SAML 2.0: “Enhanced Client Profiles” Solution Proposal Work Plan Item W-5a

Frederick Hirsch, Nokia Mobile Phones
September 18, 2003
v0.4

Table of Contents

Introduction

Not all potential participants in a single sign-on solution will support cookies and in many cases will not wish to support HTTP redirects or artifact retrieval. This may be true of a range of participants, including users of web browsers on personal computers as well as users of constrained devices such as mobile phones. The SAML technology standard should address such concerns.

The Liberty enabled client and proxy profile has been developed as part of the Liberty Federation Framework with these considerations in mind. A Liberty enabled client (LEC) is one that knows or knows how to find the appropriate identity provider (IDP) for a specific principal and service provider. A Liberty enabled proxy is a proxy that can act as a LEC on behalf of a client. The LECP authentication profile defined in Liberty avoids the use of cookies and redirects, enabling a client to participate in a single sign-on system with minimal impact. The Liberty Federation Framework 1.1 has already been requested and contributed as an input to the Oasis SAML technical committee, making LECP available to the committee.

One example use case can be provided from the mobile domain, a domain that has a currently installed base in the millions. These devices are often constrained in capability, not supporting cookies and having a minimal user interface. Performance is an important issue, making it desirable to reduce the number of protocol exchanges and complicated user interactions. Limiting the need for repeated authentication when accessing web sites and services is valuable for this reason. An example use case is the following simple scenario:

1. The mobile user unlocks their handset using their secret PIN
2. Using the handset web-browser the user accesses a web site that requires authentication. An example might be access to a financial site to obtain a bank or equity trading account balance or recent transaction information.
3. The site requests authentication.
4. The LECP proxy (or LEC handset) is able to obtain an authentication assertion from the IDP. An example IDP is the network operator, possible due to contractual and billing relationships the operator has with the user. The operator network together with handset hardware makes identification possible.
5. The authentication assertion is returned to the site on behalf of the mobile user.
6. The site provides the requested information to the handset user.

This scenario is simple, requiring little interaction with the user since it depends on the mobile network operator relationship to the user. It is enabled through a single sign on process. Stronger authentication might also require additional user interaction, either to accept an authentication action or to authenticate to the network operator IDP, but this does not change the basic operation of the use case.

This technology contribution to SAML 2.0 is the LECP profile as outlined in the Liberty Federation Framework 1.1 in section 3.2.5 of the Liberty FF 1.1 Bindings and Profiles specification [LibertyBindProf] , as well as the schema definitions for the AuthnRequest, AuthnResponse, AuthnRequestEnvelope and AuthnResponseEnvelope elements defined in sections 3.2 of the Liberty FF 1.1 Protocol and Schemas document [LibertyProtSchema].

As additional Liberty specifications are made available to the SAML technical committee then this technology contribution may be updated to reflect any necessary changes.

Definitions

These definitions are compatible with Liberty definitions [LibertyGloss].

Liberty-enabled client or proxy (LECP)
A Liberty-enabled client is a client that has, or knows how to obtain, knowledge about the identity provider that the Principal wishes to use with the service provider. A Liberty-enabled proxy is an HTTP proxy (typically a WAP gateway) that emulates a Liberty-enabled client.

Liberty-enabled client (LEC)
An entity that has, or knows how to obtain, knowledge about the identity provider that the Principal wishes to use with the service provider.

Liberty-Enabled Client and Proxy Profile
This profile specifies interactions between Liberty-enabled clients and/or proxies, service providers, and identity providers.

Liberty-enabled Proxy (LEP)
A Liberty-enabled proxy is a HTTP proxy (typically a WAP gateway) that emulates a Liberty-enabled client.

Liberty-enabled User Agent or Device (LUAD)
A user agent or device that has specific support for one or more profiles of the Liberty specifications. It should be noted that although a standard web browser can be used in many Liberty-specified scenarios, it does not provide specific support for the Liberty protocols, and thus is not a LUAD. No particular claims of specific functionality should be implied about a system entity solely based on its definition as a LUAD. Rather, a LUAD may perform one or more Liberty system entity roles as defined by the Liberty specifications it implements. For example, a LUAD-LECP is a user agent or device that supports the Liberty LECP profile.

Conventions

The following XML namespaces are used in this document and the Liberty specifications:

- The prefix `lib:` stands for the Liberty namespace http://projectliberty.org/schemas/core/2002/12
- The prefix `saml:` stands for the SAML assertion namespace (see [SAMLCore]).
- The prefix `samlp:` stands for the SAML request-response protocol namespace (see [SAMLCore]).
- The prefix `ds:` stands for the W3C XML signature namespace, `http://www.w3.org/2000/09/xmldsig#`
- The prefix `xenc:` stands for the W3C XML encryption namespace, `http://www.w3.org/2001/04/xmlenc#`
- The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace, `http://schemas.xmlsoap.org/soap/envelope` **(see [SOAP1.1]).**

Note: The Liberty namespace may change with Liberty revisions beyond FF 1.1 to reflect schema changes.

General Requirements

Note: This general requirements section is section 3.1 of the Liberty FF 1.1 Bindings and Profiles document, with the section reference removed from item #11.

