Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 31, 2012

S. Cantor
Shibboleth Consortium
S. Josefsson
SJD AB

February 28, 2012

SAML Enhanced Client SASL and GSS-API Mechanisms
draft-ietf-kitten-sasl-saml-ec-01.txt

Abstract

Security Assertion Markup Language (SAML) 2.0 is a generalized framework for the exchange of security-related information between asserting and relying parties. Simple Authentication and Security Layer (SASL) and the Generic Security Service Application Program Interface (GSS-API) are application frameworks to facilitate an extensible authentication model. This document specifies a SASL and GSS-API mechanism for SAML 2.0 that leverages the capabilities of a SAML-aware "enhanced client" to address significant barriers to federated authentication in a manner that encourages reuse of existing SAML bindings and profiles designed for non-browser scenarios.

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 August 31, 2012.

Copyright Notice

Table of Contents

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. Additional profiles and extensions are also routinely developed and published.

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, POP and XMPP [RFC3920]. The effect is to make authentication modular, so that newer authentication mechanisms can be added as needed.

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 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 uses SASL directly without GSS-API.

The mechanisms specified in this document allow a SASL- or GSS-API-enabled server to act as a SAML relying party, or service provider (SP), by advertising this mechanism as an option for SASL or GSS-API clients that support the use of SAML to communicate identity and attribute information. Clients supporting this mechanism are termed "enhanced clients" in SAML terminology because they understand the federated authentication model and have specific knowledge of the IdP(s) associated with the user. This knowledge, and the ability to act on it, addresses a significant problem with browser-based SAML profiles known as the "discovery", or "where are you from?" (WAYF) problem. Obviating the need for the RP to interact with the client to determine the right IdP (and its network location) is both a user interface and security improvement.

The SAML mechanism described in this document is an adaptation of an existing SAML profile, the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20], and therefore does not establish a separate authentication, integrity and confidentiality mechanism. It is anticipated that existing security layers, such as Transport Layer

Security (TLS) or Secure Shell (SSH), will continued to be used.

Figure 1 describes the interworking between SAML and SASL: this document requires enhancements to the RP and to the client (as the two SASL communication endpoints) but no changes to the SAML IdP are assumed apart from its support for the applicable SAML profile. To accomplish this, a SAML protocol exchange between the RP and the IdP, brokered by the client, is tunneled within SASL. There is no assumed communication between the RP and the IdP, but such communication may occur in conjunction with additional SAML-related profiles not in scope for this document.

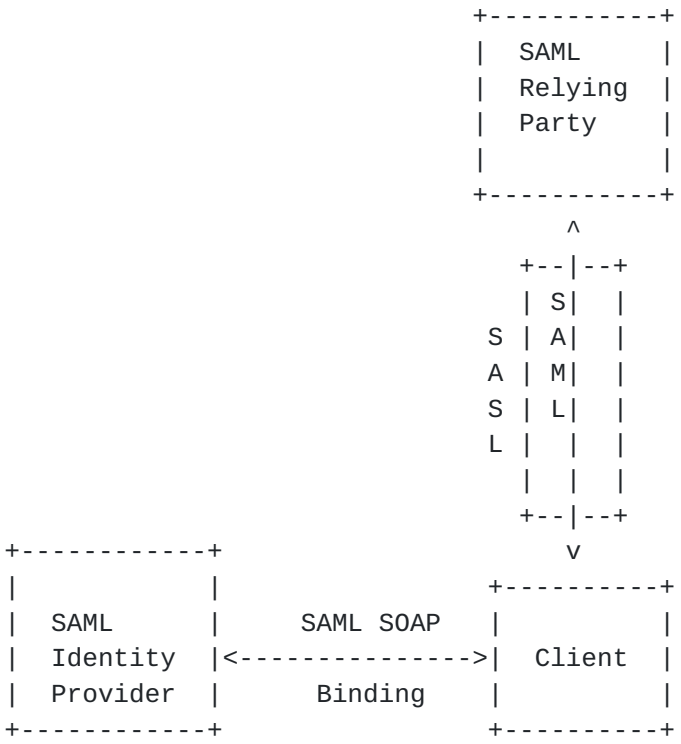

Figure 1: Interworking Architecture

2. 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 also assumed to be familiar with the terms used in the SAML 2.0 specification, and an understanding of the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20] is necessary, as part of this mechanism explicitly reuses and references it.

