

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 28, 2010

S. Cantor
Internet2
May 27, 2010

A SASL Mechanism for SAML Enhanced Clients
draft-cantor-ietf-sasl-saml-ec-00.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) is an application framework to facilitate an extensible authentication model. This document specifies a SASL 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 November 28, 2010.

Copyright Notice

Copyright (c) 2010 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	3
2. Terminology	5
3. Applicability for Non-HTTP Use Cases	6
4. SAML SASL Mechanism Specification	9
4.1. Advertisement	9
4.2. Initiation	9
4.3. Server Response	9
4.4. User Authentication with Identity Provider	9
4.5. Client Response	9
4.6. Outcome	10
4.7. Additional Notes	10
5. Example	11
6. Security Considerations	18
6.1. Risks Left Unaddressed	18
6.2. User Privacy	18
6.3. Collusion between RPs	19
7. IANA Considerations	20
8. Normative References	21
Appendix A. Acknowledgments	22
Appendix B. Changes	23
Author's Address	24

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

The mechanism specified in this document allows a SASL-enabled server to act as a SAML relying party, or service provider (SP), by advertising this mechanism as an option for SASL 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 [OASIS.saml-profiles-2.0-os], 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), will continue 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.

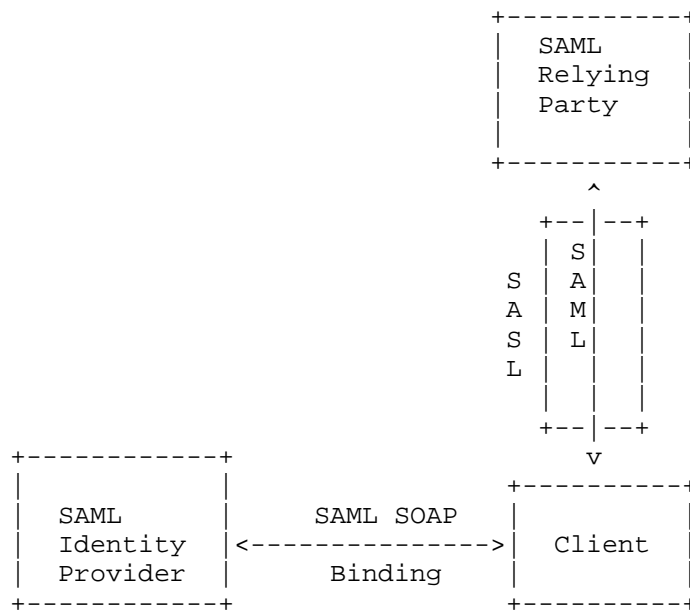


Figure 1: Interworking Architecture

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 [OASIS.saml-profiles-2.0-os] is necessary, as part of this mechanism explicitly reuses and references it.

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 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 acted like a browser from an application perspective, but included 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 [OASIS.saml-profiles-2.0-os], 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 in a new envelope 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 a new 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 also maps well to the tunneling of that same exchange in SASL. In the parlance of SASL [RFC4422], this mechanism is "variable", in that the client can accompany its authentication request with an "initial response" consisting of a SAML <Response> obtained from an IdP. The steps are shown from below:

1. The server MAY advertise the SAML20EC capability.
2. The client initiates a SASL authentication with SAML20EC. It MAY include an initial response.
3. The server sends the client one of two responses:
 1. an indication of success or failure (if the client included an initial response).
 2. a challenge containing a BASE64-encoded SOAP envelope containing a SAML <AuthnRequest>.
4. In the latter case, 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 BASE64-encoded 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 existing Enhanced Client or Proxy (ECP) Profile [OASIS.saml-profiles-2.0-os], 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:

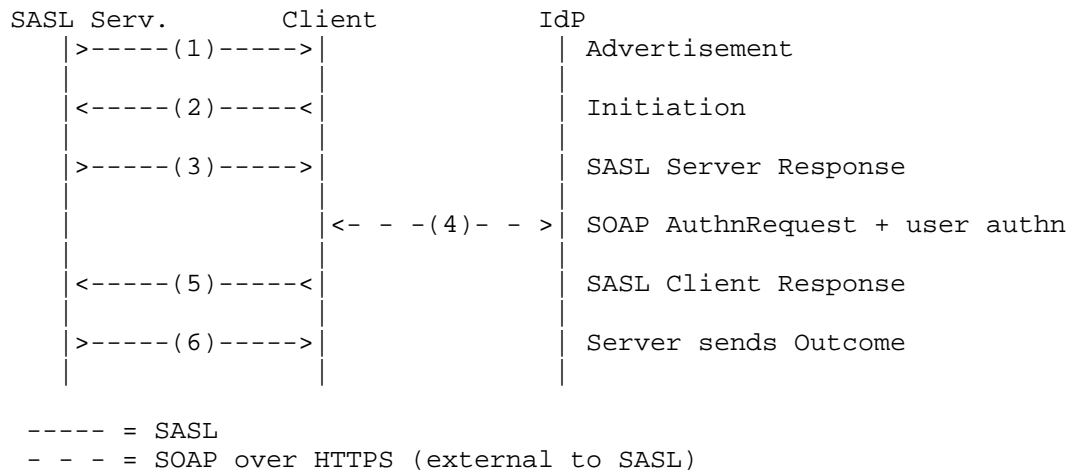


Figure 2: Authentication flow (no initial response)

An alternative in which the client interacts with the IdP ahead of time:

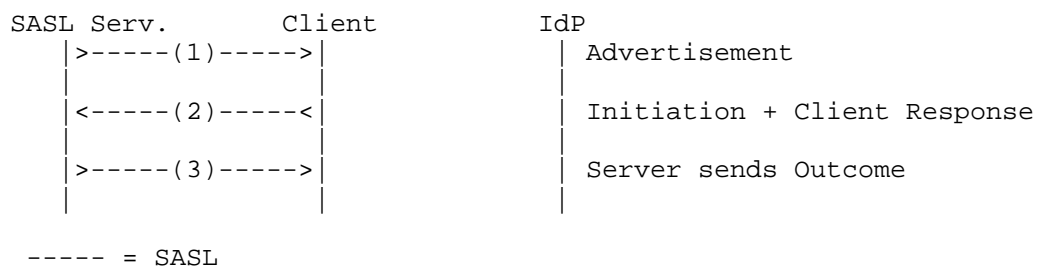


Figure 3: Authentication flow (with initial response)

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" in the list of supported SASL mechanisms.

4.2. Initiation

A client initiates "SAML20EC" authentication. If supported by the application protocol, the client MAY include an initial response in the same form described below (Section 4.5).

4.3. Server Response

Assuming no initial response from the client, the SASL server responds with a BASE64 [RFC4648] encoded SOAP envelope constructed in accordance with section 4.2.3.2 of [OASIS.saml-profiles-2.0-os]. 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.

4.4. User Authentication with Identity Provider

Upon receipt of the Server Response (Section 4.3), the steps described in sections 4.2.3.3 through 4.2.3.6 of [OASIS.saml-profiles-2.0-os] 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 SHOULD support client authentication via TLS 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 BASE64 [RFC4648] encoded SOAP envelope constructed in accordance with section 4.2.3.7 of [OASIS.saml-profiles-2.0-os]. 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 base64-encoded 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.