1. All HTTP requests and responses MUST be drawn from either HTTP 1.1 (see [RFC2616]) or HTTP 1.0 (see [RFC1945]). When an HTTP redirect is specified, the HTTP response MUST have a status code of "302." According to HTTP 1.1 and HTTP 1.0, the use of status code 302 is recommended to indicate "the requested resource resides temporarily under a different URI." The response may also include additional headers and an optional message.
2. When `https` is specified as the `<scheme>` for a URL, the HTTP connection MUST be made over either SSL 3.0 (see [SSLv3]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols that are backwards compatible with SSL 3.0 and/or TLS 1.0. Other security protocols MAY be used as long as they implement equivalent security measures.
3. Messages between providers MUST have their integrity protected, confidentiality MUST be ensured and the recipient MUST authenticate the sender.
4. Providers MUST use secure transport (`https`) to achieve confidentiality and integrity protection. The initiator of the secure connection MUST authenticate the server using server-side X.509 certificates.
5. The authenticated identity of an identity provider MUST be securely available to a Principal before the Principal presents his/her personal authentication data to that identity provider.
6. For signing and verification of protocol messages, identity and service providers SHOULD use certificates and private keys that are distinct from the certificates and private keys applied for SSL or TLS channel protection. Certificates and private keys MUST be suitable for long-term signatures. See [LibertyProtSchema] for guidelines on signature verification.
7. In transactions between service providers and identity providers, requests MUST be protected against replay, and received responses MUST be checked for correct correspondence with issued requests. (**Note:** Other steps may intervene between the issuance of a request and its eventual response within a multistep transaction involving redirections.) Additionally, time-based assurance of freshness MAY be provided.
8. Each service provider within a circle of trust MUST be configured to enable identification of the identity providers whose authentications it will accept, and each identity provider MUST be configured to enable identification of the service providers it intends to serve. (Note: The format of this configuration is a local matter and could, for example, be represented as lists of names or as sets of X.509 certificates of other circle of trust members).
9. Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509 credentials, establishing and managing trusted public keys, and tracking lifecycles of corresponding credentials are assumed and not in scope for this specification.
10. The `<scheme>` of the URL for SOAP endpoints MUST be `https`.
11. All SOAP message exchanges MUST adhere to the SOAP protocol binding for Liberty.

User Agent

A user agent, unless otherwise noted in the specific profile, MUST support the following features to be interoperable with the protocols in [LibertyProtSchema] and Liberty profiles in this document:

1. HTTP 1.0 (see [RFC1945]) or HTTP 1.1 (see [RFC2616]).
2. SSL 3.0 (see [SSL]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols which are backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for example, a WAP gateway).
3. Minimum maximum URL length of 256 bytes. See [LibertyGlossary] for definition.

Liberty-Enabled Client and Proxy Profile

Note: This is section 3.2.5 of the Liberty FF 1.1 Bindings and Profile document. Section and figure references have been removed or updated.

The Liberty-enabled client and proxy profile specifies interactions between Liberty-enabled clients and/or proxies, service providers, and identity providers. See Figure 1. A Liberty-enabled client is a client that has, or knows how to obtain, knowledge about the identity provider that the Principal wishes to use with the service provider. In addition a Liberty-enabled client receives and sends Liberty messages in the body of HTTP requests and responses. Therefore, Liberty-enabled clients have no restrictions on the size of the Liberty protocol messages.

A Liberty-enabled proxy is a HTTP proxy (typically a WAP gateway) that emulates a Liberty-enabled client. Unless stated otherwise, all statements referring to LECP are to be understood as statements about both Liberty-enabled clients as well as Liberty-enabled proxies.

The following URI-based identifier must be used when referencing this specific profile (for example, `<lib:ProtocolProfile>` element of the `<lib:AuthnRequest>` message)

URI: `http://projectliberty.org/profiles/lecp`

All LECPs, in addition to meeting the common requirements for profiles , MUST indicate that it is a LECP by including a Liberty-Enabled header or entry in the value of the HTTP User-Agent header for each HTTP request they make. The preferred method is the Liberty-Enabled header. The formats of the Liberty-Enabled header and User-Agent header entry are defined later in this document.

Interactions

Note: This is section 3.2.5.2 of the Liberty FF 1.1 Bindings and Profile document. Section and figure references have been removed or updated. Material from 3.2.1 has been integrated with the steps as required and the references to that section removed (These additions are noted with “General recommendations for all profiles :” notes). Added ProviderID requirement in step 7.

Figure 1 illustrates the Liberty-enabled client and proxy profile for single sign-on. All generic steps are shown - steps that are light are not necessary in the LECP profile (steps 2, 8, 9, 10 in the figure):

