A SASL and GSS-API Mechanism for SAML
draft-ietf-kitten-sasl-saml-03.txt

Abstract

Security Assertion Markup Language (SAML) has found its usage on the Internet for Web Single Sign-On. Simple Authentication and Security Layer (SASL) and the Generic Security Service Application Program Interface (GSS-API) are application frameworks to generalize authentication. This memo specifies a SASL mechanism and a GSS-API mechanism for SAML 2.0 that allows the integration of existing SAML Identity Providers with applications using SASL and GSS-API.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 17, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
   1.1. Terminology
   1.2. Applicability
2. Applicability for non-HTTP Use Cases
3. SAML SASL Mechanism Specification
   3.1. Initial Response
   3.2. Authentication Request
   3.3. Outcome and parameters
4. SAML GSS-API Mechanism Specification
1. Introduction

Security Assertion Markup Language (SAML) 2.0 [OASIS.saml-core-2.0-os] is a modular specification that provides various means for a user to be identified to a relying party (RP) through the exchange of (typically signed) assertions issued by an identity provider (IdP). It includes a number of protocols, protocol bindings [OASIS.saml-bindings-2.0-os], and interoperability profiles [OASIS.saml-profiles-2.0-os] designed for different use cases.

Simple Authentication and Security Layer (SASL) [RFC4422] is a generalized mechanism for identifying and authenticating a user and for optionally negotiating a security layer for subsequent protocol interactions. SASL is used by application protocols like [IMAP](RFC3501), [POP](RFC1939), and [XMPP](RFC3920). The effect is to make modular authentication, so that newer authentication mechanisms can be added as needed. This memo specifies just such a mechanism.

The Generic Security Service Application Program Interface (GSS-API) [RFC2743] provides a framework for applications to support multiple authentication mechanisms through a unified programming interface. This document defines a pure SASL mechanism for SAML, but it conforms to the new bridge between SASL and the GSS-API called GS2 [RFC5801]. This means that this document defines both a SASL mechanism and a GSS-API mechanism. We want to point out that the GSS-API interface is optional for SASL implementers, and the GSS-API considerations can be avoided in environments that uses SASL directly without GSS-API.

As currently envisioned, this mechanism is to allow the interworking between SASL and SAML in order to assert identity and other attributes to relying parties. As such, while servers (as relying parties) will advertise SASL mechanisms (including SAML), clients will select the SAML SASL mechanism as their SASL mechanism of choice.

The SAML mechanism described in this memo aims to re-use the Web Browser SSO profile defined in section 3.1 of [OASIS.saml-profiles-2.0-os] to a maximum extent and therefore does not establish a separate authentication, integrity and confidentiality mechanism. The mechanisms assumes a security layer, such as Transport Layer Security (TLS [RFC5246]), will continued to be used. This specification is appropriate for use when a browser is available.

Figure 1 describes the interworking between SAML and SASL: this document requires enhancements to the Relying Party and to the Client (as the two SASL communication end points) but no changes to the SAML Identity Provider are necessary. To accomplish this goal some indirect messaging is tunneled within SASL, and some use of external methods is made.
1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The reader is assumed to be familiar with the terms used in the SAML 2.0 specification.

1.2. Applicability

Applicability Because this mechanism transports information that should not be controlled by an attacker, the SAML mechanism MUST only be used over channels protected by TLS, and the client MUST successfully validate the server certificate, or similar integrity protected and authenticated channels. [RFC5280][RFC6125]

2. Applicability for non-HTTP Use Cases

While SAML itself is merely a markup language, its common use case these days is with HTTP [RFC2616] and HTML [W3C.REC-html401-19991224]. What follows is a typical flow:

