A SASL and GSS-API Mechanism for SAML
draft-ietf-kitten-sasl-saml-08.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 July 15, 2012.

Copyright Notice

Copyright (c) 2012 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. Authentication flow

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 [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.
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 [OASIS.saml-core-2.0-os].

1.2. 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 over similar integrity protected and authenticated channels. [RFC5280][RFC6125]

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

2. Authentication flow

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 Relying Party.

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. The steps are as follows:

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 in Figure 2:
3. SAML SASL Mechanism Specification

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

The name of this mechanism is "SAML20". The mechanism is capable of transferring an authorization identity (via the "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". 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, containing the SAML "authentication-request".

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.

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) and is defined as follows:

\[
\text{initial-response} = \text{gs2-header Idp-Identifier}
\]

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

The "gs2-header" carries the optional authorization identity as specified in [RFC5801], and it is used as follows:

- The "gs2-nonstd-flag" MUST NOT be present.
- See Section 5 for the channel binding "gs2-cb-flag" field.
- 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 redirects the SAML client 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) in the following way:

```
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].

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.

The client then 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] (message 5 and 6 in Figure 2).

The client handles both user authentication to the IdP and confirmation or rejection of the authentication 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 (message 7 in Figure 2).

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 by comparing the ID of the SAML request with that in the response.

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 MUST be used to set state in the server accordingly, and it MUST be used by the server to report that state to the SASL client as described in [RFC4422] Section 3.6 (message 8 in Figure 2).

4. SAML GSS-API Mechanism Specification

This section, 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.

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.

5. Channel Binding

The "gs2-cb-flag" MUST be set to "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:

```
<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:

```
```
Step 3: Server informs client of available authentication mechanisms:

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

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

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'>
bIwsXhhxBsZ5svcmc</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:

```
aHR0cHM6Ly9zYW1sLmV4YW1wbGUub3JnL1NBUTuwVnJvd3Nlcj9TQU1MUNVx
```

This is the BASE64 encoded challenge.
The decoded challenge is:

https://saml.example.org/SAML/Browser?SAMLRequest=PHNhbwxwOkF1dGhuUmVxdWzdC48bWxuczpzYWlscD0idXJuOm9hc2l0Zm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY3MxeG1sbnM6c2FtbD0idXJuOm9hc2l0Zm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY3MxeG1sbnM6c2FtbD0idXJuOm9hc2l0Zm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY3Mx

Where the decoded SAMLRequest looks like:


</samlp:IsNameIDPolicy>


</samlp:IsNameIDPolicy>


</samlp:IsNameIDPolicy>

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

Step 5 (alternative): Server returns error to client if no SAML Authentication Request can be constructed:

```xml
<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 =:

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

[ The client now sends the URL to a browser instance 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:

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

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

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

Step 8: Client initiates a new stream to server:

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

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

```xml
<stream:stream xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com' version='1.0'>
</stream:stream>
```
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: biwsZXhhbXBsZS5vcncc
S: + aHR0cHM6Ly9zYWlsLmV4YW1wbGUub3JnL1NBUTUwVQj3Nlcj9TQU1UMUmVx
dVVzdDQ1QSE5oYld4d09rRjFKR2h1V1wGFXVnkpQ0I0Yld4dWN6cHpsVzzFz
YQ0waW1hYswvbkTloYzd3kek9TNWhiV1Z6TzSak9STkJUVxc2TWk0d099uqnnl
MlJ2WTI5c0lMMEtJQ0FvSU5RVBTsZmZbVzqTkrKMMeptRTFVVFhF6TkrJRJNE9U
QTFVZV13wmzFeFpUTXnNVFk0TXpJM1pqYzVOREGwT1RnME1pQIdaWp6VYv6C
vDBTSX1MaFkFpRFFvZ0lDQWdTWVE56ZFDwSmJuTjBZVzUwUFNJeUErTMNEVE5
TFRD1dZERhPak0t2pNMFdpSwdSbT15WTJQmRYUm9iajBpWmiG2cyMVWl6U
UW9nSUNBZ1NYTaFWS56YVhabFTBTSm1ZV3h6WlNJTkNpQWdJQ0JRY205MGlY
TnZlRupYm15cGJtyZljb1Z5Ympwdi1LTnBjenB1WxcbGNNcDBZenBUVVUx
TU9qSXXNRHbpYVc1a2FXNW5jnenBjk0QSUvQ1BVMVFpRFFvZ0lDQWdRWE56
WlhKMGFXOXVRMjl1YzNwDFpYSrlRaWeoyYVd0bFZwSk1QUTBLSUNBZ0lDQWdJ
The decoded SAMLRequest looks like:

```xml
<ds:KeyInfo>
</ds:KeyInfo>
```

Where the decoded SAMLRequest looks like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ds:KeyInfo>
</ds:KeyInfo>
```
7. Security Considerations

This section addresses 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 verifies 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 server implementers should be aware that SAML IdPs will be able to 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 the users. 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.
8. IANA Considerations

8.1. IANA mech-profile

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

8.2. IANA OID

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) and to reference this specification in the registry.

9. References

9.1. Normative References


[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.


[Bradner, S.], “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997.

[Fielding, R.], [Gettys, J.], [Moquill, J.], [Frisby, H.], [Masinter, L.], [Leach, P.], and [T. Berners-Lee], “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999.


[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
9.2. Informative References


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.

Appendix B. Changes

This section to be removed prior to publication.

- 08 Fixed text per Gen-Art review
- 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.

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
Richtstrasse 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
URL: http://josefsson.org/