4.7. Additional Notes

Because this mechanism is an adaptation of an HTTP-based profile, there are a few requirements outlined in [OASIS.saml-profiles-2.0-os] 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, one or more URLs MUST be associated with the SASL server and used to populate the responseConsumerURL and AssertionConsumerServiceURL XML attributes described in the profile. The parties then perform the steps described in [OASIS.saml-profiles-2.0-os] as usual.

A simple means of fulfilling this requirement is to populate this URL with the RP's SAML "entityID", which is a unique identifier that is required of all SAML RPs.

5. 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:

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

Step 5: Server sends a BASE64 [RFC4648] encoded challenge to client in the form of a SOAP envelope containing its SAML `<AuthnRequest>`:

The 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="https://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="https://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="https://xmpp.example.com"/>
  </S:Header>
  <S:Body>
    <samlp:Response ID="d43h94r389309r" Version="2.0"
      IssueInstant="2007-12-10T11:42:34Z" InResponseTo="c3a4f8b9c2d"
      Destination="https://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 BASE64 [RFC4648] encoded SOAP 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="https://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.

6. 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.

6.1. 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 basis security exposures that should be weighed against the compatibility and client simplification benefits of this mechanism.

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.

This mechanism also does not support the use of channel bindings or supply a SASL security layer, so there is no assurance that the TLS endpoints are related to the SASL endpoints.

6.2. 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 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.

6.3. 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.

7. IANA Considerations

The IANA is requested to register the following SASL profile:

SASL mechanism profile: SAML20EC

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. 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.
- [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.
- [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.
- [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.
- [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/>>.

Appendix A. Acknowledgments

The author would like to thank Klaas Wierenga and Sam Hartman for their contributions.

Appendix B. Changes

This section to be removed prior to publication.

- o 00 Initial Revision, largely adapted from draft-wierenga-ietf-sasl-saml-00.

Author's Address

Scott Cantor
Internet2
2740 Airport Drive
Columbus, Ohio 43219
United States

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 14, 2013

S. Hartman, Ed.
Painless Security
J. Howlett
JANET
August 13, 2012

A GSS-API Mechanism for the Extensible Authentication Protocol
draft-ietf-abfab-gss-eap-09.txt

Abstract

This document defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (GSS-API) when using the Extensible Authentication Protocol mechanism. Through the GS2 family of mechanisms defined in RFC 5801, these protocols also define how Simple Authentication and Security Layer (SASL, RFC 4422) applications use the Extensible Authentication Protocol.

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 February 14, 2013.

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	4
1.1. Discovery	5
1.2. Authentication	5
1.3. Secure Association Protocol	6
2. Requirements notation	8
3. EAP Channel Binding and Naming	9
3.1. Mechanism Name Format	9
3.2. Internationalization of Names	12
3.3. Exported Mechanism Names	12
3.4. Acceptor Name RADIUS AVP	13
3.5. Proxy Verification of Acceptor Name	13
4. Selection of EAP Method	15
5. Context Tokens	16
5.1. Mechanisms and Encryption Types	17
5.2. Processing received tokens	17
5.3. Error Subtokens	18
5.4. Initial State	18
5.4.1. Vendor Subtoken	19
5.4.2. Acceptor Name Request	19
5.4.3. Acceptor Name Response	19
5.5. Authenticate State	20
5.5.1. EAP Request Subtoken	21
5.5.2. EAP Response Subtoken	21
5.6. Extension State	21
5.6.1. Flags Subtoken	22
5.6.2. GSS Channel Bindings Subtoken	22
5.6.3. MIC Subtoken	23
5.7. Example Token	24
5.8. Context Options	24
6. Acceptor Services	26
6.1. GSS-API Channel Binding	26
6.2. Per-message security	27
6.3. Pseudo Random Function	27
7. Iana Considerations	28
7.1. OID Registry	28
7.2. RFC 4121 Token Identifiers	29
7.3. GSS EAP Subtoken Types	29
7.4. RADIUS Attribute Assignments	30
7.5. Registration of the EAP-AES128 SASL Mechanisms	31
7.6. GSS EAP Errors	31
7.7. GSS EAP Context Flags	32

8. Security Considerations	34
9. Acknowledgements	36
10. References	37
10.1. Normative References	37
10.2. Informative References	38
Appendix A. Pre-Publication RADIUS VSA	40
Authors' Addresses	41

1. Introduction

ABFAB [I-D.ietf-abfab-arch] describes an architecture for providing federated access management to applications using the Generic Security Services Application Programming Interface (GSS-API) [RFC2743] and Simple Authentication and Security Layers (SASL) [RFC4422]. This specification provides the core mechanism for bringing federated authentication to these applications.

The Extensible Authentication Protocol (EAP) [RFC3748] defines a framework for authenticating a network access client and server in order to gain access to a network. A variety of different EAP methods are in wide use; one of EAP's strengths is that for most types of credentials in common use, there is an EAP method that permits the credential to be used.

EAP is often used in conjunction with a backend Authentication , Authorization and Accounting (AAA) server via RADIUS [RFC3579] or Diameter [RFC4072]. In this mode, the Network Access Server (NAS) simply tunnels EAP packets over the backend authentication protocol to a home EAP/AAA server for the client. After EAP succeeds, the backend authentication protocol is used to communicate key material to the NAS. In this mode, the NAS need not be aware of or have any specific support for the EAP method used between the client and the home EAP server. The client and EAP server share a credential that depends on the EAP method; the NAS and AAA server share a credential based on the backend authentication protocol in use. The backend authentication server acts as a trusted third party enabling network access even though the client and NAS may not actually share any common authentication methods. As described in the architecture document, using AAA proxies, this mode can be extended beyond one organization to provide federated authentication for network access.

The GSS-API provides a generic framework for applications to use security services including authentication and per-message data security. Between protocols that support GSS-API directly or protocols that support SASL [RFC4422], many application protocols can use GSS-API for security services. However, with the exception of Kerberos [RFC4121], few GSS-API mechanisms are in wide use on the Internet. While GSS-API permits an application to be written independent of the specific GSS-API mechanism in use, there is no facility to separate the server from the implementation of the mechanism as there is with EAP and backend authentication servers.

The goal of this specification is to combine GSS-API's support for application protocols with EAP/AAA's support for common credential types and for authenticating to a server without requiring that server to specifically support the authentication method in use. In

addition, this specification supports the architectural goal of transporting attributes about subjects to relying parties. Together this combination will provide federated authentication and authorization for GSS-API applications. This specification meets the applicability requirements for EAP to application authentication [I-D.ietf-abfab-eapapplicability].

This mechanism is a GSS-API mechanism that encapsulates an EAP conversation. From the perspective of RFC 3748, this specification defines a new lower-layer protocol for EAP. From the perspective of the application, this specification defines a new GSS-API mechanism.