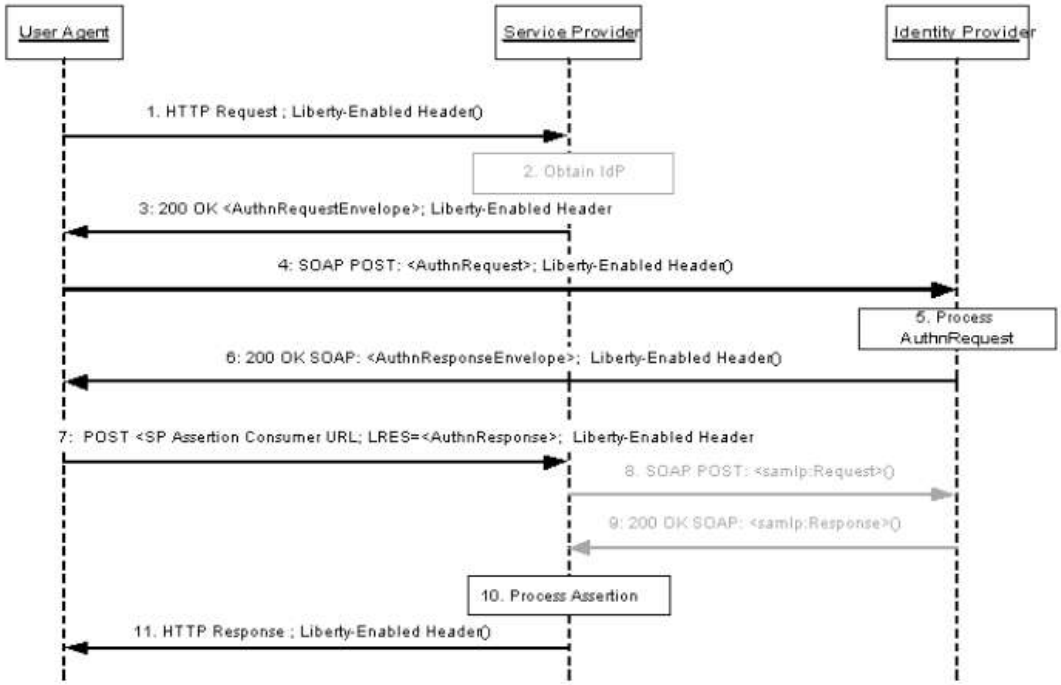

Figure 1. Liberty-enabled client and proxy profile for single sign-on

This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1. Thus, a valid session exists for the user agent at the identity provider.

The LECP receives authentication requests from the service provider in the body of the HTTP response. The LECP submits this authentication request as a SOAP request to the identity provider. Because this SOAP request is between the LECP and the identity provider, TLS authentication cannot be performed between service provider and identity provider; therefore, service providers and identity providers MUST rely on the signature of the `<lib:AuthnRequest>` and the returned `<saml:Assertion>`, respectively, for mutual authentication. ~~(When implementing this profile, processing rules for steps 5, 10, and 11 defined in 3.2.1 for single sign-on profiles MUST be followed, while steps 2, 8, and 9 MUST be omitted. Additionally, the following rules MUST be observed as they relate to steps 1, 3, 4, 6, and 7)~~

Step 1: Accessing the Service Provider

In step 1, the user agent accesses the service provider with the Liberty-Enabled header (or with the Liberty-Enabled entry in the User-Agent header) included in the HTTP request.

The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives a HTTP request that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled header. However, a proxy MAY replace the Liberty-Enabled header. A proxy that replaces or adds a Liberty-Enabled header MUST process `<lib:AuthnRequest>` messages as defined in steps 3 and 4 as well as `<lib:AuthnResponse>` messages as specified in steps 6 and 7.

It is RECOMMENDED that a LECP add "`application/vnd.liberty-request+xml`" as one of its supported content types to the Accept header.

General recommendations for all profiles :

It is RECOMMENDED that the HTTP Request URI contain a <query> component at its end

where

<query>=...RelayState=<return URL>...

The <query> component can be used to convey information about the originally requested resource at the service provider. It is RECOMMENDED that the <query> parameter be named RelayState and its value be the URL originally requested by the user agent.

It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see[SSLv3]) or TLS 1.0 (see [RFC2246]) to maintain confidentiality and message integrity in step 1.

Step 3: HTTP Response with <AuthnRequest>

In step 3, the service provider's intersite transfer service issues an HTTP 200 OK response to the user agent. The response MUST contain a single `<lib:AuthnRequestEnvelope>` with content as defined in [LibertyProtSchema]. If a service provider receives a Liberty-Enabled header, or a User-Agent header with the Liberty-Enabled entry, the service provider MUST respond according to the Liberty-enabled client and proxy profile and include a Liberty_Enabled header in its response. Hence service providers MUST support the Liberty-enabled client and proxy profile.

The processing rules and default values for the Liberty-Enabled indications are as defined in the following section on Indications. The service provider MAY advertise any Liberty version supported in this header, not only the version used for the specific response.

The HTTP response MUST contain a Content-Type header with the value `application/vnd.liberty-request+xml` unless the LECP and service provider have negotiated a different format.

A service provider MAY provide a list of identity providers it recognizes by including the `<lib:IDPList>` element in the `<lib:AuthnRequestEnvelope>`. The format and processing rules for the identity provider list MUST be as defined in [LibertyProtSchema].

Note

In cases where a value for the <lib:GetComplete> element is provided within <lib:IDPList>, the URI value for this element MUST specify https as the URL <scheme>.

The service provider MUST specify a URL for receiving `<AuthnResponse>` elements, locally generated by the intermediary, by including the `<lib:AssertionConsumerServiceURL>` element in the `<lib:AuthnRequestEnvelope>`.

The following example demonstrates the usage of the `<lib:AuthnRequestEnvelope>`:

```
<?xml version="1.0" ?>
<lib:AuthnRequest Envelope xmlns:lib="http://projectliberty.org/schemas/core/2002/12/">
<lib:AuthnRequest >
. . . AuthnRequest goes here . . .
</lib:AuthnRequest>
<lib:AssertionConsumerServiceURL>
https://service-provider.com/LibertyLogin
</lib:AssertionConsumerServiceURL>
<lib:IDPList>
. . . IdP list goes here . . .
</lib:IDPList>
</lib:AuthnRequestEnvelope>
```

If the service provider does not support the LECP-advertised Liberty version, the service provider MUST return to the LECP an HTTP 501 response with the reason phrase "Unsupported Liberty Version."

The responses in step 3 and step 6 SHOULD NOT be cached. To this end service providers and identity

providers SHOULD place both "`Cache-Control: no-cache`" and "`Pragma: no-cache`" on their responses to ensure that the LECP and any intervening proxies will not cache the response.