1. The browser requests a resource of a Relying Party (RP) (via an HTTP request).
2. The RP sends an HTTP redirect as described in Section 10.3 of [RFC2616] to the browser to the Identity Provider (IdP) or an IdP discovery service with an authentication request that contains the name of resource being requested, some sort of a cookie and a return URL [RFC1738].
3. The user authenticates to the IdP and perhaps authorizes the authentication to the service provider.
4. In its authentication response, the IdP redirects (via an HTTP redirect) the browser back to the RP with an authentication assertion (stating that the IdP vouches that the subject has successfully authenticated), optionally along with some additional attributes.
5. RP now has sufficient identity information to approve access to the resource or not, and acts accordingly. The authentication is concluded.
When considering this flow in the context of SASL, we note that while the RP and the client both must change their code to implement this SASL mechanism, the IdP must remain untouched. The RP already has some sort of session (probably a TCP connection) established with the client. However, it may be necessary to redirect a SASL client to another application or handler. This will be discussed below. The steps are shown from below:

1. The Relying Party or SASL server advertises support for the SASL SAML20 mechanism to the client
2. The client initiates a SASL authentication with SAML20 and sends a domain
3. The Relying Party transmits an authentication request encoded using a Universal Resource Identifier (URI) as described in RFC 3986 [RFC3986] and an HTTP redirect to the IdP corresponding to the domain
4. The SASL client now sends an empty response, as authentication continues via the normal SAML flow.
5. At this point the SASL client MUST construct a URL containing the content received in the previous message from the RP. This URL is transmitted to the IdP either by the SASL client application or an appropriate handler, such as a browser.
6. Next the client authenticates to the IdP. The manner in which the end user is authenticated to the IdP and any policies surrounding such authentication is out of scope for SAML and hence for this draft. This step happens out of band from SASL.
7. The IdP will convey information about the success or failure of the authentication back to the RP in the form of an Authentication Statement or failure, using an indirect response via the client browser or the handler. This step happens out of band from SASL.
8. The SASL Server sends an appropriate SASL response to the client, along with an optional list of attributes

Please note: What is described here is the case in which the client has not previously authenticated. It is possible that the client already holds a valid SAML authentication token so that the user does not need to be involved in the process anymore, but that would still be external to SASL. This is classic Web Single Sign-On, in which the Web Browser client presents the authentication token (cookie) to the RP without renewed user authentication at the IdP.

With all of this in mind, the flow appears as follows:

<table>
<thead>
<tr>
<th>SASL Serv.</th>
<th>Client</th>
<th>IdP</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;-----(1)-----</td>
<td></td>
<td>Advertisement</td>
</tr>
<tr>
<td>&lt;-----(2)-----&lt;</td>
<td></td>
<td>Initiation</td>
</tr>
<tr>
<td>&gt;-----(3)-----</td>
<td></td>
<td>Authentication Request</td>
</tr>
<tr>
<td>&lt;-----(4)-----&lt;</td>
<td></td>
<td>Empty Response</td>
</tr>
<tr>
<td></td>
<td>&lt; - - - - - -&gt;</td>
<td>Client&lt;&gt;IDP Authentication</td>
</tr>
<tr>
<td></td>
<td>&lt;- - - - - - - - - - - - - - - -</td>
<td>Authentication Statement</td>
</tr>
<tr>
<td>&gt;-----(6)-----</td>
<td></td>
<td>SASL completion with status</td>
</tr>
</tbody>
</table>

----- = SASL
- - - = HTTP or HTTPs (external to SASL)
3. SAML SASL Mechanism Specification

This section specifies the details of the SAML SASL mechanism. Recall section 5 of [RFC4422] for what needs to be described here.

The name of this mechanism "SAML20". The mechanism is capable of transferring an authorization identity (via "gs2-header"). The mechanism does not offer a security layer.

The mechanism is client-first. The first mechanism message from the client to the server is the "initial-response" described below. As described in [RFC4422], if the application protocol does not support sending a client-response together with the authentication request, the server will send an empty server-challenge to let the client begin.

The second mechanism message is from the server to the client, the "authentication-request" described below.

The third mechanism message is from client to the server, and is the fixed message consisting of "=".

The fourth mechanism message is from the server to the client, indicating the SASL mechanism outcome described below.