Section 1.3 of [RFC5247] outlines the typical conversation between EAP peers where an EAP key is derived:

- o Phase 0: Discovery
- o Phase 1: Authentication
 - o 1a: EAP authentication
 - o 1b: AAA Key Transport (optional)
- o Phase 2: Secure Association Protocol
 - o 2a: Unicast Secure Association
 - o 2b: Multicast Secure Association (optional)

1.1. Discovery

GSS-API peers discover each other and discover support for GSS-API in an application-dependent mechanism. SASL [RFC4422] describes how discovery of a particular SASL mechanism such as a GSS-API mechanism is conducted. The Simple and Protected Negotiation mechanism (SPNEGO) [RFC4178] provides another approach for discovering what GSS-API mechanisms are available. The specific approach used for discovery is out of scope for this mechanism.

1.2. Authentication

GSS-API authenticates a party called the GSS-API initiator to the GSS-API acceptor, optionally providing authentication of the acceptor to the initiator. Authentication starts with a mechanism-specific message called a context token sent from the initiator to the acceptor. The acceptor responds, followed by the initiator, and so on until authentication succeeds or fails. GSS-API context tokens are reliably delivered by the application using GSS-API. The

application is responsible for in-order delivery and retransmission.

EAP authenticates a party called a peer to a party called the EAP server. A third party called an EAP passthrough authenticator may decapsulate EAP messages from a lower layer and reencapsulate them into an AAA protocol. The term EAP authenticator refers to whichever of the passthrough authenticator or EAP server receives the lower-layer EAP packets. The first EAP message travels from the authenticator to the peer; a GSS-API message is sent from the initiator to acceptor to prompt the authenticator to send the first EAP message. The EAP peer maps onto the GSS-API initiator. The role of the GSS-API acceptor is split between the EAP authenticator and the EAP server. When these two entities are combined, the division resembles GSS-API acceptors in other mechanisms. When a more typical deployment is used and there is a passthrough authenticator, most context establishment takes place on the EAP server and per-message operations take place on the authenticator. EAP messages from the peer to the authenticator are called responses; messages from the authenticator to the peer are called requests.

Because GSS-API applications provide guaranteed delivery of context tokens, the EAP retransmission timeout MUST be infinite and the EAP layer MUST NOT retransmit a message.

This specification permits a GSS-API acceptor to hand-off the processing of the EAP packets to a remote EAP server by using AAA protocols such as RADIUS, RadSec or Diameter. In this case, the GSS-API acceptor acts as an EAP pass-through authenticator. The pass-through authenticator is responsible for retransmitting AAA messages if a response is not received from the AAA server. If a response cannot be received, then the authenticator generates an error at the GSS-API level. If EAP authentication is successful, and where the chosen EAP method supports key derivation, EAP keying material may also be derived. If an AAA protocol is used, this can also be used to replicate the EAP Key from the EAP server to the EAP authenticator.

See Section 5 for details of the authentication exchange.

1.3. Secure Association Protocol

After authentication succeeds, GSS-API provides a number of per-message security services that can be used:

GSS_Wrap() provides integrity and optional confidentiality for a message.

GSS_GetMIC() provides integrity protection for data sent independently of the GSS-API

GSS_Pseudo_random [RFC4401] provides key derivation functionality.

These services perform a function similar to secure association protocols in network access. Like secure association protocols, these services need to be performed near the authenticator/acceptor even when a AAA protocol is used to separate the authenticator from the EAP server. The key used for these per-message services is derived from the EAP key; the EAP peer and authenticator derive this key as a result of a successful EAP authentication. In the case that the EAP authenticator is acting as a pass-through it obtains it via the AAA protocol. See Section 6 for details.

2. Requirements notation

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 [RFC2119].

3. EAP Channel Binding and Naming

EAP authenticates a user to a realm. The peer knows that it has exchanged authentication with an EAP server in a given realm. Today, the peer does not typically know which NAS it is talking to securely. That is often fine for network access. However privileges to delegate to a chat server seem very different than privileges for a file server or trading site. Also, an EAP peer knows the identity of the home realm, but perhaps not even the visited realm.

In contrast, GSS-API takes a name for both the initiator and acceptor as inputs to the authentication process. When mutual authentication is used, both parties are authenticated. The granularity of these names is somewhat mechanism dependent. In the case of the Kerberos mechanism, the acceptor name typically identifies both the protocol in use (such as IMAP) and the specific instance of the service being connected to. The acceptor name almost always identifies the administrative domain providing service.

An EAP GSS-API mechanism needs to provide GSS-API naming semantics in order to work with existing GSS-API applications. EAP channel binding [I-D.ietf-emu-chbind] is used to provide GSS-API naming semantics. Channel binding sends a set of attributes from the peer to the EAP server either as part of the EAP conversation or as part of a secure association protocol. In addition, attributes are sent in the backend authentication protocol from the authenticator to the EAP server. The EAP server confirms the consistency of these attributes. Confirming attribute consistency also involves checking consistency against a local policy database as discussed in Section 3.5. In particular, the peer sends the name of the acceptor it is authenticating to as part of channel binding. The acceptor sends its full name as part of the backend authentication protocol. The EAP server confirms consistency of the names.

EAP channel binding is easily confused with a facility in GSS-API also called channel binding. GSS-API channel binding provides protection against man-in-the-middle attacks when GSS-API is used as authentication inside some tunnel; it is similar to a facility called cryptographic binding in EAP. See [RFC5056] for a discussion of the differences between these two facilities and Section 6.1 for how GSS-API channel binding is handled in this mechanism.

3.1. Mechanism Name Format

Before discussing how the initiator and acceptor names are validated in the AAA infrastructure, it is necessary to discuss what composes a name for an EAP GSS-API mechanism. GSS-API permits several types of generic names to be imported using `GSS_Import_name()`. Once a

mechanism is chosen, these names are converted into a mechanism-specific name called a "Mechanism Name". Note that a Mechanism Name is the name of an initiator or acceptor, not of a GSS-API mechanism. This section first discusses the mechanism name form and then discusses what name forms are supported.

The string representation of the GSS-EAP mechanism name has the following ABNF [RFC5234] representation:

```
char-normal = %x00-2E/%x30-3F/%x41-5B/%x5D-FF
char-escaped = "\" %x2F / "\" %x40 / "\" %x5C
name-char = char-normal / char-escaped
name-string = 1*name-char
user-or-service = name-string
host = [name-string]
realm = name-string
service-specific = name-string
service-specifics = service-specific 0*("/" service-specifics)
name = user-or-service ["/" host [ "/" service-specifics]] [ "@"
    realm ]
```

Special characters appearing in a name can be backslash escaped to avoid their special meanings. For example "\\" represents a literal backslash. This escaping mechanism is a property of the string representation; if the components of a name are transported in some mechanism that will keep them separate without backslash escaping, then backslash SHOULD have no special meaning.

The user-or-service component is similar to the portion of a network access identifier (NAI) before the '@' symbol for initiator names and the service name from the registry of GSS-API host-based services in the case of acceptor names [GSS-IANA]. The NAI specification provides rules for encoding and string preparation in order to support internationalization of NAIs; implementations of this mechanism MUST NOT prepare the user-or-service according to these rules; see Section 3.2 for internationalization of this mechanism. The host portion is empty for initiators and typically contains the domain name of the system on which an acceptor service is running. Some services MAY require additional parameters to distinguish the entity being authenticated against. Such parameters are encoded in the service-specifics portion of the name. The EAP server MUST reject authentication of any acceptor name that has a non-empty service-specifics component unless the EAP server understands the service-specifics and authenticates them. The interpretation of the service-specifics is scoped by the user-or-service portion. The realm is similar to the the realm portion of a NAI for initiator names; again the NAI specification's internationalization rules MUST NOT be applied to the realm. The realm is the administrative realm

of a service for an acceptor name.