Step 4: HTTP Request with <AuthnRequest>

In step 4, the LECP determines the appropriate identity provider to use and then issues an HTTP POST of the `<lib:AuthnRequest>` in the body of a SOAP message to the identity provider's single sign-on service URL. The request MUST contain the same `<lib:AuthnRequest>` as was received in the `<lib:AuthnRequestEnvelope>` from the service provider in step 3.

> *Note*
>
> The identity provider list can be used by the LECP to create a user identifier to be presented to the Principal. For example, the LECP could compare the list of the Principal's known identities (and the identities of the identity provider that provides those identities) against the list provided by the service provider and then only display the intersection.

If the LECP discovers a syntax error due to the service provider or cannot proceed any further for other reasons (for example, cannot resolve identity provider, cannot reach the identity provider, etc), the LECP MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the desired error element as defined in [LibertyProtSchema]. The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service provider's assertion consumer service URL obtained from the `<lib:AssertionConsumerServiceURL>` element of the `<lib:AuthnRequestEnvelope>`. The POST MUST be a form that contains the field LARES with the value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema], containing the `<samlp:Status>`. The `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the `<lib:AuthnResponse>` and all its elements.

General recommendations for all profiles:

In step 5, the identity provider MUST process the <lib:AuthnRequest> message according to the rules specified in [LibertyProtSchema].

If the Principal has not yet been authenticated with the identity provider, authentication at the identity provider MAY occur in this step. The identity provider MAY obtain consent from the Principal for federation, or otherwise consult the Principal. To this end the identify provider MAY return to the HTTP request any HTTP response; including but not limited to HTTP Authentication, HTTP redirect, or content. The identity provider SHOULD respect the HTTP User-Agent and Accept headers and SHOULD avoid responding with content-types that the User-Agent may not be able to accept. Authentication of the Principal by the identity provider is dependent upon the <lib:AuthnRequest> message content.

In case the identity provider responds to the user agent with a form, it is RECOMMENDED that the <input> parameters of the form be named according to [RFC3106] whenever possible.

Step 6: HTTP Response with <AuthnResponse>

In step 6, the identity provider responds to the `<lib:AuthnRequest>` by issuing an HTTP 200 OK response. The response MUST contain a single `<lib:AuthnResponseEnvelope>` in the body of a SOAP message with content as defined in [LibertyProtSchema].

In step 6, the identity provider responds to the `<lib:AuthnRequest>` by issuing an HTTP 200 OK response. The response MUST contain a single `<lib:AuthnResponseEnvelope>` in the body of a SOAP message with content as defined in [LibertyProtSchema].

The identity provider MUST include the Liberty-Enabled HTTP header following the same processing rules

as defined in the section on Indications.

The Content-Type MUST be set to application/vnd.liberty-response+xml.

If the identity provider discovers a syntax error due to the service provider or LECP or cannot proceed any further for other reasons (for example, unsupported Liberty version), the identity provider MUST return to the LECP a `<lib:AuthnResponseEnvelope>` containing a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the desired error element as defined in [LibertyProtSchema].

Step 7: Posting the Form Containing the <AuthnResponse>

In step 7, the LECP issues an HTTP POST of the `<lib:AuthnResponse>` that was received in the `<lib:AuthnResponseEnvelope>` SOAP response in step 6. The `<lib:AuthnResponse>` MUST be sent using a POST to the service provider's assertion consumer service URL identified by the `<lib:AssertionConsumerServiceURL>` element within the `<lib:AuthnResponseEnvelope>` *obtained from the identity provider in step 6.* The POST MUST be a form that contains the field `LARES` with the value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema]. The `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to [RFC2045]) to the `<lib:AuthnResponse>` and all its elements. The service provider's assertion consumer service URL used as the target of the form POST MUST specify `https` as the URL scheme; if another scheme is specified, it MUST be treated as an error by the identity provider.

If the LECP discovers an error (for example, syntax error in identity provider response), the LECP MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the appropriate error element as defined in [LibertyProtSchema].

The `<ProviderID>` in the `<lib:AuthnResponse>` MUST be set to *urn:liberty:iff:lecp*.

The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service provider's assertion consumer service URL. The POST MUST be a form that contains the field named LARES with its value being the `<lib:AuthnResponse>` protocol message as defined in [LibertyProtSchema] with formatting as specified in the Liberty URL-encoding rules (section 3.1.2). Any `<lib:AuthnResponse>` messages created by the identity provider MUST not be sent to the service provider.

Step 10: Process Assertion

This is not a LECP specific step, but a general profile processing step:

In step 10, the service provider processes the <saml:Assertion> returned in the <samlp:Response> or <lib:AuthnResponse> protocol message to determine its validity and how to respond to the Principal's original request. The signature on the <saml:Assertion> must be verified.

The service provider processing of the assertion MUST adhere to the rules defined in [SAMLCore] for things such as assertion <saml:Conditions> and <saml:Advice>.

The service provider MAY obtain authentication context information for the Principal's current session from the <lib:AuthnContext> element contained in the <saml:advice>. Similarly, the information in the <lib:RelayState> element MAY be obtained and used in further processing by the service provider.

Liberty-Enabled Indications

Note: This is section 3.2.5.1 of the Liberty FF 1.1 Bindings and Profiles document. Added text to anticipate FF 1.2 submission (See this).

