OASIS SSTC Bindings Model

Prateek Mishra, Netegrity
Bob Blakley, Tivoli
Scott Cantor, Ohio State University
Marlena Erdos, Tivoli
Chris Ferris, SUN Microsystems
Simon Godik, Crosslogix
Jeff Hodges, Oblix
<big><small>Tim Moses, Entrust
Bob Morgan, University of Washington
Evan Prodromou, Securant
Irving Reid, Baltimore
Krishna Sankar, Cisco
</small>

draft-sstc-bindings-model-07.doc

7 December 2001

1 Revision History

Revision	Date	Editor	Title
0.5	18 August 2001	Prateek Mishra	Bindings model draft
0.6	8 November 2001	Prateek Mishra	Removed SAML HTTP binding, removed artifact PUSH case, updated SOAP profile based on Blakley note
0.7	3 December 2001		Re-structured based on F2F#5 comments; separated discussion and normative language

2 Introduction

2.1 Scope

<big>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</big><big> </big><big>request-response message exchanges. </big>

<big>The high-level goal of this document is to specify how: </big>

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

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.</big><big> </big>

<big>(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, </big><big></big><big>communicated from the originating site to a destination, and subsequently processed at the destination. A set of rules</big><big> </big><big>describing how to embed and extract SAML assertions into a framework or protocol is termed a </big><big>*profile*</big><big> for SAML. A set of rules for embedding and extracting SAML assertions into a </big><big></big><big>specific class of <FOO> objects is termed a </big><big>*<FOO> profile*</big><big> of 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, description of SAML-related error states at the destination.

</big>

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

2.2 Contents

<big>The remainder of this document is in four sections:
</big>

- <big>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.
  </big>
- <big>A process framework for describing and registering proposed and future protocol bindings and profiles.
  </big>
- <big>Protocol bindings for selected protocols. Bindings MUST be specified in enough detail to satisfy the inter-operability requirement.
  </big>
- <big>Profiles for selected protocols and frameworks. Profiles MUST be specified in enough detail to satisfy the inter-operability requirement.
  </big>

2.3 Guidelines for Specifying Protocol Bindings and Profiles<big> </big>

<big>Issues that MUST be identified in each protocol binding and profile:</big><big>
</big><big></big><big></big><big>
</big><big>(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.</big><big>
</big><big>
</big><big>(2) Identification of parties involved in each interaction: how many parties are involved in the interaction? Can intermediaries be involved?
</big>

<big>(3) Authentication of parties involved in each interaction: Is authentication required? What types of authentication are acceptable?</big><big>
</big><big>
</big><big>(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? </big><big></big><big>
</big><big>
</big><big>(6) Error states: characterization of error states at each participant, especially those that receive and process SAML assertions or messages.</big>

(7) Security considerations: including analysis of threats and description of counter-measures.

2.4 Process Framework for Describing and Registering Protocol Bindings and Profiles

<big>When a profile or protocol binding is registered, the following information MUST be supplied:</big>

<big> </big>

1. <big>Identification: specify a URI that authoritatively identifies this profile or protocol binding.
   </big>
2. <big>Contact information: specify the postal and electronic contact information for the author of the profile or protocol binding.
   </big>
3. <big>Description: the description SHOULD follow the guidelines for profiles and protocol bindings given above.
   </big>
4. <big>Updates: references to previously registered profiles or bindings that the current entry improves or obsoletes.

The Security Services Technical Committee (SSTC) at OASIS (http://www.oasis-open.org) will maintain a respository of submitted bindings and profiles titled “Additional Bindings and Profiles”. The SSTC will also provide instructions for submission of bindings and profiles by Oasis members.</big><big>
</big>

<big>Whe</big>

3 Protocol Bindings

3.1 SAML Binding for SOAP

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 over HTTP. Since its introduction, it has attracted much attention, and it is expected to provide the foundation for many future Web-based services.

SOAP 1.1 [SOAP1.1] 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 extended 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, the former describes using SAML over SOAP, and the latter describes using SAML for SOAP.

Like SAML, SOAP can be used over multiple underlying transports. This document describes protocol independent aspects of the SAML SOAP binding and calls out the use of HTTP protocol as mandatory-to-implement. It includes recomendations for HTTP specifics, including HTTP headers, error reporting, authentication, message integrity, and confidentiality.

[Issue: Bob B wanted to include: “This description is general for SOAP and may use any protocol”. I think paragraph above says the same thing].

SOAP over HTTP does not cover security considerations. Refer to SAML security considerations document [SEC-CONS] for details.

3.1.1 Overview.

3.1.1.1 Referenced Namespaces

SOAP envelope namespace:

SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope

SAML core assertions namespace:

saml=http://www.oasis-open.org/committees/security/docs/sstc-schema-assertion.xsd

SAML protocol namespace:

samlp=http://www.oasis-open.org/committees/secutiry/docs/sstc-schema-protocol.xsd

3.1.1.2 Basic Operation

SOAP messages consist of three elements: an envelope, header data, and a message body. SAML messages (<samlp:Request> and <samlp:Response>) MUST be enclosed within the SOAP message body.

SOAP 1.1 also defines an optional data encoding system. This system is not used within the SAML SOAP binding. This means that SAML messages can be transported using SOAP without re-encoding from the "standard" SAML schema to one based on SOAP encoding.

The system model used for SAML conversations over SOAP is a simple request-response model. A sender transmits a SAML <samlp:Request> within the body of a SOAP message to a receiver. The receiver processes the SAML request and returns a <samlp:Response> within the body of another SOAP message.

3.1.2 SOAP Headers

A SAML sender in a SAML conversation over SOAP MAY add arbitrary headers to the SOAP message. SAML 1.0 does not define any additional SOAP headers.

[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.]

A SAML receiver MUST NOT require any headers for the SOAP message.

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

3.1.3 SAML Requests

A SAML request <samlp:Request> is stored as the (only) child of the <SOAP-ENV:body> element of a SOAP message. The sender MUST NOT include more than one SAML request per SOAP message or include any additional XML elements in the SOAP body.

On receiving a SAML request as a SOAP message, the SAML receiver MUST return either a SAML response <samlp:Response> or a SOAP fault code.

3.1.4 SAML Responses

A SAML response <samlp:Response> MUST appear as the (only) child of the <SOAP-ENV:body> element in a SOAP message. The SOAP message MUST contain exactly one SAML response element. The SAML receiver MUST NOT include any additional XML elements in the SOAP body.

On receiving a SAML response in a SOAP message, the SAML sender MUST NOT send a fault code or other error messages to the receiver.

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

3.1.5 Fault Codes

If a receiver cannot, for some reason, process a SAML request, it should return a SOAP fault code. SOAP Fault codes MUST NOT be sent for errors within the SAML problem domain, e.g. inability to find extension schema or as a signal that the subject is not authorized to access resource in an authorization query.

[Issue: If valid SAML requests can not be extracted, SOAP fault code must be returned]

Section 4.1 of [SOAP1.1] describes SOAP faults and fault codes.

3.1.6 Authentication

Authentication of both sender and receiver is optional and depends upon the environment of use. Authentication protocols available from the underlying substrate protocol MAY be utilized to provide authentication. Section 3.1.9.2 describes authentication in the HTTP environment.

3.1.7 Message Integrity

Message integrity of both request and response is optional and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message integrity.

3.1.8 Confidentiality

Confidentiality of both request and response is optional and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message confidentiality.

3.2 SAML use of the SOAP binding over HTTP.

Any SAML processor implementing the SAML SOAP binding MUST implement SAML over SOAP over HTTP.