The string representation of this name form is designed to be generally compatible with the string representation of Kerberos names defined in [RFC1964].

The GSS_C_NT_USER_NAME form represents the name of an individual user. From the standpoint of this mechanism it may take the form either of an undecorated user name or a name semantically similar to a network access identifier (NAI) [RFC4282]. The name is split at the first at-sign ('@') into the part preceeding the realm which is the user-or-service portion of the mechanism name and the realm portion which is the realm portion of the mechanism name.

The GSS_C_NT_HOSTBASED_SERVICE name form represents a service running on a host; it is textually represented as "service@host". This name form is required by most SASL profiles and is used by many existing applications that use the Kerberos GSS-API mechanism. While support for this name form is critical, it presents an interesting challenge in terms of EAP channel binding. Consider a case where the server communicates with a "server proxy," or a AAA server near the server. That server proxy communicates with the EAP server. The EAP server and server proxy are in different administrative realms. The server proxy is in a position to verify that the request comes from the indicated host. However the EAP server cannot make this determination directly. So, the EAP server needs to determine whether to trust the server proxy to verify the host portion of the acceptor name. This trust decision depends both on the host name and the realm of the server proxy. In effect, the EAP server decides whether to trust that the realm of the server proxy is the right realm for the given hostname and then makes a trust decision about the server proxy itself. The same problem appears in Kerberos: there, clients decide what Kerberos realm to trust for a given hostname. The service portion of this name is imported into the user-or-service portion of the mechanism name; the host portion is imported into the host portion of the mechanism name. The realm portion is empty. However, authentication will typically fail unless some AAA component indicates the realm to the EAP server. If the application server knows its realm, then it should be indicated in the outgoing AAA request. Otherwise, a proxy SHOULD add the realm. An alternate form of this name type MAY be used on acceptors; in this case the name form is "service" with no host component. This is imported with the service as user-or-service and an empty host and realm portion. This form is useful when a service is unsure which name an initiator knows it by.

If the null name type or the GSS_EAP_NT_EAP_NAME (OID 1.3.6.1.5.5.15.2.1) (see Section 7.1) is imported, then the string

representation above should be directly imported. Mechanisms MAY support the GSS_KRB5_NT_KRB5_PRINCIPAL_NAME name form with the OID {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) krb5(2) krb5_name(1)}. In many circumstances, Kerberos GSS-API mechanism names will behave as expected when used with the GSS-API EAP mechanism, but there are some differences that may cause some confusion. If an implementation does support importing Kerberos names it SHOULD fail the import if the Kerberos name is not syntactically a valid GSS-API EAP mechanism name as defined in this section.

3.2. Internationalization of Names

For the most part, GSS-EAP names are transported in other protocols; those protocols define the internationalization semantics. For example, if an AAA server wishes to communicate the user-or-service portion of the initiator name to an acceptor, it does so using existing mechanisms in the AAA protocol. Existing internationalization rules are applied. Similarly, within an application, existing specifications such as [RFC5178] define the encoding of names that are imported and displayed with the GSS-API.

This mechanism does introduce a few cases where name components are sent. In these cases the encoding of the string is UTF-8. Senders SHOULD NOT normalize or map strings before sending. These strings include RADIUS attributes introduced in Section 3.4.

When comparing the host portion of a GSS-EAP acceptor name supplied in EAP channel binding by a peer to that supplied by an acceptor, EAP servers SHOULD prepare the host portion according to [RFC5891] prior to comparison. Applications MAY prepare domain names prior to importing them into this mechanism.

3.3. Exported Mechanism Names

GSS-API provides the GSS_Export_name call. This call can be used to export the binary representation of a name. This name form can be stored on access control lists for binary comparison.

The exported name token MUST use the format described in section 3.2 of RFC 2743. The mechanism specific portion of this name token is the string format of the mechanism name described in Section 3.1.

RFC 2744 [RFC2744] places the requirement that the result of importing a name, canonicalizing it to a Mechanism Name and then exporting it needs to be the same as importing that name, obtaining credentials for that principal, initiating a context with those credentials and exporting the name on the acceptor. In practice, GSS

mechanisms often, but not always meet this requirement. For names expected to be used as initiator names, this requirement is met. However, permitting empty host and realm components when importing hostbased services may make it possible for an imported name to differ from the exported name actually used. Other mechanisms such as Kerberos have similar situations where imported and exported names may differ.

3.4. Acceptor Name RADIUS AVP

See Section 7.4 for registrations of RADIUS attribute types to carry the acceptor service name. All the attribute types registered in that section are strings. See Section 3.1 for details of the values in a name.

If RADIUS is used as an AAA transport, the acceptor **MUST** send the acceptor name in these attribute types. That is, the acceptor decomposes its name and sends any non-empty portion as a RADIUS attribute. With the exception of the service-specifics portion of the name, the backslash escaping mechanism is not used in RADIUS attributes; backslash has no special meaning. In the service-specifics portion, a literal "/" separates components. In this one attribute, "/" indicates a slash character that does not separate components and "\\" indicates a literal backslash character.

The initiator **MUST** require that the EAP method in use support channel binding and **MUST** send the acceptor name as part of the channel binding data. The client **MUST NOT** indicate mutual authentication in the result of GSS_Init_Sec_Context unless all name elements that the client supplied are in a successful channel binding response. For example, if the client supplied a hostname in channel binding data, the hostname **MUST** be in a successful channel binding response.

If an empty target name is supplied to GSS_Init_Sec_Context, the initiator **MUST** fail context establishment unless the acceptor supplies the acceptor name response (Section 5.4.3). If a null target name is supplied, the initiator **MUST** use this response to populate EAP channel bindings.

3.5. Proxy Verification of Acceptor Name

Proxies may play a role in verification of the acceptor identity. For example, an AAA proxy near the acceptor may be in a position to verify the acceptor hostname, while the EAP server is likely to be too distant to reliably verify this on its own.

The EAP server or some proxy trusted by the EAP server is likely to be in a position to verify the acceptor realm. In effect, this proxy

is confirming that the right AAA credential is used for the claimed realm and thus that the acceptor is in the organization it claims to be part of. This proxy is also typically trusted by the EAP server to make sure that the hostname claimed by the acceptor is a reasonable hostname for the realm of the acceptor.

A proxy close to the EAP server is unlikely to be in a position to confirm that the acceptor is claiming the correct hostname. Instead this is typically delegated to a proxy near the acceptor. That proxy is typically expected to verify the acceptor hostname and to verify the appropriate AAA credential for that host is used. Such a proxy may insert the acceptor realm if it is absent, permitting realm configuration to be at the proxy boundary rather than on acceptors.