A LECP SHOULD add the Liberty-Enabled header to each HTTP request. The Liberty-Enabled header MUST be named Liberty-`Enabled` and be defined as using Augmented BNF as specified in section 2 of [RFC2616].

```
Liberty-Enabled = "Liberty-Enabled" ":" LIB_Version ["," 1 #Extension]
LIB_Version = "LIBV" "=" 1*absoluteURI
; any spaces or commas in the absoluteURI MUST be escaped as defined in section 2.4 of [RFC 2396]
Extension = ExtName "=" ExtValue
ExtName = (["." host] | <any field-value but ".", "," or " =">) <any field-value but "=" or ",">
ExtValue = <any field-value but ",">)
```

The comment, field-value, and product productions are defined in [RFC2616]. `LIB_Version` identifies the versions of the Liberty specifications that are supported by this LECP. Each version is identified by a URI. Service providers or identity providers receiving a Liberty-Enabled header MUST ignore any URIs listed in the `LIB_Version` production that they do not recognize. All LECPs compliant with this specification MUST send out, at minimum, the URI `http://projectliberty.org/specs/v1` as a value in the `LIB_Version` production. *It SHOULD precede this with the URI* `urn:liberty:iff:1.2` *if it supports version 1.2 requests and knows that the identity providers available to it also support version 1.2 requests and responses. It MUST NOT include this URI if it knows that the identity providers available to it cannot process version 1.2 messages.* The ordering of the URIs in the `LIB_Version` header is meaningful; therefore, service providers and identity providers are encouraged to use the first version in the list that they support. Supported Liberty versions are not negotiated between the LECP and the service provider. The LECP simply advertises what version it does support, and the service provider MUST return the response for the corresponding version as defined in step 3 below.

Optional extensions MAY be added to the Liberty-Enabled header to indicate new information. The value of the `ExtName` production MUST use the "`host`" "`;`" prefixed form if the new extension name has not been standardized and registered with Liberty or its designated registration authorities. The value of the host production MUST be an IP or DNS address that is owned by the issuer of the new name. By using the DNS/IP prefix, namespace collisions can be effectively prevented without having to introduce yet another centralized registration agency.

LECPs MAY include the Liberty-Agent header in their requests. This header provides information about the software implementing the LECP functionality and is similar to the User-Agent and Server headers in HTTP.

Liberty-Agent = "Liberty-Agent" ":" 1 (product | comment)*

> *Note*
>
> The reason for introducing the new header (that is, Liberty-Enabled) rather than just using User-Agent is that LECP may be a Liberty-enabled proxy. In that case the information about the Liberty-enabled proxy would not be in the User-Agent header. In theory the information could be in the VIA header. However, for security reasons, values in the VIA header can be collapsed, and comments (where software information would be recorded) can always be removed. As such, the VIA header is not suitable. Using the User-Agent header for a Liberty-enabled client and the Liberty-Agent header for a Liberty-enabled proxy was also discussed. However, this approach seemed too complex.
>
> Originally the Liberty-Agent header was going to be part of the Liberty-Enabled header. However, header lengths in HTTP implementations are limited; therefore, putting this information in its own header was considered the preferred approach.

A LECP MAY add a Liberty-Enabled entry in the HTTP User-Agent request header. The HTTP User-Agent header is specified in [RFC2616]. A LECP MAY include in the value of this header the `Liberty-Enabled` string as defined above for the Liberty-Enabled header.

> *Note*
>
> The reason for adding information to the User-Agent header is to allow for Liberty-enabled client products that must rely on a platform that cannot be instructed to insert new

headers in each HTTP request.

The User-Agent header is often overloaded; therefore, the Liberty-Enabled header should be the first choice for any implementation of a LECP. The entry in the User-Agent header then remains as a last resort.

Processing Rules for Active Intermediaries

Note: This is from section 3.2.3.1 of the Liberty FF 1.1 Protocols and Schema document.

For all profiles specifying an active intermediary, the intermediary MUST follow these processing rules:

- If the profile specifies that the message sent from the service provider to the identity provider, via the intermediary, is wrapped in an <AuthnRequestEnvelope>:
    - The intermediary MUST remove the enveloping <AuthnRequestEnvelope> before forwarding the <AuthnRequest> element to the identity provider.

• The intermediary MAY locally generate <AuthnResponse> elements and send them to the service provider using the <AssertionConsumerServiceURL> contained within the <AuthnRequestEnvelope>. Such <AuthnResponse> elements MUST NOT contain any <lib:Assertion> elements. The <AuthnResponse> elements MUST have an InResponseTo attribute set to the RequestID of the <AuthnRequest> that could not be serviced. If the <AuthnRequest> contained a <RelayState> element, the <AuthnResponse> MUST include a <RelayState> element with its value set to that supplied in the <AuthnRequest>. Such responses MAY be generated as a result of local errors on the intermediary, and should indicate the underlying reasons in the <samlp:Status> element in the <AuthnResponse>. The following are error conditions for which second-level <samlp:StatusCode> values are defined below:

• The identity provider cannot be reached
• There is no identity provider in common between the intermediary and the service provider

• If the profile specifies that the message from the identity provider to the service provider, via the intermediary, is wrapped in an <AuthnResponseEnvelope>:

• The intermediary MUST remove the enveloping <AuthnResponseEnvelope> before forwarding the <AuthnResponse> element to the service provider.
• The intermediary MUST send <AuthnResponse> messages received from the identity provider to the service provider using the <AssertionConsumerServiceURL> contained within the <AuthnResponseEnvelope> sent by the identity provider.

Status Code Values for Error Conditions

If an error occurs in the processing at the intermediary, the following values are defined for use in second-level <samlp:StatusCode> elements:

• lib:NoAvailableIDP: Used to indicate that none of the supported identity provider URLs from the <IDPList> can be resolved or that none of the supported identity providers are available.
• lib:NoSupportedIDP: Used to indicate that none of the identity providers are supported by the intermediary.