3.1. Initial Response

A client initiates a "SAML20" authentication with SASL by sending the GS2 header followed by the authentication identifier. The GS2 header carries the optional authorization identity.

\[
\text{initial-response} = \text{gs2-header Idp-Identifier} \\
\text{Idp-Identifier} = \text{domain} ; \text{domain name with corresponding IdP}
\]

The "gs2-header" is specified in [RFC5801], and it is used as follows. The "gs2-nonstd-flag" MUST NOT be present. Regarding the channel binding "gs2-cb-flag" field, see Section 5. The "gs2-authzid" carries the optional authorization identity. Domain name is specified in [RFC1035].

3.2. Authentication Request

The SASL Server transmits a redirect URI to the IdP that corresponds to the domain the user provided, with a SAML authentication request as one of the parameters. Note: The SASL server may have a static mapping of domain to corresponding IdP or alternatively a DNS-lookup mechanism could be envisioned, but that is out-of-scope for this document.

\[
\text{authentication-request} = \text{URI}
\]

URI is specified in [RFC3986] and is encoded according to Section 3.4 (HTTP Redirect) of the SAML bindings 2.0 specification [OASIS.saml-bindings-2.0-os]. The SAML authentication request is encoded according to Section 3.4 (Authentication Request) of the SAML core 2.0 specification [OASIS.saml-core-2.0-os].

The client now sends the authentication request via an HTTP GET to the IdP, as if redirected.
to do so from an HTTP server and in accordance with the Web Browser SSO profile, described in section 3.1 of [OASIS.saml-profiles-2.0-os]

The client MUST handle both user authentication to the IdP and confirmation or rejection of the authentication of the RP.

After all authentication has been completed by the IdP, and after the response has been sent to the client, the client will relay the response to the Relying Party via HTTP(S), as specified in the authentication request ("AssertionConsumerServiceURL").

Please note: this means that the SASL server needs to implement a SAML Relying Party. Also, the RP needs to correlate the TCP session from the SASL client with the SAML authentication.

3.3. Outcome and parameters

The Relying Party now validates the response it received from the client via HTTP or HTTPS, as specified in the SAML specification

The response by the Relying Party constitutes a SASL mechanism outcome, and SHALL be used to set state in the server accordingly, and it shall be used by the server to report that state to the SASL client as described in [RFC4422] Section 3.6.

4. SAML GSS-API Mechanism Specification

This section and its sub-sections and all normative references of it not referenced elsewhere in this document are INFORMATIONAL for SASL implementors, but they are NORMATIVE for GSS-API implementors.

The SAML SASL mechanism is actually also a GSS-API mechanism. The messages are the same, but

a) the GS2 header on the client's first message and channel binding data is excluded when SAML is used as a GSS-API mechanism, and

b) the RFC2743 section 3.1 initial context token header is prefixed to the client's first authentication message (context token).

The GSS-API mechanism OID for SAML is 1.3.6.1.4.1.11591.4.8.

SAML20 security contexts always have the mutual_state flag (GSS_C_MUTUAL_FLAG) set to TRUE. SAML does not support credential delegation, therefore SAML security contexts always have the deleg_state flag (GSS_C_DELEG_FLAG) set to FALSE.

The mutual authentication property of this mechanism relies on successfully comparing the TLS server identity with the negotiated target name. Since the TLS channel is managed by the application outside of the GSS-API mechanism, the mechanism itself is unable to confirm the name while the application is able to perform this comparison for the mechanism. For this reason, applications MUST match the TLS server identity with the target name, as discussed in [RFC6125].

The SAML mechanism does not support per-message tokens or GSS_Pseudo_random.

4.1. GSS-API Principal Name Types for SAML

SAML supports standard generic name syntaxes for acceptors such as GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4.1). SAML supports only a single name type for initiators: GSS_C_NT_USER_NAME. GSS_C_NT_USER_NAME is the default name type for SAML. The query, display, and exported name syntaxes for SAML principal names are all the same. There are no SAML-specific name syntaxes -- applications should use generic GSS-API name types such as GSS_C_NT_USER_NAME and
should use generic GSS-API name types such as GSS_C_NT_USER_NAME and GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4). The exported name token does, of course, conform to [RFC2743], Section 3.2.