The HTTP binding for SOAP is described in Section 6.0 of [SOAP1.1]. It requires the use of a SOAPAction header as part of a SOAP HTTP request. A SAML receiver MUST NOT depend on the value of this header. A SAML sender MAY set the value of SOAPAction header to "http://www.oasis-open.org/committees/security".

3.2.1.1 HTTP Headers.

HTTP proxies MUST NOT cache responses carrying SAML assertions.

When using HTTP 1.1:

(1) a SAML receiver MUST NOT include Cache-Control header field in the response UNLESS its value is set to no-store.

(2) Expires response header field SHOULD NOT be included, UNLESS it is disabled by Cache-Control header with the value of no-store.

There are no other restrictions on HTTP headers.

3.2.1.2 Authentication

SAML sender and SAML receiver MUST implement following authentication methods:

1. No client authentication.

2. HTTP basic client authentication [rfc2617] with and without SSLv3 or TLS 1.0.

3. HTTP over SSLv3 or TLS 1.0[Appendix C] server authentication with a server-side certificate.

4. HTTP over SSLv3 or TLS 1.0 [Appendix C] client authentication with a client-side certificate.

Should a SAML receiver utilize SSLv3 or TLS 1.0 [Appendix C] it MUST use a server-side certificate.

3.2.1.3 **Message Integrity**

SAML receivers MUST implement message integrity by utilizing HTTP over SSLv3 or TLS1.0 [AppendixC] with a server-side certificate.

3.2.1.4 Message Confidentiality

When message confidentiality is required, HTTP over SSLv3 or TLS 1.0 [Appendix C] with a server-side certificate MUST be used.

3.2.1.5 Security Considerations

Each combination of authentication-message integrity-confidentiality should be analyzed for vulnerability in the context of deployment environment. See the security considerations document [saml-sec-cons] for detailed discussion.

[Rfc2617] provides descriptions of possible attacks in HTTP environment using basic and authentication schemes.

3.2.1.6 Error reporting

A SAML receiver that refuses to perform a SAML message exchange with the sender it should return a "403 Forbidden" response. In this case content of the HTTP body is undefined.

As described in [SOAP1.1 section 6.2], in case of a SOAP error while processing SOAP request the SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP message in response containing a SOAP Fault element. This type of error should be returned for SOAP related errors detected before control is passed to the SAML processor, or when the SOAP processor reports an internal error. Examples include situations when soap namespace is incorrect, SAML schema can not be located, SOAP message signature does not validate, etc.

In case of a SAML processing error the SOAP HTTP server MUST respond with "200 OK" and include SAML specified error description as the only child of the SOAP-ENV:Body element. For complete list of SAML error codes see [SAML-CoreDoc].

3.2.1.7 Example: SAML over SOAP/HTTP

REQUEST:

```
POST /SamlService HTTP/1.1
Host: www.example.com
Content-Type: text/xml
Content-Length: nnn
SOAPAction: http://www.oasis-open.org/committees/security
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
            <samlp:Request xmlns:samlp="..." xmlns:saml="..."
            xmlns:ds="...">
                  <ds:Signature> ... </ds:Signature>
                  <samlp:AuthenticationQuery>
```

```
                        ...
                  </samlp:AuthenticationQuery>
            </samlp:Request>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<SOAP-ENV:Envelope
         xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body>
            <samlp:Response xmlns:samlp="..." xmlns:saml="..."
                  xmlns:ds="..." samlp:StatusCode="Success">
                  <ds:Signature> ... </ds:Signature>
                  <saml:AssertionSimple>
                     <saml:AuthenticationStatement>
                              ...
                      </saml:AuthenticationStatement>
                  </saml:AssertionSimple>
            </samlp:Response>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4 Profiles</big>

4.1 Web Browser Single Sign-On

4.1.1 Overview

The web browser profile utilizes terminology taken from Use Case 1 and Scenario 1-1 of the SAML Requirements document. In this use-case, a web user authenticates with a *source site*. The web user then uses a secured resource at a destination site, without directly authenticating to the *destination site*.

We assume that <big>the user is utilizing a standard commercial browser and has authenticated to a source site. Further, the source site has some form of security engine in place that can track locally authenticated users [WEB-SSO]. Typically, this takes the form of a session which may be represented by an encrypted cookie or an encoded URL or by the use of some other technology [SESSION]. This is a substantial requirement but one which is met by a large class of security engines.

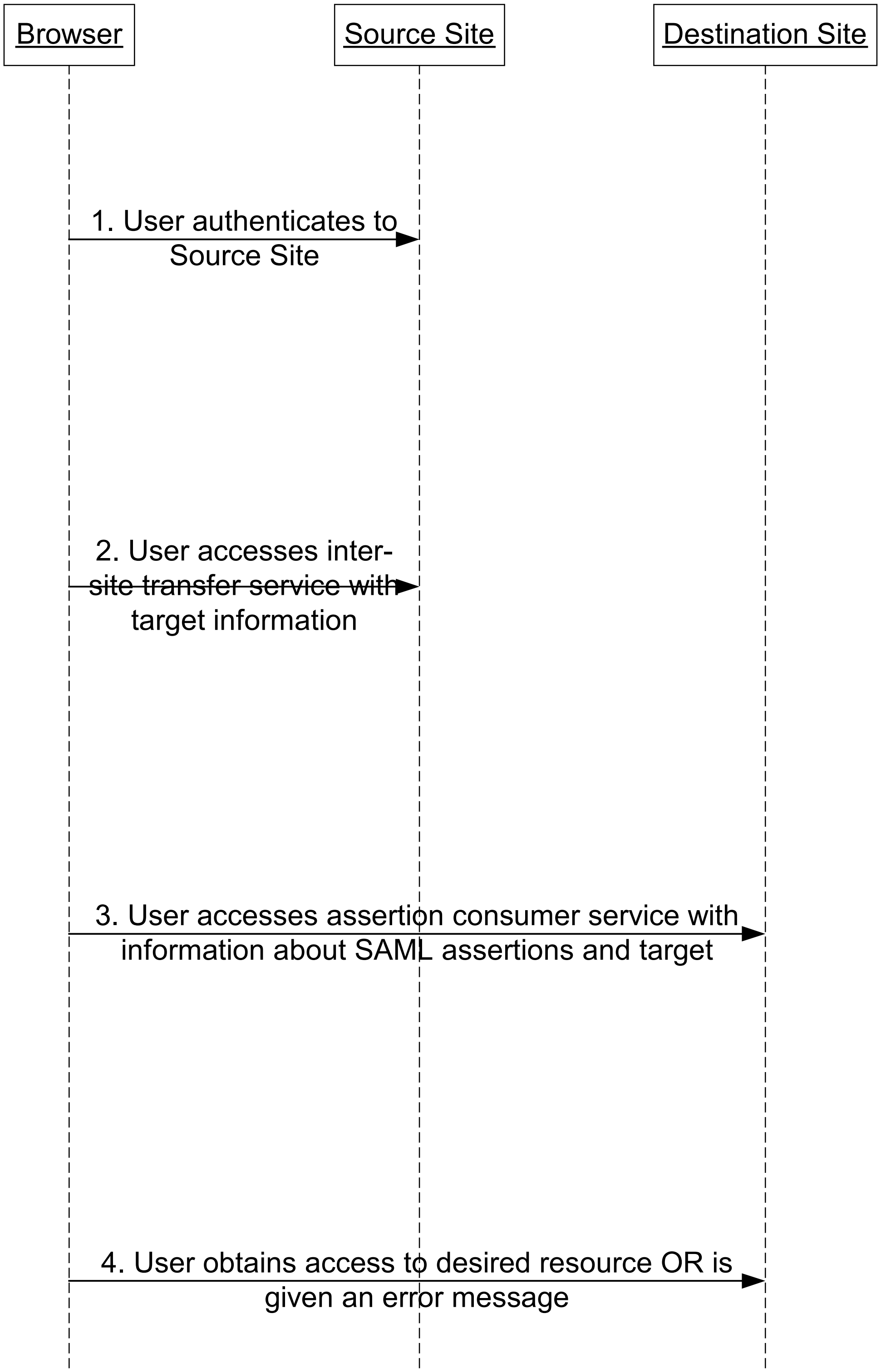

Browser
Source Site
Destination Site
1. User authenticates to Source Site
2. User accesses inter-site transfer service with target information
3. User accesses assertion consumer service with information about SAML assertions and target
4. User obtains access to desired resource OR is given an error message

Figure 1: Web Browser Single Sign-On

At some point, the user attempts to access a *target* resource available from the destination site and subsequently through one or more steps (e.g., re-direction) arrives at an *inter-site transfer service*[1] at the source site. Starting from this point, the SAML web browser profiles describe a canonical sequence of HTTP protocol exchanges that transit the user browser to a distinguished *assertion consumer service* at the destination site. Information about *SAML assertions* associated with the user and the desired target are conveyed from the source to the destination site by the protocol exchange.

The destination site can examine both the assertions and target information and determine whether to allow access to the target resource, thereby achieving web single sign-on for authenticated users originating from a source site. Often, the destination site also utilizes a standard security engine that will create and maintain a session, possibly utilizing information contained in the source site assertions, for the user at the destination site.

4.1.1.1 Relevant Technology

We describe two HTTP-based techniques available for conveying information from one site to another via a stock commercial browser. We do not discuss the use of cookies, as these impose the limitation that both the source and destination site belong to the same "cookie domain".

- *Form POST*: SAML assertions are uploaded to the user browser within a HTML Form [HTML] and conveyed to the destination site as part of a HTTP POST payload when the user “submits” the form,

- *SAML Artifact*: A “small”, bounded-size SAML artifact, which unambiguously identifies an assertion to the source site, is carried as part of a URL query string and conveyed via re-direction to the destination site; the destination site must acquire the referenced assertion by some further steps. Typically, this involves the use of a registered SAML protocol binding.

The need for a “small’’ SAML artifact is motivated by restrictions on URL size imposed by commercial web browsers. While [RFC2616] does not specify any restrictions on URL length, in practice commercial web browsers and </big><big></big><big>application servers impose size constraints on URLs (maximum size of approximately 2000 characters [Appendix A]). Further, as developers will need to estimate and set aside URL ``real-estate’’ for the artifact, it is important that the artifact have a bounded size, i.e. with predefined maximum size. These measures ensure that the artifact can be reliably carried as part of the URL query string and thereby transferred from source to destination site.

[1] One or more URLs may be associated with such a service.

4.1.2 Profile Overview

Two distinct web browser profiles are described: one based on use of artifacts and one based on form POST. For each type of profile, a section describing the threat model and relevant counter-measures is also included.

4.1.3 SAML Artifact Profile

4.1.3.1 SAML artifact format

Depending on upon the level of security desired and associated profile protocol steps, many viable architectures may be developed for the SAML artifact ([Core-Assertions-Examples, Shib-Marlena]. We accommodate variability in the architecture by a mandatory two byte artifact type code in the representation:

```
<SAML_artifact> :=
         B64 representation of <TypeCode> <RemainingArtifact>
         <TypeCode> := Byte1Byte2
```

The following fixed size artifact is mandatory to implement for any implementation of the SAML artifact profile.

```
<TypeCode> := 0x0001
<RemainingArtifact> := <SourceID> <AssertionHandle>
<SourceID> := 20 byte sequence
<AssertionHandle> := 20 byte sequence
```

`<SourceID>` is a twenty byte sequence used by the destination site to determine source site identity. We assume that the destination site will maintain a table of sourceID values as well as the URL (or address) for the corresponding SAML query service. This information is communicated between the source and destination sites using an out-of-band technique. On receiving the SAML artifact, the destination site determines if the `<SourceID>` belongs to a known source site, retrieves the “assertion lookup” service information and invokes the service with the `<SAML_artifact>` and other values as an argument.

Any two source sites with a common destination site MUST use distinct `<SourceID>` values. Construction of <AssertionHandle> values is governed by the principle that they should have no predictable relationship to the contents of the referenced assertion at the source site and should also be difficult to “guess”.

The following practices are RECOMMENDED for the creation of SAML artifacts at source sites:

(1) Each source site selects a single Identification URL which it communicates to all potential destination sites. The domain name used within the identification URL MUST be administered by source site.

(2) The source site constructs the <SourceID> component of the artifact by taking the SHA-1 [SHA-1] hash of the identification URL.

(3) The value should be constructed from a pseudo-random number sequence [RFC1750] generated by the source site. The sequence must consist of values of size at least eight bytes.

4.1.3.2 Artifact Message Flows

</big>

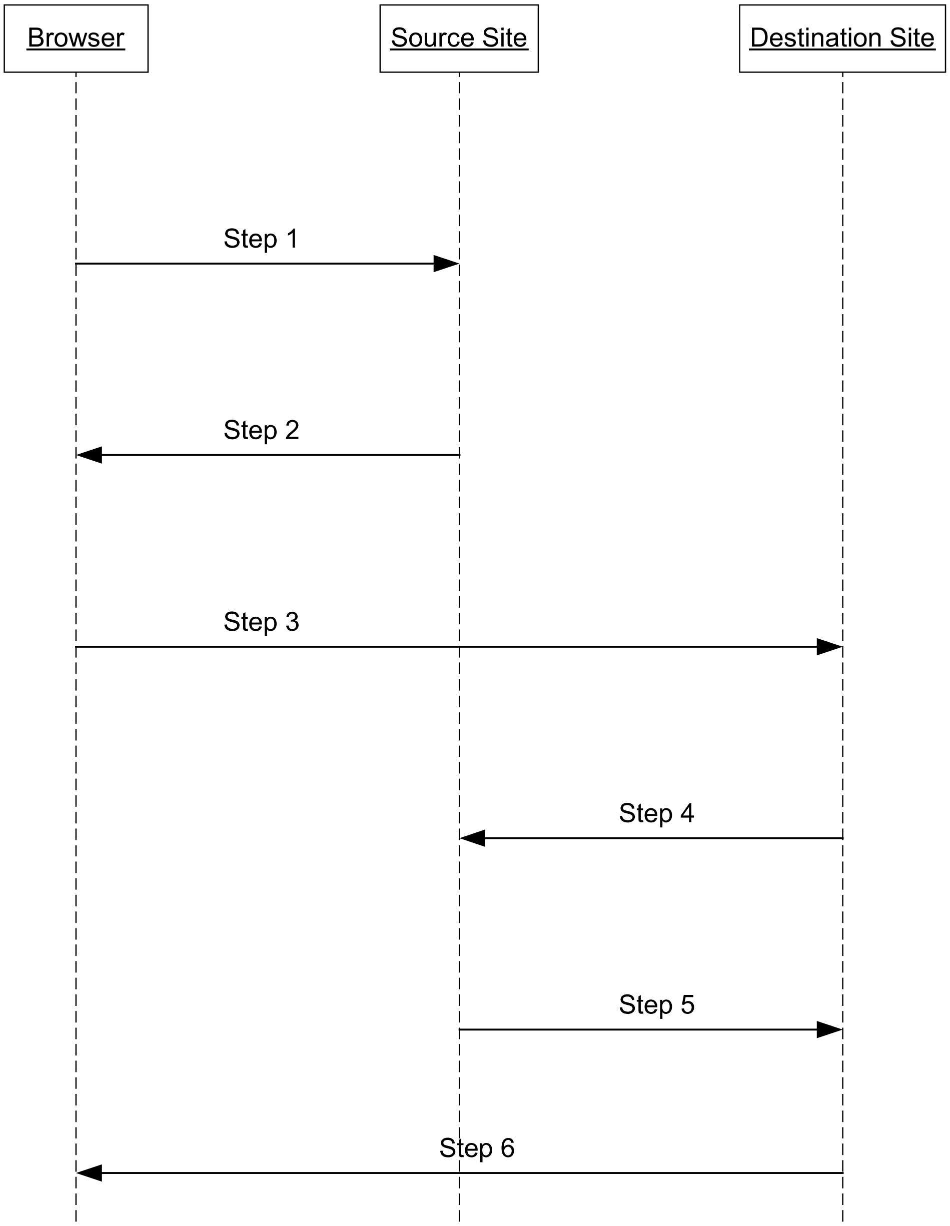
<big>This profile consists of a single interaction between three parties (source site, user equipped with a browser, destination site), with a nested sub-interaction between two parties (source site, destination site). The interaction sequence is diagrammed in Figure 1.

Terminology from [RFC1738] is used to describe components of a URL. An HTTP URL has the form:

```
http://<HOST>:<port>/<path>?<searchpart>
```

In what follows, we will specify certain portions of the searchpart component of the URL. Ellipses will be used to indicate additional but unspecified portions of the searchpart.

HTTP requests and responses may be drawn from HTTP 1.1 [RFC2068] or HTTP 1.0 [RFC1945]. Distinctions between the two are drawn only when necessary.


Browser
Source Site
Destination Site
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6

4.1.3.2.1 Step 1: HTTP Request

No normative form is given for Step 1. It is RECOMMENDED that the HTTP request take the form:

```
GET http://<inter-site transfer host name and path>?…TARGET=<Target>…<HTTP-Version>
<other HTTP 1.0 or 1.1 Components>
```

Notes:

1. `<inter-site transfer host name and path>` refers to the host name, port number and path components of an inter-site transfer URL of the source site.

2. The `Target=<Target>` name-value pair occurs in the searchpart and is used to convey information about the desired target resource at the destination site.

4.1.3.2.2 Step 2: HTTP Response

The HTTP Response MUST take the form:

```
<HTTP-Version> 302 <Reason Phrase>
<other headers>
Location : http://<assertion consumer host name and path>?<SAML searchpart>
<other HTTP 1.0 or 1.1 Components>
```

Notes:

1. `<assertion consumer host name and path>` refers to the host name, port number and path components of an assertion consumer URL at the destination site.

2. `<SAML searchpart>= …TARGET=<Target>…SAMLart=<SAML artifact> …`

A single target description MUST be included in the SAML searchpart component. At least one SAML artifact MUST be included in the SAML searchpart component; multiple SAML artifacts MAY be included. If more than one artifact is carried within `<SAML searchpart>,` all the artifacts MUST have the same SourceID.

3. HTTP 1.1 and HTTP 1.0 recommend the use of status code 302 to indicate “the requested resource resides temporarily under a different URI”. The response may also include

additional headers and an (optional) message body as described in FRC2068 and RFCXXXX.

4. Confidentiality and message integrity MUST be maintained in steps 1 and 2.

5. It is RECOMMENDED that the inter-site transfer URL be exposed over SSLv3 or TLS 1.0 [Appendix C]. Otherwise, the artifact(s) returned in step 2 will be available in plain text to any attacker.

4.1.3.2.3 Step 3: HTTP Request:

The HTTP request MUST take the form:

```
GET http://<assertion consumer host name and path>?<SAML searchpart> <HTTP-Version>
<Other HTTP 1.0 or 1.1 request components>
```

Notes:

1. `<assertion consumer host name and path>` refers to the host name, port number and path components of an assertion consumer URL at the destination site.

2. `<SAML searchpart>= …TARGET=<Target>…SAMLart=<SAML artifact> …`

A single target description MUST be included in the SAML searchpart component. At least one SAML artifact MUST be included in the SAML searchpart component; multiple SAML artifacts MAY be included. If more than one artifact is carried within `<SAML searchpart>,` all the artifacts MUST have the same SourceID.

3. Confidentiality and message integrity MUST be maintained for the HTTP request in Step 5.

4. It is RECOMMENDED that the assertion consumer URL be exposed over SSLv3 or TLS 1.0 [Appendix C]. Otherwise, the artifact(s) transmitted in Step 3 will be available in plain text to any attacker.

4.1.3.2.4 Step 6: HTTP Response

No normative form is given for the HTTP response in Step 6. Implementations SHOULD provide some form of helpful error-message in the case where access to resources at the destination site is disallowed.

4.1.3.2.5 Steps 4 and 5

1. These steps MUST utilize a SAML protocol binding for a SAML message exchange between source and destination site.

2. The destination site MUST send a ***<samlp:Request>*** message to the source site, querying against all of the SAML artifacts delivered to the destination site in step 3.

3. If the source site can find or construct the requested assertions it responds with a <samlp:Response> message with the requested assertions. Otherwise, it returns an appropriate error, as defined within the selected SAML binding, to the destination site.

4. In the case where the source site returns assertions within <samlp:Response>, it MUST return exactly one assertion for each SAML artifact found in the corresponding <samlp:Request> element. The case where fewer or greater number of assertions is returned within the <samlp:Respond> element MUST be treated as an error state by the destination site.

5. The source site MUST implement a "one-time request" property for any SAML artifact. Many simple implementations meet this constraint, such as deleting the relevant assertion from persistent storage at the source site after one lookup. Should a SAML artifact is presented to the source site again, the source site MUST return the same message as when it is queried with an unknown artifact.

6. The selected SAML protocol binding MUST provide confidentiality, message integrity and bilateral authentication. The source site MUST implement the SAML SOAP binding with support for confidentiality (SSLv3 or TLS 1.0 [Appendix C]); support for other protocol bindings is not mandatory.

7. [pm1]The source site MUST return an error response if it receives a <samlp:Request> message from a destination site X containing an artifact issued by the source site to some other destination site Y. One way to implement this feature is to have source sites maintain a list of artifact and destination site pairs.

8. We will refer to an assertion with one or more authentication statements and a <Conditions> element, with NotBefore and NotOnOrAfter attributes present, as a *SSO (single-sign on) assertion*. At least one of the SAML assertions returned to the destination site MUST be a *SSO assertion*.

9. Authentication statements MAY be contained within one or more returned assertions.

10. The <saml:ConfirmationMethod> element of each assertion MUST be set to SAML Artifact (5.1.1 of [Core-20]).

4.1.3.3 Threat Model and Counter-Measures

This section utilizes materials from [Shib-Marlena] and [Rescorla-Security].

4.1.3.3.1 Stolen artifact

Threat:

If an eavesdropper (Eve) can copy the real user's SAML artifact, then the Eve could construct a URL with the real user's SAML artifact and be able to impersonate the user at the destination site.

Counter-Measure:

As indicated in Steps 1, 2, 5 and 6, confidentiality must be provided whenever an artifact is communicated between a site and the user's browser. This provides protection against an Eve gaining access to a real user's SAML artifact.

Should Eve defeat the measures used to ensure confidentiality, additional counter-measures are available. Recall that SAML assertions communicated through Step 5 must always include an SSO assertion. SSO assertions SHOULD have short validity periods (values for `NotBefore` and `NotOnOrAfter` attributes) consistent with successful functioning of the profile. This ensures that a stolen artifact can only be used successfully within a small time window.

Source and destination sites SHOULD make some reasonable effort to ensure that clock settings are both sites differ by at most a few minutes. Many forms of time synchronization service are available, both over the Internet and from proprietary sources.

RECOMMENDATIONS for the Source Site:

(a) Source sites SHOULD track the time difference between when a SAML artifact is generated and placed on a URL line and when the destination site "calls back" for an assertion. A maximum time limit of a few minutes is recommended. Should an assertion be requested by a destination site query beyond this time limit, a SAML error should be returned by the source site.

(b) SSO assertions MAY BE created by the source site either when the corresponding SAML artifact is created or when the destination site "calls back" for an assertion. In each of these cases, the validity period of the assertion should be set appropriately (longer in the former case, shorter for the latter).

(c) values for `NotBefore` and `NotOnOrAfter` attributes of SSO assertions SHOULD have the shortest possible validity period consistent with successfully communication of the assertion from source to destination site. This is typically on the order of a few minutes.

RECOMMENDATIONS for Destination Site:

(a) The destination site MUST check the validity period of all assertions obtained from the source site and reject expired assertions. A destination site MAY choose to implement a stricter test of validity for SSO assertions, such as for example, requiring the `IssueInstant` attribute value or `AuthenticationInstant` attribute value of the assertion to be within a few minutes of the time at which the assertion is received at the destination site.

(b) Authentication statements MAY include an `<AuthenticationLocality>` element with the IP address of the user. The destination site MAY check the browser IP address against the IP address contained in the authentication statement.

4.1.3.3.2 Attacks on Steps 4 and 5

Threat: The message exchange on steps 4 and 5 may be attacked in a variety of ways, including: artifact or assertion theft, replay, message insertion or modification, MITM (man-in-the-middle attack).

Counter-Measure: The requirement for the use of a SAML protocol binding with the properties of bilateral authentication, message integrity and confidentiality obviates these attacks.

4.1.3.3.3 Malicious Destination Site

Threat: Since the destination site obtains artifacts from the user, a malicious site could impersonate the user at some new destination site. The new destination site would obtain assertions from the source site and believe the malicious site to be the user.

Counter-Measure:

The new destination site will need to authenticate itself to the source site so as to obtain the SAML assertions corresponding to the SAML artifacts. There are two cases:

(a) If the new destination site has no relationship with the source site, it will be unable to authenticate and this step will fail.

(b) If the new destination site has an existing relationship with the source site, the source site will determine that artifacts are being queried against from a site other than the one to which the artifacts were issued. In such a case, the source site will not provide the assertions to the new destination site.

4.1.3.3.4 Forged SAML artifact

Threat: A MAL (malicious user) could forge a SAML artifact.

Counter-Measure:

A SAML artifact must be constructed in such a way that it is very hard to guess and Section 4.1.3 provides specific recommendations in this space. A MAL could attempt to repeatedly "guess" a valid SAML artifact value (one that corresponds to an existing assertion at a source site) but given the size of the value space would likely require a very large number of failed attempts. A source site SHOULD implement measures to ensure that repeated attempts at querying against non-existent artifacts are monitored.

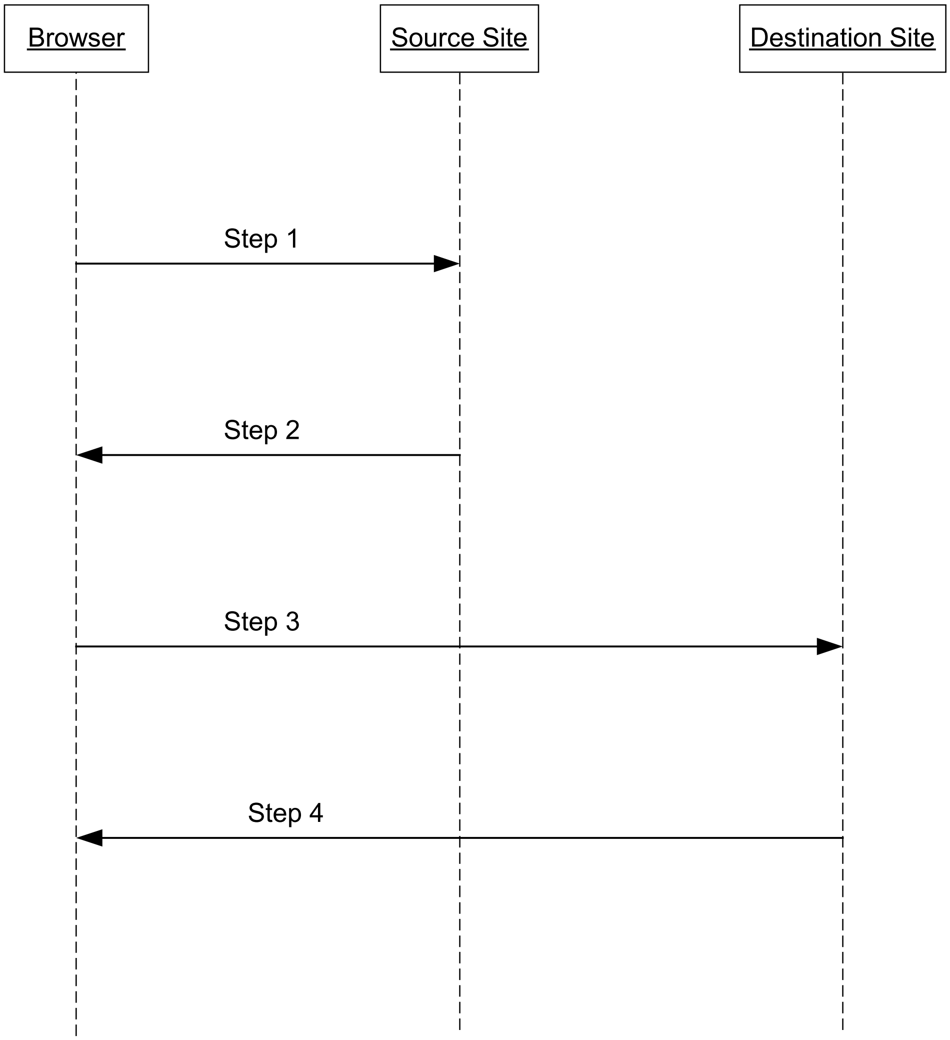
4.1.3.3.5 Browser State Exposure

Threat: The SAML artifact profile involves "upload" of SAML artifacts to the web browser from a source site. This information is available as part of the web browser state and is usually stored in persistent storage on the user system in a completely unsecured fashion. The threat here is that the artifact may be "re-used" at some later point in time.

Counter-Measure: The "one-use" property of SAML artifacts ensures that they may not be re-used from a browser. Due to the recommended short life-times of artifacts and mandatory SSO assertions, it is difficult to steal an artifact and re-use it from some other browser at a later time.

4.1.4 Form POST

Figure 2 provides a description of a web browser profile based upon the use of "POST" to convey SAML assertions from source to destination site [S2ML, Anders-Browser-Profile].



4.1.4.1.1 Step 1: HTTP Request

No normative form is given for Step 1 (HTTP request). It is RECOMMENDED that the request take the form:

```
GET http://<inter-site transfer host name and path>?…TARGET=<Target>…<HTTP-Version>
<other HTTP 1.0 or 1.1 Components>
```

Notes:

`<inter-site transfer host name and path>` refers to the host name, port number and path components of an inter-site transfer URL at the source site.

4.1.4.1.2 Step 2: HTTP Response

The HTTP Response in MUST take the form:

```
<HTTP-Version> 200 <Reason Phrase>
<additional HTTP 1.0 or 1.1 Components>
```

Notes:

1. `<additional HTTP 1.0 or 1.1 Components>` MUST include an HTML Form [Chapter 17, HTML 4.01] with the following Form body:

```
<Body>
<FORM Method=”Post” Action=”<assertion consumer host name and path>”>
<INPUT TYPE=”Submit” NAME=”button” Value=”Submit”>
<INPUT TYPE=”hidden” NAME=”SAMLAssertion” Value=”B64(<assertion>)”>
…
<INPUT TYPE=”hidden” NAME=”TARGET” Value=”<Target>”>
</Body>
```

`2. <assertion consumer host name and path>` refers to the host name, port number and path components of an assertion consumer URL at the destination site.

3. At least one SAML assertion MUST be returned included within the FORM body with the control name `SAMLAssertion`; multiple SAML assertion MAY be included. A single target description MUST be included with the control name `TARGET`.

3. Every SAML assertion MUST be digitally signed following the guidelines given in [SAML-DSIG-Profile].

4. Confidentiality and message integrity MUST be maintained for steps 1 and 2. It is RECOMMENDED that the inter-site transfer URL exposed over SSLv3 or TLS 1.0 [Appendix C]. Otherwise, the assertion(s) returned on (step (2)) will be available in plain text to any attacker.

Step 3: HTTP Request

In step 3, the browser submits a form and creates the following HTTP request. Appendix B describes a technique for form submission which avoids user input.

The HTTP request MUST include the following components:

```
POST http://<assertion consumer host name and path>
<Other HTTP 1.0 or 1.1 request components>
```

Notes:

1.
`<Other HTTP 1.0 or 1.1 request components>`

Consists of the form data set derived by the browser processing of the form data received in Step 2 according to 17.13.3 of [HTML4.01]. At least one SAML assertion MUST be included within the form data set with control name `SAMLAssertion`; multiple SAML assertions MAY be included. A single target description MUST be included with the control name set to `TARGET`.

2. At least one of the SAML assertions posted to the destination site MUST be a single-sign on assertion with the additional restriction that the `<Target>` element MUST also be included within the SSO assertion and its value set to `<assertion consumer host name and path>`.

3. The destination site MUST ensure a “single use” policy for SSO assertions communicated via form data. The implication here is that the destination site will need to be stateful. A simple implementation maintains a table of pairs:

```
Assertion Id,  Time at which entry is to be deleted
```

The time at which an entry is to be deleted is based upon the SSO assertion life-time. Since SSO assertions containing authentication statements are recommended to have short life-times in the web browser context, such a table would be of manageable size.

4. Confidentiality and message integrity MUST be maintained for the HTTP request in Step 3. It is RECOMMENDED that the assertion consumer URL be exposed over SSLv3 or TLS 1.0 [Appendix C]. Otherwise, the assertion(s) transmitted in Step 3 will be available in plain text to any attacker.

5. The <saml:ConfirmationMethod> element of each assertion MUST be set to Assertion Bearer (5.1.2 of [Core-20]).

4.1.4.1.3 Step 4: HTTP Response

No normative form is given for the HTTP response in Step 6. Implementations SHOULD provide some form of helpful error-message in the case where access to resources at the destination site is disallowed.

4.1.4.2 Threat Model and Counter-Measures

This section utilizes materials from [Shib-Marlena] and and [Rescorla-Security].

4.1.4.2.1 Stolen assertion

Threat: If an eavesdropper (Eve) can copy the real user's SAML assertion (Form POST), then the Eve could construct an appropriate POST body and be able to impersonate the user at the destination site.

Counter-Measure: As indicated in Steps 1, 2, 3 and 4, confidentiality must be provided whenever an assertion is communicated between a site and the user's browser. This provides protection against an Eve gaining access to a user's SAML assertion.

Should Eve defeat the measures used to ensure confidentiality, additional counter-measures are available. Recall, that SAML assertions communicated through Step 3 must always include an SSO assertion. SSO assertions SHOULD have short validity periods (values for `NotBefore` and `NotOnOrAfter` attributes) consistent with successful functioning of the profile. This ensures that a stolen assertion can only be used successfully within a small time window.

Source and destination sites SHOULD make some reasonable effort to ensure that clock settings are both sites differ by at most a few minutes. Many forms of time synchronization service are available, both over the Internet and from proprietary sources.

RECOMMENDATIONS for the Source Site:

(a) values for `NotBefore` and `NotOnOrAfter` attributes of SSO assertions SHOULD have the shortest possible validity period consistent with successfully communicating the assertion from source to destination site. This is typically of the order of a few minutes.

RECOMMENDATIONS for Destination Site:

(a) The destination site MUST check the validity period of all assertions obtained from the source site and reject expired assertions. A destination site MAY choose to implement a stricter test of validity for SSO assertions, such as for example, requiring the `IssueInstant` attribute value or `AuthenticationInstant` attribute value of the assertion to be within a few minutes of the time at which the assertion is received at the destination site.

(b) Authentication statements MAY include an `<AuthenticationLocality>` element with the IP address of the user. The destination site MAY check the browser IP address against the IP address contained in the authentication statement.

4.1.4.2.2 MITM Attack

Threat: Since the destination site obtains bearer SAML assertions from the user via a Form post, a malicious site could impersonate the user at some new destination site. The new destination site would believe the malicious site to be the user.

Counter-Measure:

The destination site MUST check the `<saml:Target>` elements of the SSO assertion to ensure that at least one of their values matches the `<assertion consumer host name and path>.` As the assertion is digitally signed, the `<saml:Target>` value cannot be altered by the malicious site.

4.1.4.2.3 Forged Assertion

Threat: A MAL or the browser user could forge or alter a SAML assertion (form POST).

Counter-Measure: The POST browser profile requires SAML assertions to be signed, thus providing both message integrity and authentication. The destination site MUST verify the signature and authenticate the issuer.

4.1.4.2.4 Browser State Exposure

Threat: The POST browser profile involve upload of assertions to the web browser from a source site. This information is available as part of the web browser state and is usually stored in persistent storage on the user system in a completely unsecured fashion. The threat here is that the assertion may be "re-used" at some later point in time.

Counter-Measure: Assertions communicated using FORM post must always include a SSO assertion. It is recommended that SSO assertions have short life-times and that destination sites must ensure that they may be used only once.

4.2 SOAP Profile of SAML

4.2.1 Overview

The SOAP profile of SAML is a realization of User Case 3, Scenarios 3-1 and 3-3 of the SAML Requirements document in the context of SOAP. It is based on a single interaction between a *sender* and a *receiver*. The sender adds with one or more SAML assertions to a SOAP document and sends the message to the receiver. The receiver extracts the SAML assertion from the message and processes them. If it is unable to process the assertions it returns an error. Otherwise, it processes the message and assertions in a standard way. The message may be sent over any protocol for which a SOAP protocol binding is available [SOAP1.1].

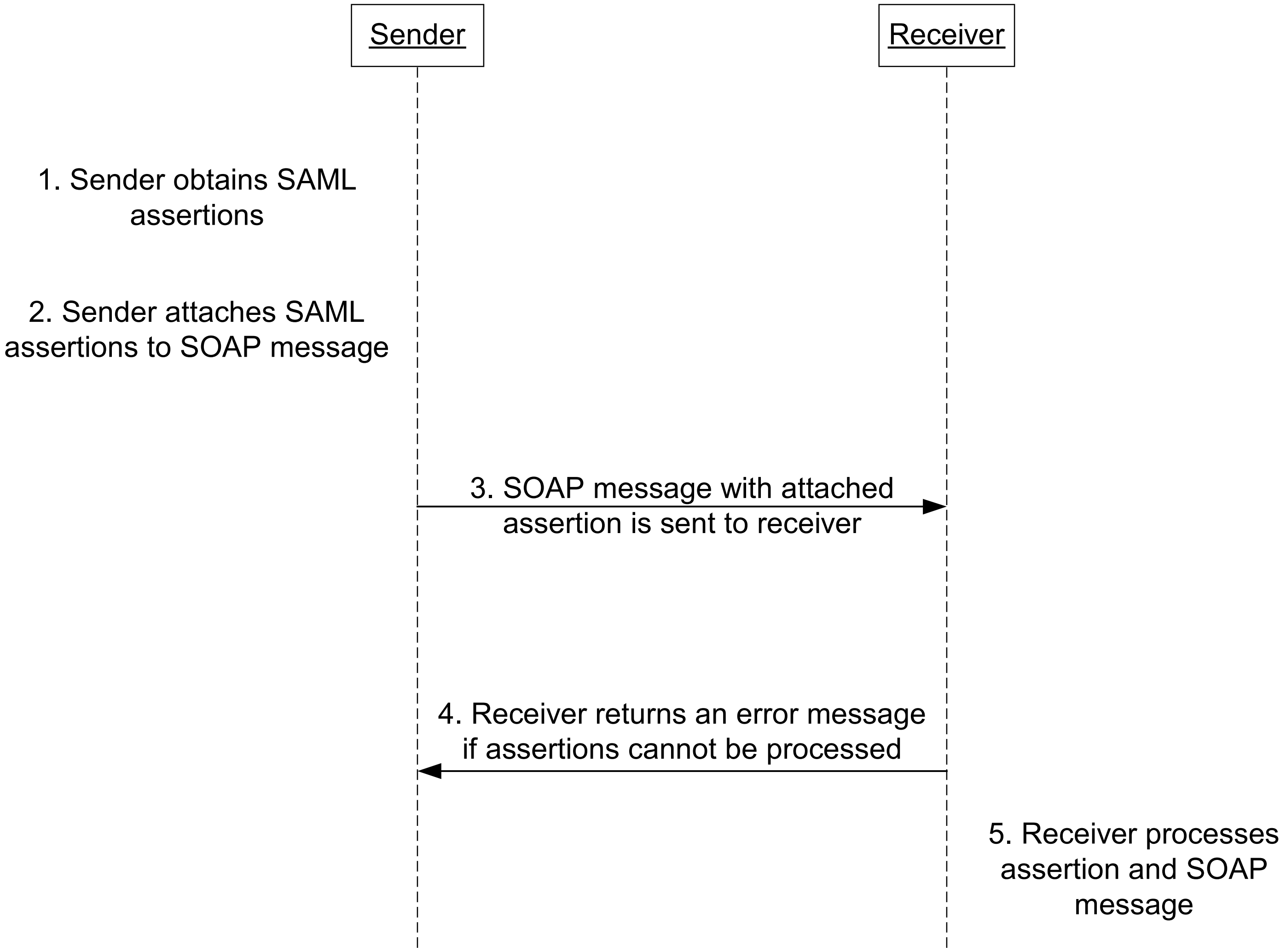

Sender
Receiver
1. Sender obtains SAML assertions
2. Sender attaches SAML assertions to SOAP message
3. SOAP message with attached assertion is sent to receiver
4. Receiver returns an error message if assertions cannot be processed
5. Receiver processes assertion and SOAP message

Figure 4: SOAP Profile of SAML

4.2.2 SOAP Headers

SOAP provides a flexible header mechanism, which may be (optionally) used for extending SOAP payloads with additional information. Rules for SOAP headers are given in Section 4.2 of [SOAP1.1].

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 SAML header is directed must process the SAML assertions as explained in Section 4.2.3 of [SOAP1.1].

4.2.3 SOAP Errors

If the receiver is able to access the SAML assertions contained in the SOAP header, but is unable to process them , the receiver SHOULD return a

SOAP message with a <Fault> element as the message body. Reasons why the

receiver may be able to process SAML assertions, include, but are not limited to:

1. The assertion contains a `<Condition>` element that the receiver does not understand.
2. The signature on the assertion is invalid.
3. The receiver does not accept assertions from the issuer of the assertion in question.
4. The receiver does not have access to extension schema utilized in the assertion.

The returned `<Fault>` element takes the form:

```
<Fault>
     <Faultcode>Client.SAML</Faultcode>
     <Faultstring>...</Faultstring>
</Fault>
```

It is recommended that the <Faultstring> element contain an informative message. This specification does not specify any normative text. Sending parties MUST NOT rely on specific contents in the <Faultstring> element.

4.2.4 Security Considerations

Every assertion MUST be signed by the issuer following the guidelines in [SAML-DSIG-Profile].

Sender and Receiver MUST utilize means to ensure that the data integrity of SOAP messages containing assertions is assured. A number of different techniques are available for providing data integrity including use of SSL, digital signatures, IPsec etc.

When a receiver processes a SOAP message with attached assertions, it MUST make an explicit determination of whether the sender has a right to possess and communicate the attached assertions. Merely obtaining a message containing assertions carries no implication about the sender's right to possess and communicate the included assertions. A variety of means can be used to make such a determination, including, for example, explicit policies at the receiver, authentication of sender, use of digital signature etc.

Two formats for securing the attachment of assertions to an arbitrary SOAP message are described below. Senders and receivers implementing the SOAP Profile of SAML MUST implement both models.

4.2.4.1 HolderOfKey

4.2.4.1.1 Sender

In this case, the sender and subject are the same entity. The sender obtains one or more assertions from one or more authorities. Each assertion MUST include the following <SubjectConfirmation> element:

```
<SubjectConfirmation>
  <ConfirmationMethod>HolderOfKey</ConfirmationMethod>
  <dsig:KeyInfo>…<dsig:KeyInfo>
 </SubjectConfirmation>
```

The <SubjectConfirmation> element carries information about the sender’s key within the <dsig:KeyInfo> element. The <dsig:KeyInfo> provides varied ways for describing information about the sender’s public or secret key.

In addition to the assertions, the sender MUST include an digital signature <dsig:Signature> element within the SOAP <Header> element as described in [XML-DSIG]. The <dsig:Signature> element MUST apply to all the SAML assertion elements in the SOAP <Header>, and all the relevant portions of the SOAP <Body>, as required by the application. Specific applications may require that the signature also apply to additional elements.

4.2.4.1.2 Receiver

The receiver MUST verify that each assertion carries a <SubjectConfirmation> element of the form:

```
<SubjectConfirmation>
  <ConfirmationMethod>HolderOfKey</ConfirmationMethod>
  <dsig:KeyInfo>…<dsig:KeyInfo>
 </SubjectConfirmation>
```

The receiving party MUST check the validity of the signature found in a <SOAP:Envelope>/<dsig:Signature> sub-element of the SOAP message. Information about the sender’s public or secret key may be found in the

<saml:SubjectConfirmation>/<dsig:KeyInfo>

element carried within each assertion.

Notice the <ds:KeyInfo> element is used only for checking integrity of assertion attachment (message integrity). Therefore, there is no requirement that the receiver validate the key or certificate. This suggests that, if needed, a sender may generate a public/private key pair and utilize them for this purpose.

Once the above steps are complete, the receiver may further process the assertions and SOAP message contents with the assurance that portions of the SOAP message covered by the digital signature (a) have been constructed by the sender, (b) have not been altered by an intermediary, (c) the sender has provided proof of possession of the private-key component of the information included in <saml:SubjectConfirmation>/<dsig:KeyInfo>.

4.2.4.1.3 Example

The following example illustrates the HolderOfKey model for securing SAML assertions to a SOAP message:

{PRIVATE "TYPE=PICT;ALT=Figure 3: SOAP document with inserted assertions"}

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Header>
        <saml:AssertionList mustUnderstand="1"
        AssertionID="192.168.2.175.1005169137985" IssueInstant="2001-11-07T21:38:57Z"
        Issuer="M and M Consulting" MajorVersion="1" MinorVersion="0"
        xmlns:saml="http://... /security/docs/draft-sstc-schema-assertion-16.xsd">
        <saml:Conditions NotBefore="2001-11-07T21:33:57Z"
                     NotOnOrAfter="2001-11-07T21:48:57Z">    <saml:AbstractCondition
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="AudienceRestrictionConditionType">
<saml:Audience>http://www.example.com/research_finance_agreement.xml
        </saml:Audience>
      </saml:AbstractCondition>
        </saml:Conditions>
      <saml:AuthenticationStatement AuthenticationInstant="2001-11-07T21:38:57Z"
AuthenticationMethod="Password">
        <saml:Subject>
          <saml:NameIdentifier Name="goodguy" SecurityDomain="www.example.com"/>
<saml:SubjectConfirmation>HolderOfKey</SubjectConfirmation>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyValue>
                ...
                </KeyValue>
                <X509Data>
                ...
                        </X509Data>
      </KeyInfo>
</saml:Subject>
<saml:AuthenticationLocality DNSAddress="some_computer" IPAddress="111.111.111.111"/>
        </saml:AuthenticationStatement>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                <SignedInfo>
                <CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000119"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
                <Reference URI="">
                <Transforms>
        <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>GSUvQSPfYkAC9wpHbLSfPEjMlIo=</DigestValue>
                </Reference>
                </SignedInfo>
                <SignatureValue>
                iLJj64yusw7h4FTbiyKRvAQoALlmeCnKxhKqStrFahVXIZUXacmDJw==
                </SignatureValue>
                <KeyInfo>
                <KeyValue>
                ...
                </KeyValue>
                <X509Data>
                ...
                </X509Data>
        </KeyInfo>
        </Signature>
        </saml:AssertionList>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                <SignedInfo>
                <CanonicalizationMethod  Algorithm="http://www.w3.org/TR/2000/WD-xml-
c14n-20000119"/>
                <SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
                <Reference URI="">
        <Transforms>
                <Transform
                  Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>UYRsLhRffJagF7d+RfNt8CPKhbM=</DigestValue>
        </Reference>
                </SignedInfo>
                <SignatureValue>
                 HJJWbvqW9E84vJVQkjjLLA6nNvBX7mY00TZhwBdFNDEIgscSXZ5Ekw==
               </SignatureValue>
  </Signature>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
        <ReportRequest>
        <TickerSymbol>SUNW</TickerSymbol>
        </ReportRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.2.4.2 SenderVouches

4.2.4.2.1 Sender

In this case, the sender and subject may be distinct entities. The subject obtains one or more assertions from one or more authorities. Each assertion MUST include the following <SubjectConfirmation> element:

```
<SubjectConfirmation>
  <ConfirmationMethod>SenderVouches</ConfirmationMethod>
 </SubjectConfirmation>
```

In this model, information about the sender's key is held within the <dsig:KeyInfo> element associated with the senders signature. The <dsig:KeyInfo> provides varied ways for describing information about the sender's public or secret key.

In addition to the assertions, the sender MUST include an digital signature <dsig:Signature> element within the SOAP <Header> element as described in [XML-DSIG]. The <dsig:Signature> element MUST apply to all the SAML assertion elements in the SOAP <Header>, and all the relevant portions of the SOAP <Body>, as required by the application. Specific applications may require that the signature also apply to additional elements.

The sender MUST include a <dsig:KeyInfo> element with the <dsig:Signature> element.

4.2.4.2.2 Receiver

The receiver MUST verify that each assertion carries a <SubjectConfirmation> element of the form:

```
<SubjectConfirmation>
  <ConfirmationMethod>SenderVouches</ConfirmationMethod>
  </SubjectConfirmation>
```

The receiving party MUST check the validity of the signature found in the <SOAP:Envelope>/<dsig:Signature> element. Information about the sender's public or secret key may be found in the <SOAP:Envelope>/<dsig:Signature>/<dsig:KeyInfo> element carried within each assertion.

Once the above steps are complete, the receiver may further process the assertions and SOAP message contents with the assurance that portions of the SOAP message covered by the digital signature (a) have been constructed by the sender, (b) have not been altered by an intermediary.

4.2.4.2.3 Example

The following example illustrates the SenderVouches architecture for adding SAML assertions to a SOAP message:

```
<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>
              <dsig:signature>…</signature>
</SOAP-ENV:Header>
…
<SOAP-ENV:Body>
   <message_payload/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

{PRIVATE "TYPE=PICT;ALT=Figure 3: SOAP document with inserted assertions"}

4.2.4.3 Additional Security Considerations

The model described in this section does not take into account such issues as replay attacks, authentication of sender by receiver and vice-versa and confidentiality. These must be addressed by means other than those described in this specification.

5 References

[Anders-Browser-Profile] A suggestion on how to implement SAML browser bindings without using “Artifacts”, http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt

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

[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

[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-Marlena] Marlena Erdos, Shibboleth Architecture DRAFT v1.1, http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecturel-00.pdf

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1

[RFC1750] Randomness Recommendations for Security.

[SOAP1.1] 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

[XML-DSIG] XML – Signature Syntax and Processing, available from http://www.w3.org

[WEBSSO] RL “Bob” Morgan, Interactions between Shibboleth and local-site web sign-on services, http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt

[SESSION] RL “Bob” Morgan, Support of target web server sessions in Shibboleth,

http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt

[rfc1945] Hypertext Transfer Protocol -- HTTP/1.0, http://www.ietf.org/rfc/rfc1945.txt

[rfc2616] Hypertext Transfer Protocol -- HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt

[rfc2617] HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt

[rfc2774] An HTTP Extension Framework, http://www.ietf.org/rfc/rfc2774.txt

[RFC2246] The TLS Protocol Version 1.0, http://www.ietf.org/rfcs/rfc2246.html

[SSLv3] The SSL Protocol Version 3.0, http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt

[Rescorla-Security] E. Rescorla, B. Korver, Guidelines for Writing RFC Text on Security Considerations, http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-03.txt

6 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.

<map></map>

7 Appendix B

Javascript may be used to avoid an additional “submit” step from the user. This material is taken from [Anders-Browser-Profile].

```
<HTML>
<BODY Onload="javascript:document.forms[0].submit ()">
<FORM METHOD="POST" ACTION="Destination-site URL">
...
<INPUT TYPE="HIDDEN" NAME="SAMLAssertion" VALUE="Assertion in Base64-
```

```
coding">
</FORM>
</BODY>
</HTML>
```

8 Appendix C

In any SAML use of SSLv3 [SSLv3] or TLS 1.0 [RFC2246], servers MUST authenticate to clients using a X.509.v3 certificate. The client MUST establish server identity based on contents of the certificate (typically through examination of the certificate subject DN field).

8.1 Web Browser Profile

SSL-capable [SSLv3] implementations MUST implement the SSL_RSA_WITH_3DES_EDE_CBC_SHA ciphersuite.

TLS-capable [RFC2246] implementations MUST implement the TLS_RSA_WITH_3DES_EDE_CBC_SHA ciphersuite.

8.2 SAML SOAP Binding

TLS-capable implementations MUST implement the TLS_RSA_WITH_3DES_EDE_CBC_SHA ciphersuite and MAY implement the TLS_RSA_AES_128_CBC_SHA ciphersuite [AES].

Page: 23
[pm1]This needs to be moved elsewhere, perhaps in a mandatory-to-implement section.