XML Schema Definitions

Authentication Envelope

Request Envelope

Note: This is section 3.2.4 of the Liberty FF 1.1 Protocol and Schema document, with material removed as appropriate since this section is specific to the LECP profile. IDPList element optionality is clarified by putting Optional, Required after element definitions as noted in FF 1.2.

Note that in FF 1.2 the Extension element is defined rather than using XML schema "any" with *processContents="skip"* as in this contribution and the FF 1.1 schema definitions. This is relevant to the RequestEnvelopeType and ResponseEnvelopeType elements [LibertyProtSchema1_2].

The LECP profile wraps the `<AuthnRequest>` element in an envelope. This envelope allows for extra processing by the LEC intermediary between the service provider and the identity provider. ~~An example of an intermediary is a user agent or proxy.~~ Processing rules are given in the section on Intermediary Processing Rules. Note that the envelope is for consumption by the intermediary and is removed before the enveloped `<AuthnRequest>` element is forwarded to the identity provider.

Element <AuthnRequestEnvelope>

The authentication request envelope contains the following elements:

`AuthnRequest` [Required]
The authentication request contained within the envelope.

`ProviderID` [Required]
The requester's ProviderID.

`ProviderName` [Optional]
The human-readable name of the requester.

`AssertionConsumerServiceURL` [Required]
A URL specifying where `<AuthnResponse>` elements, locally generated by an intermediary, should be sent. See the processing rules for active intermediaries specified in the section on Intermediary Processing Rules.

`IDPList` [Optional]
A list of identity providers, from which, one may be chosen to service the authentication request.

`IsPassive` [Optional]
If "true," specifies that any intermediary between the service provider and identity provider MUST NOT interact with the Principal. If not specified, "true" is presumed.

The schema fragment is as follows:

```
<xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType"/>
<xs:complexType name="AuthnRequestEnvelopeType">
<xs:complexContent>
<xs:extension base="RequestEnvelopeType">
<xs:sequence>
<xs:element ref="AuthnRequest"/ >
<xs:element ref="ProviderID"/>
<xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
<xs:element name=" AssertionConsumerServiceURL" type="xs:anyURI"/>
<xs:element ref="IDPList" minOccurs="0"/>
<xs:element name= "IsPassive" type="xs:boolean" minOccurs="0"/>
</xs:sequence>
</xs:extension>
```

```
</xs:complexContent>
</xs:complexType>
<xs:complexType name="RequestEnvelopeType">
<xs:sequence>
<any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
```

Element <IDPList>

In the request envelope, some profiles may wish to allow the service provider to transport a list of identity providers to the user agent. This specification provides a schema that profiles SHOULD use for this purpose. The elements are as follows:

`IDPList` [Optional]
The container element for an IDP List.

`IDPEntries` [Required]
Contains a list of identity provider entries.

`IDPEntry` [Required]
Describes an identity provider that the service provider supports.

`ProviderID` [Required]
The identity provider's unique identifier.

`ProviderName` [Optional]
The identity provider's human-readable name.

`Loc` [Optional]
The identity provider's URI, to which authentication requests may be sent. If present, this MUST be set to the value of the identity provider's `<SingleSignOnService>` element, obtained from their metadata ([LibertyMetadata]).

GetComplete [Optional]
If the identity provider list is not complete, this element may be included with a URI that points to where the complete list can be retrieved.

The schema fragment is as follows:

```
<xs:element name="IDPList " type="IDPListType"/>
<xs:complexType name="IDPListType">
<xs:sequence>
<xs:element ref="IDPEntries"/>
<xs:element ref="GetComplete" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:element name="IDPEntry">
<xs:complexType>
<xs:sequence>
<xs:element ref="ProviderID"/>
<xs:element name="ProviderName" type="xs:string" minOccurs=" 0"/>
<xs:element name="Loc" type="xs:anyURI"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="IDPEntries">
<xs:complexType>
<xs:sequence>
<xs:element ref="IDPEntry " maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="GetComplete" type="xs:anyURI"/>
```

Example

```
<AuthnRequestEnvelope>
<AuthnRequest> ... </AuthnRequest>
<ProviderID>http://ServiceProvider.com</ProviderID>
<ProviderName>Service Provider X</ProviderName>
<AssertionConsumerServiceURL>http://ServiceProvider.com/lecp_assertion_consumer
</AssertionConsumerServiceURL>
<IDPList>
<IDPEntries>
<IDPEntry>
<ProviderID>http://IdentityProvider.com</ProviderID>
<ProviderName>Identity Provider X</ProviderName>
<Loc>http://www.Id entityProvider.com/liberty/sso</Loc>
</IDPEntry>
</IDPEntries>
<GetComplete>https://ServiceProvider.com/idpli st?id=604be136-fe91-441e-afb8-f88748ae3b8b
</GetComplete>
</IDPList>
<IsPassive>0</IsPassive>
</AuthnRequestEnvelope>
```

Response Envelope

As with the `<AuthnRequest>` element, some profiles MAY wrap the `<AuthnResponse>` element in an envelope. This envelope allows for extra processing by an intermediary (such as a user agent or proxy) between the identity provider and the service provider. Applicable processing rules are given in the section on Intermediary Processing Rules. Note that the envelope is for consumption by the intermediary and is removed prior to the forwarding of the enveloped `<AuthnResponse>` element to the service provider.