Ultimately specific proxy behavior is a matter for deployment. The EAP server MUST assure that the appropriate validation has been done before including acceptor name attributes in a successful channel binding response. If the acceptor service is included the EAP server asserts that the service is plausible for the acceptor. If the acceptor hostname is included the EAP server asserts that the acceptor hostname is verified. If the realm is included the EAP server asserts that the realm has been verified, and if the hostname was also included, that the realm and hostname are consistent. Part of this verification MAY be delegated to proxies, but the EAP server configuration MUST guarantee that the combination of proxies meets these requirements. Typically such delegation will involve business or operational measures such as cross-organizational agreements as well as technical measures.

It is likely that future technical work will be needed to communicate what verification has been done by proxies along the path. Such technical measures will not release the EAP server from its responsibility to decide whether proxies on the path should be trusted to perform checks delegated to them. However technical measures could prevent misconfigurations and help to support diverse environments.

4. Selection of EAP Method

EAP does not provide a facility for an EAP server to advertise what methods are available to a peer. Instead, a server starts with its preferred method selection. If the peer does not accept that method, the peer sends a NAK response containing the list of methods supported by the client.

Providing multiple facilities to negotiate which security mechanism to use is undesirable. Section 7.3 of [RFC4462] describes the problem referencing the SSH key exchange negotiation and the SPNEGO GSS-API mechanism. If a client preferred an EAP method A, a non-EAP authentication mechanism B, and then an EAP method C, then the client would have to commit to using EAP before learning whether A is actually supported. Such a client might end up using C when B is available.

The standard solution to this problem is to perform all the negotiation at one layer. In this case, rather than defining a single GSS-API mechanism, a family of mechanisms should be defined. Each mechanism corresponds to an EAP method. The EAP method type should be part of the GSS-API OID. Then, a GSS-API rather than EAP facility can be used for negotiation.

Unfortunately, using a family of mechanisms has a number of problems. First, GSS-API assumes that both the initiator and acceptor know the entire set of mechanisms that are available. Some negotiation mechanisms are driven by the client; others are driven by the server. With EAP GSS-API, the acceptor does not know what methods the EAP server implements. The EAP server that is used depends on the identity of the client. The best solution so far is to accept the disadvantages of multi-layer negotiation and commit to using EAP GSS-API before a specific EAP method. This has two main disadvantages. First, authentication may fail when other methods might allow authentication to succeed. Second, a non-optimal security mechanism may be chosen.

5. Context Tokens

All context establishment tokens emitted by the EAP mechanism SHALL have the framing described in section 3.1 of [RFC2743], as illustrated by the following pseudo-ASN.1 structures:

```
GSS-API DEFINITIONS ::=
    BEGIN

    MechType ::= OBJECT IDENTIFIER
    -- representing EAP mechanism
    GSSAPI-Token ::=
    -- option indication (delegation, etc.) indicated within
    -- mechanism-specific token
    [APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech MechType,
        innerToken ANY DEFINED BY thisMech
        -- contents mechanism-specific
        -- ASN.1 structure not required
    }

    END
```

The innerToken field starts with a 16-bit network byte order token type identifier. The remainder of the innerToken field is a set of type-length-value subtokens. The following figure describes the structure of the inner token:

Octet Position	Description
0..1	token ID
2..5	first subtoken type
6..9	length of first subtoken
10..10+n-1	first subtoken body
10+n..10+n+3	second subtoken type

The inner token continues with length, second subtoken body, and so forth. If a subtoken type is present, its length and body MUST be present.

Structure of Inner Token

The length is a four-octet length of the subtoken body in network

byte order. The length does not include the length of the type field or the length field; the length only covers the body.

Tokens from the initiator to acceptor use an inner token type with ID 06 01; tokens from acceptor to initiator use an inner token type with ID 06 02. These token types are registered in the registry of RFC 4121 token types; see Section 7.2.

See Section 5.7 for the encoding of a complete token. The following sections discuss how mechanism OIDs are chosen and the state machine that defines what subtokens are permitted at each point in the context establishment process.

5.1. Mechanisms and Encryption Types

This mechanism family uses the security services of the Kerberos cryptographic framework [RFC3961]. The root of the OID ARC for mechanisms described in this document is 1.3.6.1.5.5.15.1.1; a Kerberos encryption type number [RFC3961] is appended to that root OID to form a mechanism OID. As such, a particular encryption type needs to be chosen. By convention, there is a single object identifier arc for the EAP family of GSS-API mechanisms. A specific mechanism is chosen by adding the numeric Kerberos encryption type number to the root of this arc. However, in order to register the SASL name, the specific usage with a given encryption type needs to be registered. This document defines the EAP-AES128 GSS-API mechanism.

5.2. Processing received tokens

Whenever a context token is received, the receiver performs the following checks. First the receiver confirms the object identifier is that of the mechanism being used. The receiver confirms that the token type corresponds to the role of the peer: acceptors will only process initiator tokens and initiators will only process acceptor tokens.

Implementations of this mechanism maintain a state machine for the context establishment process. Both the initiator and acceptor start out in the initial state; see Section 5.4 for a description of this state. Associated with each state are a set of subtoken types that are processed in that state and rules for processing these subtoken types. The receiver examines the subtokens in order, processing any that are appropriate for the current state. Unknown subtokens or subtokens that are not expected in the current state are ignored if their critical bit (see below) is clear.

A state may have a set of required subtoken types. If a subtoken

type is required by the current state but no subtoken of that type is present, then the context establishment MUST fail.

The most-significant bit (0x80000000) in a subtoken type is the critical bit. If a subtoken with this bit set in the type is received, the receiver MUST fail context establishment unless the subtoken is understood and processed for the current state.

The subtoken type MUST be unique within a given token.

5.3. Error Subtokens

The acceptor may always end the exchange by generating an error subtoken. The error subtoken has the following format:

Pos	Description
0..3	0x80 00 00 01
4..7	length of error token
8..11	major status from RFC 2744 as 32-bit network byte order
12..15	GSS EAP error code as 32-bit network byte order; see Section 7.6

Initiators MUST ignore octets beyond the GSS EAP error code for future extensibility. As indicated, the error token is always marked critical.

5.4. Initial State

Both the acceptor and initiator start the context establishment process in the initial state.

The initiator sends a token to the acceptor. It MAY be empty; no subtokens are required in this state. Alternatively the initiator MAY include a vendor ID subtoken or an acceptor name request subtoken.

The acceptor responds to this message. It MAY include an acceptor name response subtoken. It MUST include a first eap request; this is an EAP request/identity message (see Section 5.5.1 for the format of this subtoken).

The initiator and acceptor then transition to authenticate state.

5.4.1. Vendor Subtoken

The vendor ID token has type 0x0000000B and the following structure:

Pos	Description
0..3	0x0000000B
4..7	length of vendor token
8..8+length	Vendor ID string

The vendor ID string is an UTF-8 string describing the vendor of this implementation. This string is unstructured and for debugging purposes only.

