.0, 4.01, 4.01 SP1, 4.01 SP2, 5, 5.01, 5.5

SUMMARY
Internet Explorer has a maximum uniform resource locator (URL) length of 2,083 characters, with a maximum path length of 2,048 characters. This limit applies to both POST and GET request URLs.
If you are using the GET method, you are limited to a maximum of 2,048 characters (minus the number of characters in the actual path, of course).
POST, however, is not limited by the size of the URL for submitting name/value pairs, because they are transferred in the header and not the URL.
RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, does not specify any requirement for URL length.

REFERENCES
Further breakdown of the components can be found in the Wininet header file. Hypertext Transfer Protocol -- HTTP/1.1 General Syntax, section 3.2.1
Additional query words: POST GET URL length
Keywords : kbIE kbIE400 kbie401 kbGrpDSInet kbie500 kbDSupport kbie501 kbie550 kbieFAQ
Issue type : kbinfo
Technology :

Issue: 19971110-3 Product: Enterprise Server

Created: 11/10/1997 Version: 2.01
Last Updated: 08/10/1998 OS: AIX, Irix, Solaris
Does this article answer your question?
Please let us know!

Question:
How can I determine the maximum URL length that the Enterprise server will accept? Is this configurable and, if so, how?

Answer:

Any single line in the headers has a limit of 4096 chars; it is not configurable.

----

issue: 19971015-8 Product: Communicator, Netcaster

Created: 10/15/1997 Version: all

Last Updated: 08/10/1998 OS: All

Does this article answer your question?

Please let us know!

Question:

Is there a limit on the length of the URL string?

Answer:

Netscape Communicator and Navigator do not have any limit. Windows 3.1 has a restriction of 32kb (characters). (Note that this is operating system limitation.) See this article for information about Netscape Enterprise Server.