This document can be implemented without knowledge of GSS-API since the normative aspects of the GS2 protocol syntax have been duplicated in this document. The document may also be implemented to provide a GSS-API mechanism, and then knowledge of GSS-API is essential. To faciliate these two variants, the references has been split into two parts, one part that provides normative references for all readers, and one part that adds additional normative references required for implementers that wish to implement the GSS-API portion.

3. Applicability for Non-HTTP Use Cases

While SAML is designed to support a variety of application scenarios, the profiles for authentication defined in the original standard are designed around HTTP [RFC2616] applications. They are not, however, limited to browsers, because it was recognized that browsers suffer from a variety of functional and security deficiencies that would be useful to avoid where possible. Specifically, the notion of an "Enhanced Client" (or a proxy acting as one on behalf of a browser, thus the term "ECP") was specified for a software component that acts somewhat like a browser from an application perspective, but includes limited, but sufficient, awareness of SAML to play a more conscious role in the authentication exchange between the RP and the IdP. What follows is an outline of the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20], as applied to the web/HTTP service use case:

1. The Enhanced Client requests a resource of a Relying Party (RP) (via an HTTP request). In doing so, it advertises its "enhanced" capability using HTTP headers.

2. The RP, desiring SAML authentication and noting the client's capabilities, responds not with an HTTP redirect or form, but with a SOAP [W3C.soap11] envelope containing a SAML <AuthnRequest> along with some supporting headers. This request identifies the RP (and may be signed), and may provide hints to the client as to what IdPs the RP finds acceptable, but the choice of IdP is generally left to the client.

3. The client is then responsible for delivering the body of the SOAP message to the IdP it is instructed to use (often via configuration ahead of time). The user authenticates to the IdP ahead of, during, or after the delivery of this message, and perhaps explicitly authorizes the response to the RP.

4. Whether authentication succeeds or fails, the IdP responds with its own SOAP envelope, generally containing a SAML <Response> message for delivery to the RP. In a successful case, the message will include a SAML <Assertion> containing authentication, and possibly attribute, information about the user. Either the response or assertion alone is signed, and the assertion may be encrypted to a key negotiated with or known to belong to the RP.

5. The client then delivers the SOAP envelope containing the <Response> to the RP at a location the IdP directs (which acts as an additional, though limited, defense against MITM attacks). This completes the SAML exchange.

6. The RP now has sufficient identity information to approve the original HTTP request or not, and acts accordingly. Everything between the original request and this response can be thought of as an "interruption" of the original HTTP exchange.

When considering this flow in the context of an arbitrary application protocol and SASL, the RP and the client both must change their code to implement this SASL mechanism, but the IdP can remain untouched. The existing RP/client exchange that is tunneled through HTTP maps well to the tunneling of that same exchange in SASL. In the parlance of SASL [RFC4422], this mechanism is "client-first" for consistency with GS2. The steps are shown below:

1. The server MAY advertise the SAML20EC and/or SAML20EC-PLUS mechanisms.

2. The client initiates a SASL authentication with SAML20EC or SAML20EC-PLUS.

3. The server sends the client a challenge consisting of a SOAP envelope containing its SAML <AuthnRequest>.

4. The SASL client unpacks the SOAP message and communicates with its chosen IdP to relay the SAML <AuthnRequest> to it. This communication, and the authentication with the IdP, proceeds separately from the SASL process.

5. Upon completion of the exchange with the IdP, the client responds to the SASL server with a SOAP envelope containing the SAML <Response> it obtained, or a SOAP fault, as warranted.

6. The SASL Server indicates success or failure.

Note: The details of the SAML processing, which are consistent with the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20], are such that the client MUST interact with the IdP in order to complete any SASL exchange with the RP. The assertions issued by the IdP for the purposes of the profile, and by extension this SASL mechanism, are short lived, and therefore cannot be cached by the client for later use.

Encompassed in step four is the client-driven selection of the IdP, authentication to it, and the acquisition of a response to provide to the SASL server. These processes are all external to SASL.