5. Channel Binding

The "gs2-cb-flag" MUST use "n" because channel binding data cannot be integrity protected by the SAML negotiation.

Note: In theory channel binding data could be inserted in the SAML flow by the client and verified by the server, but that is currently not supported in SAML.

6. Examples

6.1. XMPP

Suppose the user has an identity at the SAML IdP saml.example.org and a Jabber Identifier (JID) "somenode@example.com", and wishes to authenticate his XMPP connection to xmpp.example.com. The authentication on the wire would then look something like the following:

Step 1: Client initiates stream to server:

```xml
<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' to='example.com' version='1.0'>
```

Step 2: Server responds with a stream tag sent to client:

```xml
<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' id='some_id' from='example.com' version='1.0'>
```

Step 3: Server informs client of available authentication mechanisms:

```xml
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <mechanism>SAML20</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Client selects an authentication mechanism and provides the initial client response containing the **BASE64** [RFC4648] encoded gs2-header and domain:

```xml
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='SAML20'>
The decoded challenge is:

https://saml.example.org/SAML/Browser?SAMLRequest=PHNhbWxwOk5hbWVJRFBvbGljeSB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIg0KICAgIElEPSJfYmVjNDI0ZmE1MTAzNDI4OTMwZmYxZTMxMTY4MzI3Zjc5NDc0OTg0IiBWZXJzaW9uPSIyLjAiDQogICAgSXNzdWVJbnN0YW50PSIyMDA3LTEyLTEwVDExOjM5OjM0WiIgRm9yY2VBdXRpbGRJb0iZmFsc2UiDQogICAgSXNQYXNzaXZlPSJmYWxzZSINCiAgICBQcm90b2NvbEJpbmRpbmc9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDpuYW1laWQtZm9ybWF0OnA5YTMwZmYxMTYmxMzI3zj5Ndc60TgqObIbwZXJaazWgpSyljaiDAdQigAgSXNzdWVJb0uN0WS5P0SiYMDA3LwEjTEwVDE0xMjQ5oMjQwI09mRnyY2VBdX9Rojb0iZmFsc2UiDAdQigAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgAgA
Where the decoded SAMLRequest looks like:

```xml
<AuthnRequest xmlns="urn:oasis:names:tc:SAML:2.0:protocol"
ID="_bec424fa5103428909a30ff1e31168327f79474984" Version="2.0"
IssueInstant="2007-12-10T11:39:34Z" ForceAuthn="false"
IsPassive="false"
ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
AssertionConsumerServiceURL="https://xmpp.example.com/SAML/AssertionConsumerService">
  <Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    https://xmpp.example.com
  </Issuer>
  <NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
    SPNameQualifier="xmpp.example.com" AllowCreate="true" />
  <RequestedAuthnContext Comparison="exact"/>
</AuthnRequest>
```

Note: the server can use the request ID (_bec424fa5103428909a30ff1e31168327f79474984) to correlate the SASL session with the SAML authentication.

Step 5 (alt): Server returns error to client:

```xml
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Step 6: Client sends a BASE64 encoded empty response to the challenge:

```xml
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'=
</response>
```

[The client now sends the URL to a browser for processing. The browser engages in a normal SAML authentication flow (external to SASL), like redirection to the Identity Provider]
SAML authentication flow (external to SASL), like redirection to the Identity Provider (https://saml.example.org), the user logs into https://saml.example.org, and agrees to authenticate to xmpp.example.com. A redirect is passed back to the client browser who sends the AuthN response to the server, containing the subject-identifier as an attribute. If the AuthN response doesn't contain the JID, the server maps the subject-identifier received from the IdP to a JID.

Step 7: Server informs client of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 7 (alt): Server informs client of failed authentication:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</failure>
</stream:stream>
```

Step 8: Client initiates a new stream to server:

```
<stream:stream xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com' version='1.0'>
```

Step 9: Server responds by sending a stream header to client along with any additional features (or an empty features element):

```
<stream:stream xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_345' from='example.com' version='1.0'>
    <stream:features>
      <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
      <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
    </stream:features>
</stream:stream>
```