5.4.2. Acceptor Name Request

The acceptor name request token is sent from the initiator to the acceptor indicating that the initiator wishes a particular acceptor name. This is similar to TLS Server Name Indication [RFC6066] which permits a client to indicate which one of a number of virtual services to contact. The structure is as follows:

Pos	Description
0..3	0x00000002
4..7	Length of subtoken
8..n	string form of acceptor name

It is likely that channel binding and thus authentication will fail if the acceptor does not choose a name that is a superset of this name. That is, if a hostname is sent, the acceptor needs to be willing to accept this hostname.

5.4.3. Acceptor Name Response

The acceptor name response subtoken indicates what acceptor name is used. This is useful for example if the initiator supplied no target name to context initialization. This allows the initiator to learn the acceptor name. EAP channel bindings will provide confirmation that the acceptor is accurately naming itself.

this token is sent from the acceptor to initiator. In the Initial state, this token would typically be sent if the acceptor name request is absent, because if the initiator already sent an acceptor name then the initiator knows what acceptor it wishes to contact. This subtoken is also sent in extensions state Section 5.6 so the initiator can protect against a man-in-the-middle modifying the acceptor name request subtoken.

Pos	Description
0..3	0x00000003
4..7	Length of subtoken
8..n	string form of acceptor name

5.5. Authenticate State

In this state, the acceptor sends EAP requests to the initiator and the initiator generates EAP responses. The goal of the state is to perform a successful EAP authentication. Since the acceptor sends an identity request at the end of the initial state, the first half-round-trip in this state is a response to that request from the initiator.

The EAP conversation can end in a number of ways:

- o If the EAP state machine generates an EAP success message, then the EAP authenticator believes the authentication is successful. The Acceptor MUST confirm that a key has been derived (Section 7.10 of [RFC3748]). The acceptor MUST confirm that this success indication is consistent with any protected result indication for combined authenticators and with AAA indication of success for pass-through authenticators. If any of these checks fail, the acceptor MUST send an error subtoken and fail the context establishment. If these checks succeed the acceptor sends the success message using the EAP Request subtoken type and transitions to Extensions state. If the initiator receives an EAP Success message, it confirms that a key has been derived and that the EAP success is consistent with any protected result indication. If so, it transitions to Extensions state. Otherwise, it returns an error to the caller of GSS_Init_Sec_context without producing an output token.
- o If the acceptor receives an EAP failure, then the acceptor sends this in the Eap Request subtoken type. If the initiator receives

an EAP Failure, it returns GSS failure.

- o If there is some other error, the acceptor MAY return an error subtoken.

5.5.1. EAP Request Subtoken

The EAP Request subtoken is sent from the acceptor to the initiator. This subtoken is always critical and is REQUIRED in the authentication state.

Pos	Description
0..3	0x80000005
4..7	Length of EAP message
8..8+length	EAP message

5.5.2. EAP Response Subtoken

This subtoken is REQUIRED in authentication state messages from the initiator to the acceptor. It is always critical.

Pos	Description
0..3	0x80000004
4..7	Length of EAP message
8..8+length	EAP message

5.6. Extension State

After EAP success, the initiator sends a token to the acceptor including additional subtokens that negotiate optional features or provide GSS-API channel binding (see Section 6.1). The acceptor then responds with a token to the initiator. When the acceptor produces its final token it returns GSS_S_COMPLETE; when the initiator consumes this token it returns GSS_S_COMPLETE if no errors are detected.

The acceptor SHOULD send an acceptor name response (Section 5.4.3) so that the initiator can get a copy of the acceptor name protected by

the MIC subtoken.

Both the initiator and acceptor MUST include and verify a MIC subtoken to protect the extensions exchange.

5.6.1. Flags Subtoken

This token is sent to convey initiator flags to the acceptor. The flags are sent as a 32-bit integer in network byte order. The only flag defined so far is GSS_C_MUTUAL_FLAG, indicating that the initiator successfully performed mutual authentication of the acceptor. This flag is communicated to the acceptor because some protocols [RFC4462] require the acceptor to know whether the initiator has confirmed its identity. This flag has the value 0x2 to be consistent with RFC 2744.

Pos	Description
0..3	0x0000000C
4..7	length of flags token
8..11	flags

Initiators MUST send 4 octets of flags. Acceptors MUST ignore flag octets beyond the first 4 and MUST ignore flag bits other than GSS_C_MUTUAL_FLAG. Initiators MUST send undefined flag bits as zero.

5.6.2. GSS Channel Bindings Subtoken

This token is always critical when sent. It is sent from the initiator to the acceptor. The contents of this token are an RFC 3961 get_mic token of the application data from the GSS channel bindings structure passed into the context establishment call.

Pos	Description
0..3	0x80000006
4..7	length of token
8..8+length	get_mic of channel binding application data

Again, only the application data is sent in the channel binding. Any

initiator and acceptor addresses passed by an application into context establishment calls are ignored and not sent over the wire. The checksum type of the `get_mic` token SHOULD be the mandatory to implement checksum type of the Context Root Key (CRK.) The key to use is the CRK and the key usage is 60 (`KEY_USAGE_GSSEAP_CHBIND_MIC`). An acceptor MAY accept any MIC in the channel bindings subtoken if the channel bindings input to `GSS_Accept_Sec_context` is not provided. If the channel binding input to `GSS_Accept_Sec_context` is provided, the acceptor MUST return failure if the channel binding MIC in a received channel binding subtoken fails to verify.

The initiator MUST send this token if channel bindings including application data are passed into `GSS_Init_Sec_context` and MUST NOT send this token otherwise.

5.6.3. MIC Subtoken

This token MUST be the last subtoken in the tokens sent in Extensions state. This token is sent both by the initiator and acceptor.

Pos	Description
0..3	0x8000000D for initiator 0x8000000E for acceptor
4..7	Length of RFC 3961 MIC token
8..8+length	RFC 3961 result of <code>get_mic</code>

As with any call to `get_mic`, a token is produced as described in RFC 3961 using the CRK Section 6 as the key and the mandatory checksum type for the encryption type of the CRK as the checksum type. The key usage is 61 (`KEY_USAGE_GSSEAP_ACCTOKEN_MIC`) for the subtoken from the acceptor to the initiator and 62 (`KEY_USAGE_GSSEAP_INITTOKEN_MIC`) for the subtoken from the initiator to the acceptor. The input is as follows:

1. The DER-encoded object identifier of the mechanism in use; this value starts with 0x06 (the tag for object identifier). When encoded in an RFC 2743 context token, the object identifier is preceeded by the tag and length for [Application 0] SEQUENCE. This tag and the length of the overall token is not included; only the tag, length and value of the object identifier itself.
2. A 16-bit token type in network byte order of the RFC 4121 token identifier (0x0601 for initiator, 0x0602 for acceptor).

3. For each subtoken other than the MIC subtoken itself in the order the subtokens appear in the token:

1. A four octet subtoken type in network byte order
2. A four byte length in network byte order
3. Length octets of value from that subtoken

5.7. Example Token

60	23	06	09	2b	06 01 05 05 0f 01 01 11
App0	Token	OID	OID	1 3	6 1 5 5 15 1 1 17
Tag	length	Tag	length	Mechanism object id	

06 01	00 00 00 02	00 00 00 0e
Initiator	Acceptor	Length
context	name	(14 octets)
token id	request	

68 6f 73 74 2f 6c 6f 63 61 6c 68 6f 73 74
String form of acceptor name
"host/localhost"

Example Initiator Token

5.8. Context Options

GSS-API provides a number of optional per-context services requested by flags on the call to `GSS_Init_sec_context` and indicated as outputs from both `GSS_Init_sec_context` and `GSS_Accept_sec_context`. This section describes how these services are handled. Which services the client selects in the call to `GSS_Init_sec_context` controls what EAP methods MAY be used by the client. Section 7.2 of RFC 3748 describes a set of security claims for EAP. As described below, the selected GSS options place requirements on security claims that MUST be met.