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

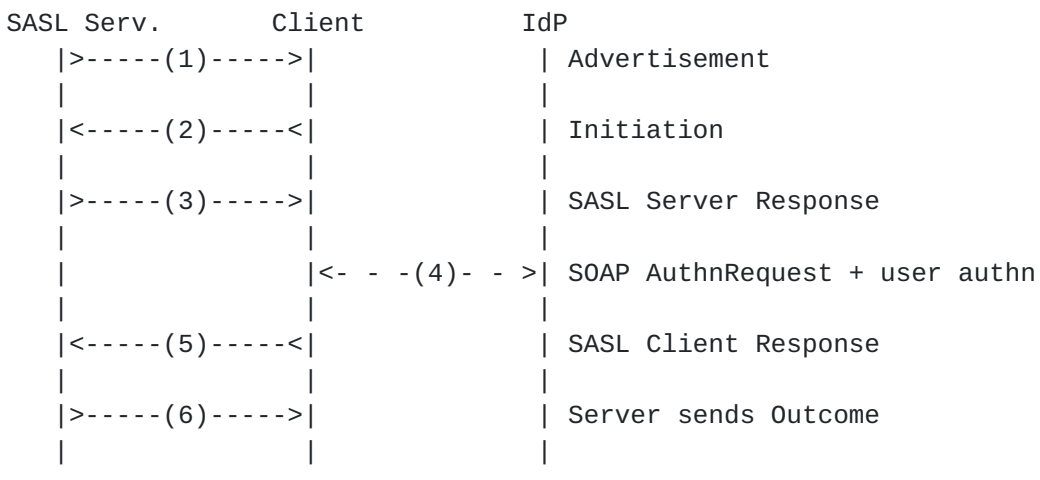

Figure 2: Authentication flow

4. SAML SASL Mechanism Specification

Based on the previous figures, the following operations are defined by the SAML SASL mechanism:

4.1. Advertisement

To advertise that a server supports this mechanism, during application session initiation, it displays the name "SAML20EC" and/or "SAML20EC-PLUS" in the list of supported SASL mechanisms (depending on its support for channel binding).

4.2. Initiation

A client initiates "SAML20EC" or "SAML20EC-PLUS" authentication. If supported by the application protocol, the client MAY include an initial response, otherwise it waits until the server has issued an empty challenge (because the mechanism is client-first).

The format of the initial client response is as follows:

```
       holder-of-key = "urn:oasis:names:tc:SAML:2.0:cm:holder-of-key"

 initial-response = gs2-cb-flag "," [gs2-authzid] "," [holder-of-key]
```

The gs2-cb-flag MUST be set as defined in [RFC5801] to indicate whether the client supports channel binding. This takes the place of the PAOS HTTP header extension used in [SAMLECP20] to indicate channel binding support.

The optional "gs2-authzid" field holds the authorization identity, as requested by the client.

The optional "holder-of-key" field is a constant that signals the client's support for stronger security by means of a locally held key. This takes the place of the PAOS HTTP header extension used in [SAMLECP20] to indicate "holder of key" support.

4.3. Server Response

The SASL server responds with a SOAP envelope constructed in accordance with section 2.3.2 of [SAMLECP20]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing profile. Various SOAP headers are also consumed by the client in exactly the same manner prescribed by that section.

4.4. User Authentication with Identity Provider

Upon receipt of the Server Response (Section 4.3), the steps described in sections 2.3.3 through 2.3.6 of [SAMLECP20] are performed between the client and the chosen IdP. The means by which the client determines the IdP to use, and where it is located, are out of scope of this mechanism.

The exact means of authentication to the IdP are also out of scope, but clients supporting this mechanism MUST support HTTP Basic Authentication as defined in [RFC2617] and TLS client authentication as defined in [RFC5246].

4.5. Client Response

Assuming a response is obtained from the IdP, the client responds to the SASL server with a SOAP envelope constructed in accordance with section 2.3.7 of [SAMLECP20]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing profile. If the client is unable to obtain a response from the IdP, it responds to the SASL server with a SOAP envelope containing a SOAP fault.

4.6. Outcome

The SAML protocol exchange having completed, the SASL server will transmit the outcome to the client depending on local validation of the client responses. This outcome is transmitted in accordance with the application protocol in use.

