Network Working Group	K. Wierenga	TOC
Internet-Draft	Cisco Systems, Inc.	
Intended status: Standards Track	E. Lear	
Expires: July 8, 2012	Cisco Systems GmbH	
	S. Josefsson	
	SJD AB	
	January 5, 2012	

A SASL and GSS-API Mechanism for SAML
draft-ietf-kitten-sasl-saml-07.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

Copyright Notice

Table of Contents

TOC

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** [RFC6120]. 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. The GSS-API interface is OPTIONAL for SASL implementers, and the GSS-API considerations can be avoided in environments that use 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 **the SAML profiles 2.0 specification** [OASIS.saml-profiles-2.0-os] to the maximum extent and therefore does not establish a separate authentication, integrity and confidentiality mechanism. The mechanism assumes a security layer, such as Transport Layer Security (**TLS** [RFC5246]), will continue 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 (the SASL server) 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.

```
                                     +------------+
                                     |            |
                                    >|  Relying   |
                                   / |  Party     |
                                 //  |            |
                               //    +------------+
                   SAML/     //            ^
                   HTTPs   //           +--|--+
                         //             | S|  |
                        /             S | A|  |
                      //              A | M|  |
                    //                S | L|  |
                  //                  L |  |  |
                //                      |  |  |
              </                        +--|--+
       +------------+                      v
       |            |                 +----------+
       |  SAML      |     HTTPs       |          |
       |  Identity  |<--------------->|  Client  |
       |  Provider  |                 |          |
       +------------+                 +----------+
```

Figure 1: Interworking Architecture

1.1. Terminology

TOC

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

TOC

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]**

Note: An Intranet does not constitute such an integrity protected and authenticated channel!

2. Authentication flow

TOC

While SAML itself is merely a markup language, its common use case these days is with **HTTP** [RFC2616] or **HTTPs** [RFC2818] 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 Relying Party redirects the browser via an HTTP redirect (as described in Section 10.3 of **[RFC2616]**) to the Identity Provider (IdP) or an IdP discovery service with as parameters an authentication request that contains the name of resource being requested, a browser cookie and a return URL as specified in Section 3.1 of the **SAML profiles 2.0 specification** [OASIS.saml-profiles-2.0-os].
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. The Relying Party 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 Relying Party and the client both must change their code to implement this SASL mechanism, the IdP can remain untouched. The Relying Party 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 SASL server (Relying Party) advertises support for the SASL SAML20 mechanism to the client
2. The client initiates a SASL authentication with SAML20 and sends a domain name that allows the SASL server to determine the appropriate IdP
3. The SASL server 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 SASL server. 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 the SASL server (Relying Party) in the form of an Authentication Statement or failure, using a indirect response via the client browser or the handler (and with an external browser client control should be passed back to the SASL client). 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:

```
          SASL Serv.       Client          IdP
             |>-----(1)----->|              | Advertisement
             |               |              |
             |<-----(2)-----<|              | Initiation
             |               |              |
             |>-----(3)----->|              | Authentication Request
             |               |              |
             |<-----(4)-----<|              | Empty Response
             |               |              |
             |               |< - - - - - ->| Client<>IDP
             |               |              | Authentication
             |               |              |
             |<- - - - - - - - - - - - - - -| Authentication Statement
             |               |              |
             |>-----(5)----->|              | SASL completion with
             |               |              | status
             |               |              |
```

```
    ----- = SASL
    - - - = HTTP or HTTPs (external to SASL)
```

Figure 2: Authentication flow

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 (message 2 in **Figure 2**). The GS2 header carries the optional authorization identity.

```
initial-response = gs2-header Idp-Identifier
IdP-Identifier = domain ; 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 to the SASL client a URI that (re)directs to the IdP (corresponding to the domain the user provided), with a SAML authentication request as one of the parameters (message 3 in **Figure 2**).

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.

Note: While the SASL client MAY sanity check the URI it received, ultimately it is the SAML IdP that will be validated by the SAML client which is out-of-scope for this document.

```
authentication-request = 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 (sent over a server-authenticated TLS channel) to the IdP, as if redirected to do so from an HTTP server and in accordance with the Web Browser SSO profile, as described in section 3.1 of **SAML profiles 2.0 specification** [OASIS.saml-profiles-2.0-os]

The client handles both user authentication to the IdP and confirmation or rejection of the authentiation of the RP (out-of-scope for this document).