Element <AuthnResponseEnvelope>

The authentication response envelope contains the following elements:

`Extension` [Optional]

Optional container for protocol extensions established by agreement between service providers and intermediaries. Implementors should note that this element may not contain content from the core Liberty namespace (which is prevented at the schema level by requiring namespace="##other").

`AuthnResponse` [Required]
The enveloped authentication response.

`AssertionConsumerServiceURL` [Required]
The service provider's URL where the authentication response should be sent. This element's value SHOULD be obtained from the element of the same name in the service provider's Provider Metadata.

The schema fragment is as follows:

```
<xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
<xs:complexType name="AuthnResponseEnvelopeType">
<xs:complexContent>
<xs:extension base="ResponseEnvelopeType">
<xs:sequence>
<xs:element ref="AuthnResponse"/>
<xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="ResponseEnvelopeType">
<xs:sequence>
<any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
```

3.2.4.2. Example

```
<AuthnResponseEnvelope>
<AuthnResponse> ... </AuthnResponse>
<AssertionConsumerServiceURL>
http://ServiceProvider.com/lecp_assertion_consumer
```

```
</AssertionConsumerServiceURL>
</AuthnResponseEnvelope>
```

Authentication request and response

Note: This is from section 3.2 of Liberty FF 1.1 Protocols and Schemas. Added clarifying text on multiple audience elements (line 691), and multiple statements (line 697).

Element <AuthnRequest>

The <AuthnRequest> is defined as an extension of **samlp:RequestAbstractType**. The RequestID attribute in **samlp:RequestAbstractType** has uniqueness requirements placed on it by, which require it to have the properties of a nonce.

The elements of the request are as follows:

`ProviderID` [Required]
The requester's unique identifier.

`IsPassive` [Optional]
If "true," specifies that the identity provider MUST NOT interact with the Principal and MUST NOT take control of the user interface from the service provider. If "false," the identity provider MAY interact with the user and MAY temporarily take control of the user interface for that purpose. If not specified, "true" is presumed.

`ForceAuthn` [Optional]
Controls whether the identity provider authenticates the Principal regardless of whether the Principal is already authenticated. This element is specified only when `<IsPassive>` is "false." If `<ForceAuthn>` is "true," specifies that the identity provider MUST always authenticate the Principal, regardless of whether the Principal is presently authenticated. If "false," specifies that the identity provider MUST re-authenticate the user only if the Principal is not presently authenticated. If not specified, "false" is presumed. The protocol profile that the requester wishes to use for the response. If the element is not specified, the default protocol profile is *http://projectliberty.org/profiles/brws-art*, defined in [LibertyBindProf].

Federate [Optional]
Specifies that the service provider wishes to federate the Principal's identity at the service provider with the Principal's identity at the identity provider. If the element is not specified, it is presumed that the service provider does not wish to federate the identity.

ProtocolProfile [Optional]
The protocol profile that the service provider wishes to use for the response. If the element is not specified, the default protocol profile is http://projectliberty.org/profiles/brws-art, defined in [LibertyBindProf].

`AuthnContext` [Optional]
Information describing which authentication context the requester desires the identity provider to use in authenticating the Principal.

`RelayState` [Optional]
This contains state information that will be relayed back in the response. This data SHOULD be integrity protected by the request author and MAY have other protections placed on it by the request author. An example of such protection is confidentiality.

id [Optional]
Identifier used to identify this element in the signature. See section 3.1.5, Signature 351 Verification for more information.

`AuthnContextComparison` [Optional]
If set to "exact", then the identity provider is asked to match at least one of the specified `<AuthnContext>` elements exactly. This can also be set to "minimum", which asks that the identity provider use a context that

he feels is at least as good as any specified in the `<AuthnContext>` or "better", which means that the they can use any context better than any that were supplied. If not specified, this is assumed to be "exact".
The `<Scoping>` element may contain the following elements:

AuthnContextClassRef [Optional]
The ordered set of authentication context class references the service provider desires the identity provider to use in authenticating the Principal.

AuthnContextStatementRef [Optional]
The ordered set of exact authentication statements the service provider desires the identity provider to use in authenticating the Principal.

The schema fragment defining the element and its type is as follows:

```
<element name="AuthnRequest" type="lib:AuthnRequestType"/>
<complexType name="AuthnRequestType">
<complexContent>
<extension base="samlp:RequestAbstractType">
<sequence>
<element ref="lib:ProviderID"/>
<element name="ForceAuthn" type="boolean" minOccurs="0"/>
<element name="IsPassive" type="boolean" minOccurs="0"/>
<element name="Federate" type="boolean" minOccurs="0"/>
<element ref="lib:ProtocolProfile" minOccurs="0"/>
<element ref="lib:AuthnContext" minOccurs="0"/>
<element ref="lib:RelayState" minOccurs="0"/>
<element name="AuthnContextComparison" type="lib:AuthnContextComparisonType" minOccurs="0"
maxOccurs="1"/>
</sequence>
<attribute name="id" type="ID" use="optional"/>
</extension>
</complexContent>
</complexType>
<simpleType name="AuthnContextComparisonType">
<restriction base="string">
<enumeration value="exact"/>
<enumeration value="minimum"/>
<enumeration value="better"/>
</restriction>
</simpleType>
<element name="RelayState"/>
<element name="ProtocolProfile" type="anyURI"/>
<element name="AuthnContext">
<complexType>
<choice>
<element name="AuthnContextClassRef" type="anyURI" maxOccurs="unbounded"/>
<element name="AuthnContextStatementRef" type="anyURI" maxOccurs="unbounded"/>
</choice>
</complexType>
</element>
```

Element `<AuthnResponse>`

The response is either an `<AuthnResponse>` element containing a set of authentication assertions or a set of artifacts the service provider can dereference into a set of authentication assertions.