4.7. Additional Notes

Because this mechanism is an adaptation of an HTTP-based profile, there are a few requirements outlined in [SAMLECP20] that make reference to a response URL that is normally used to regulate where the client returns information to the RP. There are also security-related checks built into the profile that involve this location.

For compatibility with existing IdP and profile behavior, and to provide for secure identification of the RP to the client, the SASL server MUST populate the responseConsumerURL and AssertionConsumerServiceURL attributes with its service name, expressed as an absolute URI. The parties then perform the steps described in [SAMLECP20] as usual.

Similarly, the use of HTTP status signaling between the RP and client mandated by [SAMLECP20] may not be applicable.

5. SAML EC 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 Enhanced Clients mechanism is also a GSS-API mechanism. The messages are the same, but a) the GS2 header on the client's first message is excluded when SAML EC 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 EC is 1.3.6.1.4.1.11591.4.6.

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

The mutual authentication property of this mechanism relies on successfully comparing the server identity from the existing security layer (TLS, SSH, etc.) with the negotiated target name. Since the existing security layer is managed by the application outside of the GSS-API mechanism, the mechanism itself is unable to confirm the name. For this reason, applications MUST match the server identity from the existing security layer with the target name. For TLS, this matching MUST be perfored as discussed in [RFC6125]. For SSH, this matching MUST be performed as discussed in [RFC4462]. Applications may rely on the GSS-API mechanism to perform this matching by passing the server identity as the targ_name parameter to GSS_Init_sec_context().

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

The lifetime of a security context established with this mechanism SHOULD be limited by the value of a SessionNotOnOrAfter attribute, if any, in the <AuthnStatement> of the SAML assertion received by the RP.

5.1. GSS-API Principal Name Types for SAML EC

SAML EC supports standard generic name syntaxes for acceptors such as GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4.1). These service names MUST be associated with the SAML "entityID" claimed by the RP, such as through the use of SAML metadata [OASIS.saml-metadata-2.0-os].

SAML EC 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 EC.

The query, display, and exported name syntaxes for SAML EC principal names are all the same. There are no SAML EC-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, but the "NAME" part of the token should be treated as a potential input string to the SAML EC name normalization rules.

GSS-API name attributes may be defined in the future to hold the normalized SAML EC Identifier.

6. Example

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 (and example.com and example.org have established a SAML-capable trust relationship). 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>SAML20EC</mechanism>
 </mechanisms>
</stream:features>
```

Step 4: Client selects an authentication mechanism and sends the initial client response (it is base64 encoded as specified by the XMPP SASL protocol profile):

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='SAML20EC'>
biws
</auth>
```

The initial response is "n,," which signals that channel binding is

not used, there is no authorization identity, and the client does not support key-based confirmation.