Step 10: Client binds a resource:

```
<iq type='set' id='bind_1'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
</iq>
```

Step 11: Server informs client of successful resource binding:

```
<iq type='result' id='bind_1'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
```

6.2. IMAP

The following describes an IMAP exchange. Lines beginning with 'S:' indicate data sent by the server, and lines starting with 'C:' indicate data sent by the client. Long lines are wrapped for readability.

```
S:* OK IMAP4rev1
C:. CAPABILITY
S:* CAPABILITY IMAP4rev1 STARTTLS
S:. OK CAPABILITY Completed
C:. STARTTLS
S:. OK Begin TLS negotiation now
C:. CAPABILITY
S:* CAPABILITY IMAP4rev1 AUTH=SAML20
S:. OK CAPABILITY Completed
C:. AUTHENTICATE SAML20
S: +
C: biwsZXhhbXBsZS5vcmtcI0N5
S: + Y2Fmb24uYmVkb2JqMzM5
C: biwsZXhhbXBsZS5vcmtcI0N5
S: + Y2Fmb24uYmVkb2JqMzM5
C:. SUCCESS (tls protection)
```
7. Security Considerations

This section will address only security considerations associated with the use of SAML with SASL applications. For considerations relating to SAML in general, the reader is referred to the SAML specification and to other literature. Similarly, for general SASL Security Considerations, the reader is referred to that specification.

7.1. Man in the middle and Tunneling Attacks

This mechanism is vulnerable to man in the middle and tunneling attacks unless a client always verify the server identity before proceeding with authentication (see [RFC6125]). Typically TLS is used to provide a secure channel with server authentication.

7.2. Binding SAML subject identifiers to Authorization Identities

As specified in [RFC4422], the server is responsible for binding credentials to a specific authorization identity. It is therefore necessary that only specific trusted IdPs be allowed. This is typical part of SAML trust establishment between RP’s and IdP.

7.3. User Privacy

The IdP is aware of each RP that a user logs into. There is nothing in the protocol to hide this information from the IdP. It is not a requirement to track the visits, but there is nothing that prohibits the collection of information. SASL servers should be aware that SAML IdPs will track - to some extent - user access to their services.

7.4. Collusion between RPs

It is possible for RPs to link data that they have collected on you. By using the same identifier to log into every RP, collusion between RPs is possible. In SAML, targeted identity was introduced. Targeted identity allows the IdP to transform the identifier the user typed in to an opaque identifier. This way the RP would never see the actual user identifier, but a randomly generated identifier. This is an option the user has to understand and decide to use if the IdP is supporting it.

8. IANA Considerations

The IANA is requested to register the following SASL profile:

SASL mechanism profile: SAML20
Security Considerations: See this document
Published Specification: See this document
For further information: Contact the authors of this document.
Owner/Change controller: the IETF
Note: None
9. References

9.1. Normative References


9.2. Informative References


Appendix A. Acknowledgments

The authors would like to thank Scott Cantor, Joe Hildebrand, Josh Howlett, Leif Johansson, Diego Lopez, Hank Mauldin, RL ‘Bob’ Morgan, Stefan Plug and Hannes Tschofenig for their review and contributions.

Appendix B. Changes

This section to be removed prior to publication.

- 03 Number of cosmetic changes, fixes per comments Alexey Melnikov
- 02 Changed IdP URI to domain per Joe Hildebrand, fixed some typos
00 WG -00 draft. Updates GSS-API section, some fixes per Scott Cantor
01 Added authorization identity, added GSS-API specifics, added client supplied IdP
00 Initial Revision.

Authors' Addresses

Klaas Wierenga
Cisco Systems, Inc.
Haarlerbergweg 13-19
Amsterdam, Noord-Holland 1101 CH
Netherlands
Phone: +31 20 357 1752
Email: klaas@cisco.com

Eliot Lear
Cisco Systems GmbH
Richtistrasse 7
Wallisellen, ZH CH-8304
Switzerland
Phone: +41 44 878 9200
Email: lear@cisco.com

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE
Email: simon@josefsson.org
URI: http://josefsson.org/