After all authentication has been completed by the IdP, the IdP will send a redirect message to the client in the form of a URI corresponding to the Relying Party as specified in the authentication request ("AssertionConsumerServiceURL") and with the SAML response as one of the parameters.

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

TOC

3.3. Outcome and parameters

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

The response by the SASL server 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 (message 5 in **Figure 2**).

TOC

4. SAML GSS-API Mechanism Specification

This section and its sub-sections and appropriate references of it not referenced elsewhere in this document are not required for SASL implementors, but this section MUST be observed to implement the GSS- API mechanism discussed below.

The SAML SASL mechanism is actually also a GSS-API mechanism. The SAML user takes the role of the GSS-API Initiator and the SAML Relying Party takes the role of the GSS-API Acceptor. The SAML Identity Provider does not have a role in GSS-API, and is considered an internal matter for the SAML 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 OID-TBD (IANA to assign: see IANA considerations).

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

TOC

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 GSS_C_NT_HOSTBASED_SERVICE (see **[RFC2743]**, Section 4). The exported name token does, of course, conform to **[RFC2743]**, Section 3.2.

TOC

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.

TOC

6. Examples

TOC

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:

```
<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:

```
<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:

```
<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:


```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='SAML20'>
biwsZXhhbXBsZS5vcmc</auth>
```


The decoded string is: n,,example.org

Step 5: Server sends a BASE64 encoded challenge to client in the form of an HTTP Redirect to the SAML IdP corresponding to example.org (https://saml.example.org) with the SAML Authentication Request as specified in the redirection url:


```
aHR0cHM6Ly9zYW1sLmV4YW1wbGUub3JnL1NBTUwvQnJvd3Nlcj9TQU1MUmVx
dWVzdD1QSE5oYld4d09rRjFkR2h1VW1WeGRXVnpkQ0I0Yld4dWN6cHpZVzFz
Y0QwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUVXc2TWk0d09uQnli
M1J2WTI5c0lnMEtJQ0FnSUVsRVBTSmZZbVZqTkRJMFptRTFNVEF6TkRJNE9U
QTVZVE13Wm1ZeFpUTXhNVFk0TXpJM1pqYzVORGMwT1RnMElpQldaWEp6YVc5
dVBTSXlMakFpRFFvZ0lDQWdTWE56ZFdWSmJuTjBZVzUwUFNJeU1EQTNMVEV5
TFRFd1ZERXhPak01T2pNMFdpSWdSbTl5WTJWQmRYUm9iajBpWm1Gc2MyVWlE
UW9nSUNBZ1NYTlFZWE56YVhabFBTSm1ZV3h6WlNJTkNpQWdJQ0JRY205MGIy
TnZiRUpwYm1ScGJtYzlJblZ5YmpwdllYTnBjenB1WVcxbGN6cDBZenBUUVUx
TU9qSXVNRHBpYVc1a2FXNW5jenBJVkZSUUxWQlBVMVFpRFFvZ0lDQWdRWE56
WlhKMGFXOXVRMjl1YzNWdFpYSlRaWEoyYVdObFZWSk1QUTBLSUNBZ0lDQWdJ
Q0FpYUhSMGNITTZMeTk0YlhCd0xtVjRZVzF3YkdVdVkyOXRMMU5CVFV3dlFY
TnpaWEowYVc5dVEyOXVjM1Z0WlhKVFpYSjJhV05sSWo0TkNpQThjMkZ0YkRw
SmMzTjFaWElnZUcxc2JuTTZjMkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6
T25Sak9sTkJUVXc2TWk0d09tRnpjMlZ5ZEdsdmJpSStEUW9nSUNBZ0lHaDBk
SEJ6T2k4dmVHMXdjQzVsZUdGdGNHeGxMbU52YlEwS0lEd3ZjMkZ0YkRwSmMz
TjFaWEkrRFFvZ1BITmhiV3h3T2s1aGJXVkpSRkJ2YkdsamVTQjRiV3h1Y3pw
ellXMXNjRDBpZFhKdU9tOWhjMmx6T201aGJXVnpQblJqT2xOQlRVdzZNaTR3
T25CeWIzUnZZMjlzSWcwS0lDQWdJQ0JHYjNKdFlYUTlJblZ5YmpwdllYTnBj
enB1WVcxbGN6cDBZenBUUVUxTU9qSXVNRHB1WVcxbGFXUXRabTl5YldGME9u
Qmxjbk5wYzNSbGJuUWlEUW9nSUNBZ0lGTlFUbUZ0WlZGMVlXeHBabWxsY2ow
aWVHMXdjQzVsZUdGdGNHeGxMbU52YlNJZ1FXeHNiM2REY21WaGRHVTlJblJ5
ZFdVaUlDOCtEUW9nUEhOaGJXeHdPbEpsY1hWbGMzUmxaRUYxZEdodVEyOXVk
R1Y0ZEEwS0lDQWdJQ0I0Yld4dWN6cHpZVzFzY0QwaWRYSnVPbTloYzJsek9t
NWhiV1Z6T25Sak9sTkJUVXc2TWk0d09uQnliM1J2WTI5c0lpQU5DaUFnSUNB
Z0lDQWdRMjl0Y0dGeWFFYTnZiajBpWlhoaFkzUWlQZ0BLSUNBOGMyRnRiRHBC
ZFhSb2JrTnZibUJsZUhSRGJHRnpjMUpsWmcwS0lDQWdJQ0FnZUcxc2JuTTZj
MkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUVXc2TWk0d09t
RnpjMlZ5ZEdsdmJpSStEUW9nb0NBZ0lDQjFjbTQ2YjJGemFYTTZibUZ0WlhN
NmRHTTZVMEZOVERveUxqQTZZV002WTJ4aGMzTmxjenBRWVhOemQyOXlaRkJ5
YjNSbFkzUmxaRlJ5WVc1emNHOXlkQTBLSUNBOEwzTmhiV3c2UVhWMGFHNURi
MjUwWlhoMFEyeGhjM05TWldZK0RRb2dQQzl6WVcxc2NERNcFNaWEYxWlhOMFpX
UkJkWFJvYmtOdmJuUmxlSFErRFFvZ1BDOXpZWE1zY0RwQmRYUm9ibEpsY1hW
bGMzUSs=
```

The decoded challenge is:


```
https://saml.example.org/SAML/Browser?SAMLRequest=PHNhbWxwOk
F1dGhuUmVxdWVzdCB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5hbWVzOnRjOl
NBTUw6Mi4wOnByb3RvY29sIg0KICAgIElEPSJfYmVjNDI0ZmE1MTAzNDI4OT
A5YTMwZmYxZTMxMTY4MzI3Zjc5NDc0OTg0IiBWZXJzaW9uPSIyLjAiDQogIC
AgSXNzdWVJbnN0YW50PSIyMDA3LTEyLTEwVDExOjM5OjM0WiIgRm9yY2VBdX
Robj0iZmFsc2UiDQogICAgSXNQYXNzaXZlPSJmYWxzZSINCiAgICBQcm90b2
NvbEJpbmRpbmc9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDpiaW5kaW
5nczpIVFRQLVBPU1QiDQogICAgQXNzZXJ0aW9uQ29uc3VtZXJTZXJ2aWNlVV
JMPQ0KICAgICAgICAiaHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tL1NBTUwvQX
NzZXJ0aW9uQ29uc3VtZXJTZXJ2aWNlIj4NCiA8c2FtbDpJc3N1ZXIgeG1sbn
M6c2FtbD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOmFzc2VydGlvbi
I+DQogICAgIGh0dHBzOi8veG1wcC5leGFtcGxlLmNvbQ0KIDwvc2FtbDpJc3
N1ZXI+DQogPHNhbWxwOk5hbWVJRFBvbGljeSB4bWxuczpzYW1scD0idXJuOm
9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIg0KICAgICBGb3JtYX
Q9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDpuYW1laWQtZm9ybWF0On
BlcnNpc3RlbnQiDQogICAgIFNQTmFtZVF1YWxpZmllcj0ieG1wcC5leGFtcG
xlLmNvbSIgQWxsb3dDcmVhdGU9InRydWUiIC8+DQogPHNhbWxwOlJlcXVlc3
RlZEF1dGhuQ29udGV4dA0KICAgICB4bWxuczpzYW1scD0idXJuOm9hc2lzOm
5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIiANCiAgICAgICAgQ29tcGFyaX
Nvbj0iZXhhY3QiPg0KICA8c2FtbDpBdXRobkNvbnRleHRDbGFzc1JlZg0KIC
AgICAgeG1sbnM6c2FtbD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOm
Fzc2VydGlvbiI+DQogICAgICAgICAgIHVybjpvYXNpczpuYW1lczp0YzpTQU
1MOjIuMDphYzpjbGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0DQ
ogIDwvc2FtbDpBdXRobkNvbnRleHRDbGFzc1JlZj4NCiA8L3NhbWxwOlJlcX
Vlc3RlZEF1dGhuQ29udGV4dD4gDQo8L3NhbWxwOkF1dGhuUmVxdWVzdD4=
```


Where the decoded SAMLRequest looks like:


```
<samlp:AuthnRequest xmlns:samlp="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">
 <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
     https://xmpp.example.com
 </saml:Issuer>
 <samlp:NameIDPolicy xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
     Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
     SPNameQualifier="xmpp.example.com" AllowCreate="true" />
 <samlp:RequestedAuthnContext
     xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
        Comparison="exact">
  <saml:AuthnContextClassRef
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
      urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
  </saml:AuthnContextClassRef>
 </samlp:RequestedAuthnContext>
</samlp: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:

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

Step 6: Client sends the empty response to the challenge encoded as a single =:

```
<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 (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>
```

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'>
    <jid>somenode@example.com/someresource</jid>
  </bind>
</iq>
```

Please note: line breaks were added to the base64 for clarity.

6.2. IMAP

TOC

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: biwsZXhhbXBsZS5vcmc
S: + aHR0cHM6Ly9zYW1sLmV4YW1wbGUub3JnL1NBTUwvQnJvd3Nlcj9TQU1MUmVx
dWVzdD1QSE5oYld4d09rRjFkR2h1VW1WeGRXVnpkQ0I0Yld4dWN6cHpZVzFz
Y0QwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUVXc2TWk0d09uQnli
M1J2WTI5c0lnMEtJQ0FnSUVsRVBTSmZZbVZqTkRJMFptRTFNVEF6TkRJNE9U
QTVZVE13Wm1ZeFpUTXhNVFk0TXpJM1pqYzVORGMwT1RnMElpQldaWEp6YVc5
dVBTSXlMakFpRFFvZ0lDQWdTWE56ZFdWSmJuTjBZVzUwUFNJeU1EQTNMVEV5
TFRFd1ZERXhPak01T2pNMFdpSWdSbTl5WTJWQmRYUm9iajBpWm1Gc2MyVWlE
UW9nSUNBZ1NYTlFZWE56YVhabFBTSm1ZV3h6WlNJTkNpQWdJQ0JRY205MGIy
TnZiRUpwYm1ScGJtYzlJblZ5YmpwdllYTnBjenB1WVcxbGN6cDBZenBUUVUx
TU9qSXVNRHBpYVc1a2FXNW5jenBJVkZSUUxWQlBVMVFpRFFvZ0lDQWdRWE56
WlhKMGFXOXVRMjl1YzNWdFpYSlRaWEoyYVdObFZWSk1QUTBLSUNBZ0lDQWdJ
Q0FpYUhSMGNITTZMeTk0YlhCd0xtVjRZVzF3YkdVdVkyOXRMMU5CVFV3dlFY
TnpaWEowYVc5dVEyOXVjM1Z0WlhKVFpYSjJhV05sSWo0TkNpQThjMkZ0YkRw
SmMzTjFaWElnZUcxc2JuTTZjMkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6
T25Sak9sTkJUVXc2TWk0d09tRnpjMlZ5ZEdsdmJpSStUW9nSUNBZ0lHaDBk
SEJ6T2k4dmVHMXdjQzVsZUdGdGNHeGxMbU52YlM0OGMzTjFaWEkrUkUwZ0tkMz
TjFaWEkrRFFvZ1BITmhiV3h3T2s1aGJXVkpSRkJ2YkdsamVTQjRiV3h1Y3pw
ellXXMXNjRDBpZFhKdU9tOWhjMmx6T201aGJXVnpPblJqT2xOQlRVdzZNaTR3
T25CeWIzUnZZMjlzSWcwS0lDQWdJQ0JHYjNKdFlYUTlJblZ5YmpwdllYTnBj
```

```
enB1WVcxbGN6cDBZenBUUVUxTU9qSXVNRHB1WVcxbGFXUXRabTl5YldGME9u
Qmxjbk5wYzNSbGJuUWlEUW9nSUNBZ0lGTlFUbUZ0WlZGMVlXeHBabWxsY2ow
aWVHMXdjQzVsZUdGdGNHeGxMbU52YlNJZ1FXeHNiM2REY21WaGRHVTlJblJ5
ZFdVaUlDOCtEUW9nUEhOaGJXeHdPbEpsY1hWbGMzUmxaRUYxZEdodVEyOXVk
R1Y0ZEEwS0lDQWdJQ0I0Yld4dWN6cHpZVzFzY0QwaWRYSnVPbTloYzJsek9t
NWhiV1Z6T25Sak9sTkJUVXc2TWk0d09uQnliM1J2WTI5c0lpQU5DaUFnSUNB
Z0lDQWdRMjl0Y0dGeWFYTnZiajBpWlhoaFkzUWlQZzBLSUNBOGMyRnRiRHBC
ZFhSb2JrTnZiblJsZUhSRGJHRnpjMUpsWmcwS0lDQWdJQ0FnZUcxc2JuTTZj
MkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUVXc2TWk0d09t
RnpjMlZ5ZEdsdmJpSStEUW9nb0NBZ0lDQjFjbTQ2YjJGemFYTTZibUZ0WlhN
NmRHTTZVMEZOVERveUxqQTZZV002WTJ4aGMzTmxjenBRWVhOemQyOXlaRkJ5
YjNSbFkzUmxaRlJ5WVc1emNHOXlkQTBLSUNBOEwzTmhiV3c2UVhWMGFHNURi