All authentication assertions generated by an identity provider for a service provider MUST be of type **`AssertionType`**. The `<Subject>` element in any subject statement MUST be of type **`SubjectType`**. If the service provider registered a name identifier for the Principal (see 3.3), the <saml:NameIdentifier> element in the <saml:Subject> element MUST be the service provider-provided name identifier for the Principal. Otherwise, <saml:NameIdentifier> MUST be the most current name identifier supplied by the identity provider. The <IDPProvidedNameIdentifier> MUST contain the most recent name identifier supplied by the identity provider.

All authentication statements MUST be of type **AuthenticationStatementType**.

Identity providers MUST include a `<saml:AudienceRestrictionCondition>` element that specifies the intended consumers of the assertion. One `<saml:Audience>` element MUST be set to the intended recipient's `ProviderID`. The recipient MUST validate that it is the intended viewer before using the assertion.

The assertion MAY contain additional `<saml:Audience>` elements that specify other intended relying parties.

Identity providers MAY include a `SessionIndex` attribute in resulting authentication statements, which is used to aid the identity provider in managing multiple sessions with the Principal. If the identity provider includes this `SessionIndex` attribute, subsequent messages from the service provider to the identity provider that are session- dependent MUST include this `SessionIndex` attribute.

Identity providers MAY include other types of statements in the assertion(s) returned, depending on agreements between providers and other specifications that provide additional functionality. Any such statements that include a name identifier representing the Principal MUST be consistent with the identification semantics dictated by the `<NameIDPolicy>` element. This is particularly relevant if the "onetime" policy is in effect; a temporary identifier or an otherwise obfuscated and protected value MUST be used.

Each assertion in the `<AuthnResponse>` message MUST be individually signed by the identity provider (that is, each assertion must contain a Signature element which signs only the assertion). It is RECOMMENDED that the signature be omitted from the `<AuthnResponse>` itself, but signing of the message is not forbidden.

The type AuthnResponseType is extended from samlp:ResponseType. The response contains the following elements:

`ProviderID` [Required]
The identity provider's unique identifier.

`RelayState` [Optional]
This contains state information being relayed.

id [Optional]
Identifier used to identify this element in a signature. See section 3.1.5, Signature Verification for more information.

The schema fragment is as follows:

```
<element name="AuthnResponse" type="lib:AuthnResponseType"/>
<complexType name="AuthnResponseType">
<complexContent>
<extension base="samlp:ResponseType">
<sequence>
<element ref="lib:ProviderID"/>
<element ref="lib:RelayState" minOccurs="0"/>
</sequence>
```

```
<attribute name="id" type="ID" use="optional"/>
</extension>
</complexContent>
</complexType>
```

Security Considerations

When the LECP profile is used with a Liberty Enabled Proxy (LEP) and WAP 1.0 then the LEP functionality may be integrated with the WAP gateway. This does not introduce a significant change to the WAP 1.0 security model which already does not allow end-end SSL/TLS connections through it.

When using WAP 2.0 which allows SSL/TLS to pass through it, use of a LEP requires the WAP gateway to terminate SSL/TLS so that HTTP headers may be modified at the gateway to support LECP. This can impact service provider attempts to use an end-end SSL/TLS connections with the terminal unless additional care is taken.

References

[LibertyBindProf] Rouault, J., & Wason, T., eds. (January 2003). "Liberty Bindings and Profiles Specification," Version 1.1. Liberty Alliance Project, <http://www.projectliberty.org/specs/>.

[LibertyGloss] Mauldin, H., & Wason, T., eds. (January 2003). "Liberty Architecture Glossary," Version 1.1. Liberty Alliance Project, <http://www.projectliberty.org/specs/>.

[LibertyProtSchema] Beatty, J., & Kemp, J., eds. (January 2003). "Liberty Protocols and Schema Specification," Version 1.1. Liberty Alliance Project, <http://www.projectliberty.org/specs/>.

[LibertyProtSchema1_2] Cantor, S. & Kemp, J., eds. (July 2003). "Liberty Protocols and Schema Specification," Version 1.2-14. Liberty Alliance Project, <https://www.projectliberty.org/specs/draft-lib-arch-protocols-schema-v1.2-14.pdf>

[RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H. (May 1996). "Hypertext Transfer Protocol -- HTTP/1.0," RFC 1945. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc1945.txt> [18 December 2002].

[RFC2045] Freed, N., Borenstein, N. (November 1996). "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc2045.txt> [18 December 2002].

[RFC2246] Dierks, T.,& Allen, C. (January 1999). "The TLS Protocol Version 1.0," RFC 2246. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc2246.txt> [18 December 2002]

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (June 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc2616.txt> [18 December 2002].

[RFC3106] Eastlake, D., & Goldstein, T. (April 2001). "ECML v1.1: Field Specifications for E-Commerce," RFC 3106. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc3106.txt> [18 December 2002].

[SAMLCore] Hallam-Baker, P., Maler, E., eds. (05 Nov. 2002). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard. Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/committees/security/#documents> [18 December 2002].

[SOAP1.1] D. Box et al. (May 2000). "Simple Object Access Protocol (SOAP) 1.1," Note. World Wide Web Consortium, <http://www.w3.org/TR/SOAP> [18 December 2002].

[SSLv3] Freier, A. O., Karlton, P., & Kocher, P. (November 1996). "The SSL Protocol," Version 3.0, Internet Draft 02. Internet Engineering Task Force, <http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt> [18 December 2002].