This GSS mechanism MUST only be used with EAP methods that provide dictionary attack resistance. Typically dictionary attack resistance is obtained by using an EAP tunnel method to tunnel an inner method in TLS.

The EAP method MUST support key derivation. Integrity, confidentiality, sequencing and replay detection MUST be indicated in the output of GSS_Init_Sec_Context and GSS_Accept_Sec_context regardless of which services are requested.

The PROT_READY service defined in Section 1.2.7 of [RFC2743] is never available with this mechanism. Implementations MUST NOT offer this flag or permit per-message security services to be used before context establishment.

The EAP method MUST support mutual authentication and channel binding. See Section 3.4 for details on what is required for successful mutual authentication. Regardless of whether mutual authentication is requested, the implementation MUST include channel bindings in the EAP authentication. If mutual authentication is requested and successful mutual authentication takes place as defined in Section 3.4, the initiator MUST send a flags subtoken Section 5.6.1 in Extensions state.

6. Acceptor Services

The context establishment process may be passed through to a EAP server via a backend authentication protocol. However after the EAP authentication succeeds, security services are provided directly by the acceptor.

This mechanism uses an RFC 3961 cryptographic key called the context root key (CRK). The CRK is derived from the GSK (GSS-API MSK). The GSK is the result of the random-to-key [RFC3961] operation of the encryption type of this mechanism consuming the appropriate number of bits from the EAP master session key. For example for aes128-cts-hmac-sha1-96, the random-to-key operation consumes 16 octets of key material; thus the first 16 bytes of the master session key are input to random-to-key to form the GSK. If the MSK is too short, authentication MUST fail.

In the following, pseudo-random is the RFC 3961 pseudo-random operation for the encryption type of the GSK and random-to-key is the RFC 3961 random-to-key operation for the enctype of the mechanism. The truncate function takes the initial 1 bits of its input. The goal in constructing a CRK is to call the pseudo-random function enough times to produce the right number of bits of output and discard any excess bits of output.

The CRK is derived from the GSK using the following procedure

```
Tn = pseudo-random(GSK, n || "rfc4121-gss-eap")
CRK = random-to-key(truncate(L, T0 || T1 || .. || Tn))
L = random-to-key input size
```

Where n is a 32-bit integer in network byte order starting at 0 and incremented to each call to the pseudo_random operation.

6.1. GSS-API Channel Binding

GSS-API channel binding [RFC5554] is a protected facility for exchanging a cryptographic name for an enclosing channel between the initiator and acceptor. The initiator sends channel binding data and the acceptor confirms that channel binding data has been checked.

The acceptor SHOULD accept any channel binding provided by the initiator if null channel bindings are passed into gss_accept_sec_context. Protocols such as HTTP Negotiate [RFC4559] depend on this behavior of some Kerberos implementations.

As discussed, the GSS channel bindings subtoken is sent in the extensions state.

6.2. Per-message security

The per-message tokens of section 4 of RFC 4121 are used. The CRK SHALL be treated as the initiator sub-session key, the acceptor sub-session key and the ticket session key.

6.3. Pseudo Random Function

The pseudo random function defined in [RFC4402] is used to provide GSS_Pseudo_Random functionality to applications.

7. Iana Considerations

This specification creates a number of IANA registries.

7.1. OID Registry

IANA is requested to create a registry of ABFAB object identifiers titled "Object Identifiers for Application Bridging for federated Access". The initial contents of the registry are specified below. The registration policy is IETF review or IESG approval. Early allocation is permitted. IANA is requested to update the reference for the root of this OID delegation to point to the newly created registry.

Prefix: iso.org.dod.internet.security.mechanisms.abfab (1.3.6.1.5.5.15)

Decimal	Name	Description	References
0	Reserved	Reserved	
1	mechanisms	A sub-arc containing ABFAB mechanisms	
2	nametypes	A sub-arc containing ABFAB GSS-API Name Types	

NOTE: the following mechanisms registry are the root of the OID for the mechanism in question. As discussed in Section 5.1 [draft-ietf-abbfab-gss-eap], a Kerberos encryption type number [RFC3961] is appended to the mechanism version OID below to form the OID of a specific mechanism.

Prefix: iso.org.dod.internet.security.mechanisms.abfab.mechanisms
(1.3.6.1.5.5.15.1)

Decimal	Name	Description	References
0	Reserved	Reserved	
1	gss-eap-v1	The GSS-EAP mechanism	[this spec

Prefix: iso.org.dod.internet.security.mechanisms.abfab.nametypes
(1.3.6.1.5.5.15.2)

Decimal	Name	Description	References
0	Reserved	Reserved	
1	GSS_EAP_NT_EAP_NAME		sect 3.1

7.2. RFC 4121 Token Identifiers

In the top level registry titled "Kerberos V GSS-API Mechanism Parameters," a sub-registry called "Kerberos GSS-API Token Type Identifiers" is created; the overall reference for this subregistry is section 4.1 of RFC 4121. The allocation procedure is expert review [RFC5226]. The expert's primary job is to make sure that token type identifiers are requested by an appropriate requester for the RFC 4121 mechanism in which they will be used and that multiple values are not allocated for the same purpose. For RFC 4121 and this mechanism, the expert is currently expected to make allocations for token identifiers from documents in the IETF stream; effectively for these mechanisms the expert currently confirms the allocation meets the requirements of the IETF review process.

The ID field is a hexadecimal token identifier specified in network byte order.

The initial registrations are as follows:

ID	Description	Reference
01 00	KRB_AP_REQ	RFC 4121 sect 4.1
02 00	KRB_AP_REP	RFC 4121 sect 4.1
03 00	KRB_ERROR	RFC 4121 sect 4.1
04 04	MIC tokens	RFC 4121 sect 4.2.6.1
05 04	wrap tokens	RFC 4121 sect 4.2.6.2
06 01	GSS-EAP initiator context token	Section 5
06 02	GSS EAP acceptor context token	Section 5

7.3. GSS EAP Subtoken Types

This document creates a top level registry called "The Extensible Authentication Protocol Mechanism for the Generic Security Services Application Programming Interface (GSS-EAP) Parameters". In any short form of that name, including any URI for this registry, it is important that the string GSS come before the string EAP; this will help to distinguish registries if EAP methods for performing GSS-API authentication are ever defined.

In this registry is a subregistry of subtoken types; identifiers are 32-bit integers; the upper bit (0x80000000) is reserved as a critical flag and should not be indicated in the registration. Assignments of GSS EAP subtoken types are made by expert review. The expert is expected to require a public specification of the subtoken similar in detail to registrations given in this document. The security of GSS-EAP depends on making sure that subtoken information has adequate protection and that the overall mechanism continues to be secure. Examining the security and architectural consistency of the proposed registration is the primary responsibility of the expert.