Step 5: Server sends a challenge to client in the form of a SOAP envelope containing its SAML <AuthnRequest>:


```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUNCiAgICB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoy
LjA6YXNzZXJ0aW9uIg0KICAgIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6cHJvdG9jb2wiDQogICAgeG1sbnM6Uz0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5v
cmcvc29hcC9lbnZlbG9wZS8iPg0KICA8UzpIZWFkZXI+DQogICAgPHBhb3M6UmVxdWVzdCB4
bWxuczpwYW9zPSJ1cm46bGliZXJ0eTpwYW9zOjIwMDMtMDgiDQogICAgICBtZXNzYWdlSUQ9
ImMzYTRmOGI5YzJkIiBTOm11c3RVbmRlcnN0YW5kPSIxIg0KICAgICAgUzphY3Rvcj0iaHR0
cDovL3NjaGVtYXMueG1sc29hcC5vcmcvc29hcC9hY3Rvci9uZXh0Ig0KICAgICAgcmVzcG9u
c2VDb25zdW1lclVSTD0iaHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tIg0KICAgICAgc2Vydmlj
ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb2ZpbGVzOlNTTzplY3AiLz4NCiAg
ICA8ZWNwOlJlcXVlc3QNCiAgICAgIHhtbG5zOmVjD0idXJuOm9hc2lzOm5hbWVzOnRjOlNB
TUw6Mi4wOnByb2ZpbGVzOlNTTzplY3AiDQogICAgICBTOmFjdG9yPSJodHRwOi8vc2NoZW1h
cy54bWxzb2FwLm9yZy9zb2FwL2FjdG9yL25leHQiDQogICAgICBTOm11c3RVbmRlcnN0YW5k
PSIxIiBQcm92aWRlck5hbWU9IkphYmJlciBhdCBleGFtcGxlLmNvbSI+DQogICAgICA8c2Ft
bDpJc3N1ZXI+aHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tPC9zYW1sOklzc3Vlcj4NCiAgICA8
L2VjcDpSZXF1ZXN0Pg0KICA8L1M6SGVhZGVyPg0KICA8UzpCb2R5Pg0KICAgIDxzYW1sOkFt
dXRoblJlcXVlc3QNCiAgICAgIElEPSJjM2E0ZjhiOWMyZCIgVmVyc2lvbj0iMi4wIiBJc3N1
ZUluc3RhbnQ9IjIwMDctMTItMTBUMTE6Mzk6MzRaIg0KICAgICAgUHJvdG9jb2xCaW5kaW5n
PSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6YmluZGluZ3M6UEFPUyINCiAgICAgIEFz
c2VydGlvbkNvbnN1bWVyU2VydmljZVVSTD0iaHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tIj4N
CiAgICAgIDxzYW1sOklzc3VlciB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6YXNzZXJ0aW9uIj4NCiAgICAgICAgICBodHRwczovL3htcHAuZXhhbXBsZS5jb20NCiAg
ICAgIDwvc2FtbDpJc3N1ZXI+DQogICAgICA8c2FtbHA6TmFtZUlEUG9saWN5IEFsbG93Q3Jl
YXRlPSJ0cnVlIg0KICAgICAgICBGb3JtYXQ9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIu
MDpuYW1laWQtZm9ybWF0OnBlcnNpc3RlbnQiLz4NCiAgICA8L3NhbWxwOk5hbWVJRFBvbGlj
eT4NCiAgICA8c2FtbHA6UmVxdWVzdGVkQXV0aG5Db250ZXh0DQogICAgICBDb21wYXJpc29u
PSJleGFjdCI+DQogICAgICA8c2FtbDpBdXRobkNvbnRleHRDbGFzc1JlZj5pDQogICAgICAg
dXRobkNvbnRleHQgQ29tcGFyaXNvbj0iZXhhY3QiPg0KICAgICAgICA8c2FtbDpBdXRobkNv
dGV4dENsYXNzUmVmPg0KICAgICAgICAgIHVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDphYzpj
bGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0DQogICAgICAgICAgPC9zYW1sOkF1dGhu
Q29udGV4dENsYXNzUmVmPg0KICAgICAgPC9zYW1scDpSZXF1ZXN0ZWRBdXRobkNvbnRleHQ+
IA0KICAgIDwvc2FtbHA6QXV0aG5SZXF1ZXN0Pg0KICA8L1M6Qm9keT4NCjwvUzpFbnZlbG9w
ZT4NCg==
</challenge>
```


The Base64 [RFC4648] decoded envelope:

```
<S:Envelope
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <paos:Request xmlns:paos="urn:liberty:paos:2003-08"
      messageID="c3a4f8b9c2d" S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      responseConsumerURL="xmpp:xmpp.example.com"
      service="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"/>
    <ecp:Request
      xmlns:ecp="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      S:mustUnderstand="1" ProviderName="Jabber at example.com">
      <saml:Issuer>https://xmpp.example.com</saml:Issuer>
    </ecp:Request>
  </S:Header>
  <S:Body>
    <samlp:AuthnRequest
      ID="c3a4f8b9c2d" Version="2.0" IssueInstant="2007-12-10T11:39:34Z"
      ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS"
      AssertionConsumerServiceURL="xmpp:xmpp.example.com">
      <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
       https://xmpp.example.com
      </saml:Issuer>
      <samlp:NameIDPolicy AllowCreate="true"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
      <samlp:RequestedAuthnContext Comparison="exact">
       <saml:AuthnContextClassRef>
       urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
       </saml:AuthnContextClassRef>
      </samlp:RequestedAuthnContext>
    </samlp:AuthnRequest>
  </S:Body>
</S:Envelope>
```

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 relays the request to IdP in a SOAP message transmitted over HTTP (over TLS). HTTP portion not shown, use of