MjUwWlhoMFEyeGhjM05TWldZK0RRb2dQQzl6WVcxc2NEcFNaWEYxWlhOMFpX
UkJkWFJvYmtOdmJuUmxlSFErSUEwS1BDOXpZVzFzY0RwQmRYUm9ibEpsY1hW
bGMzUSs=
C:
S: . OK 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 Relying Parties and IdP.

7.3. User Privacy

The IdP is aware of each Relying Party 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 Relying Parties to link data that they have collected on you. By using the same identifier to log into every Relying Party, collusion between Relying Parties 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 Relying Party 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

TOC

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

The IANA is further requested to assign an OID for this GSS mechanism in the SMI numbers registry, with the prefix of iso.org.dod.internet.security.mechanisms (1.3.6.1.5.5) and to reference this specification in the registry.

9. References

TOC

9.1. Normative References

TOC

[OASIS.saml-bindings-2.0-os]	Cantor, S., Hirsch, F., Kemp, J., Philpott, R., and E. Maler, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard saml-bindings-2.0-os, March 2005.
[OASIS.saml-core-2.0-os]	Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard saml-core-2.0-os, March 2005.
[OASIS.saml-profiles-2.0-os]	Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., and E. Maler, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard OASIS.saml-profiles-2.0-os, March 2005.
[RFC1035]	Mockapetris, P., "Domain names - implementation and specification," STD 13, RFC 1035, November 1987 (TXT).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).
[RFC2743]	Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1," RFC 2743, January 2000 (TXT).
[RFC2818]	Rescorla, E., "HTTP Over TLS," RFC 2818, May 2000 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC4422]	Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)," RFC 4422, June 2006 (TXT).
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008 (TXT).
[RFC5280]	Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008 (TXT).
[RFC5801]	Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family," RFC 5801, July 2010 (TXT).
[RFC6125]	Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)," RFC 6125, March 2011 (TXT).
[W3C.REC-html401-19991224]	Raggett, D., Jacobs, I., and A. Hors, "HTML 4.01 Specification," World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (HTML).

9.2. Informative References

TOC

[RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3," STD 53, RFC 1939, May 1996 (TXT).

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1," RFC 3501, March 2003 (TXT).
[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," RFC 4648, October 2006 (TXT).
[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120, March 2011 (TXT).

TOC

Appendix A. Acknowledgments

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

TOC

Appendix B. Changes

This section to be removed prior to publication.

- 07 Fixed text per comments Alexey Melnikov
- 06 Fixed text per AD comments
- 05 Fixed references per ID-nits
- 04 Added request for IANA assignment, few text clarifications
- 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.

TOC

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/