Basic Authentication is assumed. The body of the SOAP envelope is exactly the same as received in the previous step.

```
<S:Envelope
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <samlp:AuthnRequest>
        <!-- same as above -->
        </samlp:AuthnRequest>
    </S:Body>
</S:Envelope>
```

Step 7: IdP responds to client with a SOAP response containing a SAML <Response> containing a short-lived SSO assertion (shown as an encrypted variant in the example).

```
<S:Envelope
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <ecp:Response S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      AssertionConsumerServiceURL="xmpp:xmpp.example.com"/>
  </S:Header>
  <S:Body>
    <samlp:Response ID="d43h94r389309r" Version="2.0"
        IssueInstant="2007-12-10T11:42:34Z" InResponseTo="c3a4f8b9c2d"
        Destination="xmpp:xmpp.example.com">
      <saml:Issuer>https://saml.example.org</saml:Issuer>
      <samlp:Status>
        <samlp:StatusCode
            Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
      </samlp:Status>
      <saml:EncryptedAssertion>
        <!-- contents elided -->
      </saml:EncryptedAssertion>
    </samlp:Response>
  </S:Body>
</S:Envelope>
```

Step 8: Client sends SOAP envelope containing the SAML <Response> as

a response to the SASL server's challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUNCiAgICB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoy
LjA6YXNzZXJ0aW9uIg0KICAgIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6cHJvdG9jb2wiDQogICAgeG1sbnM6Uz0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5v
cmcvc29hcC9lbnZlbG9wZS8iPg0KICA8UzpIZWFkZXI+DQogICAgPHBhb3M6UmVzcG9uc2Ug
eG1sbnM6cGFvcz0idXJuOmxpYmVydHk6cGFvczoyMDAzLTA4Ig0KICAgICAgUzphY3Rvcj0i
aHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvc29hcC9hY3Rvci9uZXh0Ig0KICAgICAgUzpt
dXN0VW5kZXJzdGFuZD0iMSIgcmVmVG9NZXNzYWdlSUQ9IjZjM2E0ZjhiOWMyZCIvPg0KICA8
L1M6SGVhZGVyPg0KICA8UzpCb2R5Pg0KICAgIDxzYW1scDpSZXNwb25zZSBJRD0iZDQzaDk0
cjM4OTMwOXIiIFZlcnNpb249IjIuMCINCiAgICAgICAgSXNzdWVJbnN0YW50PSIyMDA3LTEy
LTEwVDExOjQyOjM0WiIgSW5SZXNwb25zZVRvPSJjM2E0ZjhiOWMyZCINCiAgICAgICAgRGVz
dGluYXRpb249Imh0dHBzOi8veG1wcC5leGFtcGxlLmNvbSI+DQogICAgICA8c2FtbDpJc3N1
ZXI+aHR0cHM6Ly9zYW1sLmV4YW1wbGUub3JnPC9zYW1sOklzc3Vlcj4NCiAgICAgIDxzYW1s
cDpTdGF0dXM+DQogICAgICAgIDxzYW1scDpTdGF0dXNDb2RlDQogICAgICAgICAgICBWYWx1
ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1czpTdWNjZXNzIi8+DQogICAg
ICA8L3NhbWxwOlN0YXR1cz4NCiAgICAgIDxzYW1sOkVuY3J5cHRlZEFzc2VydGlvbj4NCiAg
ICAgICAgPENvbnRlbnQgLS0+DQogICAgICA8L3NhbWw6RW5jcnlwdGVkQXNzZXJ0aW9uPg0K
QXNzZXJ0aW9uPg0KICAgIDwvc2FtbHA6UmVzcG9uc2U+DQogICAgPC9wYW9TOkVudmVsb3BlPg0K
dmVsb3BlPg0K
</response>
```

The Base64 [RFC4648] decoded envelope:

```
<S:Envelope
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <paos:Response xmlns:paos="urn:liberty:paos:2003-08"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      S:mustUnderstand="1" refToMessageID="6c3a4f8b9c2d"/>
  </S:Header>
  <S:Body>
    <samlp:Response ID="d43h94r389309r" Version="2.0"
        IssueInstant="2007-12-10T11:42:34Z" InResponseTo="c3a4f8b9c2d"
        Destination="xmpp:xmpp.example.com">
      <saml:Issuer>https://saml.example.org</saml:Issuer>
      <samlp:Status>
        <samlp:StatusCode
            Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
      </samlp:Status>
      <saml:EncryptedAssertion>
        <!-- contents elided -->
      </saml:EncryptedAssertion>
    </samlp:Response>
  </S:Body>
</S:Envelope>
```

Step 9: Server informs client of successful authentication:

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

Step 9 (alt): Server informs client of failed authentication:

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

Step 10: 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 11: 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 12: 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 13: 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.

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.

Version 2.0 of the Enhanced Client or Proxy Profile [SAMLECP20] adds optional support for channel binding and use of "Holder of Key" subject confirmation. The former is strongly recommended for use with this mechanism to detect "Man in the Middle" attacks between the client and the RP without relying on flawed commercial TLS infrastructure. The latter may be impractical in many cases, but is a valuable way of strengthening client authentication, protecting against phishing, and improving the overall mechanism.

7.1. Risks Left Unaddressed

The adaptation of a web-based profile that is largely designed around security-oblivious clients and a bearer model for security token validation results in a number of basic security exposures that should be weighed against the compatibility and client simplification benefits of this mechanism.

When channel binding is not used, protection against "Man in the Middle" attacks is left to lower layer protocols such as TLS, and the development of user interfaces able to implement that has not been effectively demonstrated. Failure to detect a MITM can result in phishing of the user's credentials if the attacker is between the client and IdP, or the theft and misuse of a short-lived credential (the SAML assertion) if the attacker is able to impersonate a RP. SAML allows for source address checking as a minor mitigation to the latter threat, but this is often impractical. IdPs can mitigate to some extent the exposure of personal information to RP attackers by encrypting assertions with authenticated keys.

7.2. 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 activity, but there is nothing technically that prohibits the collection of this information. SASL servers should be aware that SAML IdPs will track - to some extent - user access to their services.

It is also out of scope of the mechanism to determine under what

conditions an IdP will release particular information to a relying party, and it is generally unclear in what fashion user consent could be established in real time for the release of particular information. The SOAP exchange with the IdP does not preclude such interaction, but neither does it define that interoperably.

7.3. Collusion between RPs

Depending on the information supplied by the IdP, it may be possible for RPs to correlate data that they have collected. By using the same identifier to log into every RP, collusion between RPs is possible. SAML supports the notion of pairwise, or targeted/ directed, identity. This allows the IdP to manage opaque, pairwise identifiers for each user that are specific to each RP. However, correlation is often possible based on other attributes supplied, and is generally a topic that is beyond the scope of this mechanism. It is sufficient to say that this mechanism does not introduce new correlation opportunities over and above the use of SAML in web-based use cases.

8. IANA Considerations

The IANA is requested to register the following SASL profile:

SASL mechanism profiles: SAML20EC and SAML20EC-PLUS

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

[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.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, May 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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.

[SAMLECP20]
Cantor, S., "SAML V2.0 Enhanced Client or Proxy Profile Version 2.0", OASIS Working Draft OASIS.sstc-saml-ecp-v2.0-wd04, August 2011.

[W3C.soap11]
Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C Note soap11, May 2000, <http://www.w3.org/TR/SOAP/>.

9.2. Normative References for GSS-API Implementers

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[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.

9.3. Informative References

[OASIS.saml-metadata-2.0-os]
Cantor, S., Moreh, J., Philpott, R., and E. Maler, "Metadata for the Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-metadata-2.0-os, March 2005.

[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.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

Appendix A. Acknowledgments

The authors would like to thank Klaas Wierenga, Sam Hartman, and Nico Williams for their contributions.

Appendix B. Changes

This section to be removed prior to publication.

- 01, SSH language added, noted non-assumption of HTTP error handling, added guidance on life of security context.

- 00, Initial Revision, first WG-adopted draft. Removed support for unsolicited SAML responses.

Authors' Addresses

Scott Cantor
Shibboleth Consortium
2740 Airport Drive
Columbus, Ohio 43219
United States

Phone: +1 614 247 6147
Email: cantor.2@osu.edu

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

Email: simon@josefsson.org
URI: http://josefsson.org/