Type	Description	Reference
0x00000001	Error	Section 5.3
0x0000000B	Vendor	Section 5.4.1
0x00000002	Acceptor name request	Section 5.4.2
0x00000003	Acceptor name response	Section 5.4.3
0x00000005	EAP request	Section 5.5.1
0x00000004	EAP response	Section 5.5.2
0x0000000C	Flags	Section 5.6.1
0x00000006	GSS-API channel bindings	Section 5.6.2
0x0000000D	Initiator MIC	Section 5.6.3
0x0000000E	Acceptor MIC	Section 5.6.3

7.4. RADIUS Attribute Assignments

The following RADIUS attribute type values [RFC3575] are assigned. The assignment rules in section 10.3 of [I-D.ietf-radext-radius-extensions] may be used if that specification is approved when IANA actions for this specification are processed.

Name	Attribute	Description
GSS-Acceptor-Service-Name	TBD1	user-or-service portion of name
GSS-Acceptor-Host-Name	TBD2	host portion of name
GSS-Acceptor-Service-specifics	TBD3	service-specifics portion of name
GSS-Acceptor-Realm-Name	TBD4	Realm portion of name

7.5. Registration of the EAP-AES128 SASL Mechanisms

Subject: Registration of SASL mechanisms
EAP-AES128 and EAP-AES128-PLUS

SASL mechanism names: EAP-AES128 and EAP-AES128-PLUS

Security considerations: See RFC 5801 and draft-ietf-abfab-gss-eap

Published specification (recommended): draft-ietf-abfab-gss-eap

Person & email address to contact for further information:
Abfab Working Group abfab@ietf.org

Intended usage: common

Owner/Change controller: iesg@ietf.org

Note: This mechanism describes the GSS-EAP mechanism used with the aes128-cts-hmac-shal-96 enctype. The GSS-API OID for this mechanism is 1.3.6.1.5.5.15.1.1.17

As described in RFC 5801 a PLUS variant of this mechanism is also required.

7.6. GSS EAP Errors

A new subregistry is created in the GSS EAP parameters registry titled "Error Codes". The error codes in this registry are unsigned 32-bit numbers. Values less than or equal to 127 are assigned by standards action. Values 128 through 255 are assigned with the specification required assignment policy. Values greater than 255 are reserved; updates to registration policy may make these values available for assignment and implementations MUST be prepared to

receive them.

This table provides the initial contents of the registry.

Value	Description
0	Reserved
1	Buffer is incorrect size
2	Incorrect mechanism OID
3	Token is corrupted
4	Token is truncated
5	Packet received by direction that sent it
6	Incorrect token type identifier
7	Unhandled critical subtoken received
8	Missing required subtoken
9	Duplicate subtoken type
10	Received unexpected subtoken for current state xxx
11	EAP did not produce a key
12	EAP key too short
13	Authentication rejected
14	AAA returned an unexpected message type
15	AAA response did not include EAP request
16	Generic AAA failure

7.7. GSS EAP Context Flags

A new sub-registry is created in the GSS EAP parameters registry. This registry holds registrations of flag bits sent in the flags subtoken Section 5.6.1. There are 32 flag bits available for registration represented as hexadecimal numbers from the most-

significant bit 0x80000000 to the least significant bit 0x1. The registration policy for this registry is IETF review or in exceptional cases IESG approval. The following table indicates initial registrations; all other values are available for assignment.

Flag	Name	Reference
0x2	GSS_C_MUTUAL_FLAG	Section 5.6.1

8. Security Considerations

RFC 3748 discusses security issues surrounding EAP. RFC 5247 discusses the security and requirements surrounding key management that leverages the AAA infrastructure. These documents are critical to the security analysis of this mechanism.

RFC 2743 discusses generic security considerations for the GSS-API. RFC 4121 discusses security issues surrounding the specific per-message services used in this mechanism.

As discussed in Section 4, this mechanism may introduce multiple layers of security negotiation into application protocols. Multiple layer negotiations are vulnerable to a bid-down attack when a mechanism negotiated at the outer layer is preferred to some but not all mechanisms negotiated at the inner layer; see section 7.3 of [RFC4462] for an example. One possible approach to mitigate this attack is to construct security policy such that the preference for all mechanisms negotiated in the inner layer falls between preferences for two outer layer mechanisms or falls at one end of the overall ranked preferences including both the inner and outer layer. Another approach is to only use this mechanism when it has specifically been selected for a given service. The second approach is likely to be common in practice because one common deployment will involve an EAP supplicant interacting with a user to select a given identity. Only when an identity is successfully chosen by the user will this mechanism be attempted.

EAP channel binding is used to give the GSS-API initiator confidence in the identity of the GSS-API acceptor. Thus, the security of this mechanism depends on the use and verification of EAP channel binding. Today EAP channel binding is in very limited deployment. If EAP channel binding is not used, then the system may be vulnerable to phishing attacks where a user is diverted from one service to another. If the EAP method in question supports mutual authentication then users can only be diverted between servers that are part of the same AAA infrastructure. For deployments where membership in the AAA infrastructure is limited, this may serve as a significant limitation on the value of phishing as an attack. For other deployments, use of EAP channel binding is critical to avoid phishing. These attacks are possible with EAP today although not typically with common GSS-API mechanisms. For this reason, implementations are required to implement and use EAP channel binding; see Section 3 for details.

The security considerations of EAP channel binding [I-D.ietf-emu-chbind] describe the security properties of channel binding. Two attacks are worth calling out here. First, when a

tunneled EAP method is used, it is critical that the channel binding be performed with an EAP server trusted by the peer. With existing EAP methods this typically requires validating the certificate of the server tunnel endpoint back to a trust anchor and confirming the name of the entity who is a subject of that certificate. EAP methods may suffer from bid-down attacks where an attacker can cause a peer to think that a particular EAP server does not support channel binding. This does not directly cause a problem because mutual authentication is only offered at the GSS-API level when channel binding to the server's identity is successful. However when an EAP method is not vulnerable to these bid-down attacks, additional protection is available. This mechanism will benefit significantly from new strong EAP methods such as [I-D.ietf-emu-eap-tunnel-method].

Every proxy in the AAA chain from the authenticator to the EAP server needs to be trusted to help verify channel bindings and to protect the integrity of key material. GSS-API applications may be built to assume a trust model where the acceptor is directly responsible for authentication. However, GSS-API is definitely used with trusted-third-party mechanisms such as Kerberos.

RADIUS does provide a weak form of hop-by-hop confidentiality of key material based on using MD5 as a stream cipher. Diameter can use TLS or IPsec but has no mandatory-to-implement confidentiality mechanism. Operationally, protecting key material as it is transported between the IDP and RP is critical to per-message security and verification of GSS-API channel binding [RFC5056]. Mechanisms such as RADIUS over TLS [I-D.ietf-radext-radsec] provide significantly better protection of key material than the base RADIUS specification.

9. Acknowledgements

Luke Howard, Jim Schaad, Alejandro Perez Mendez, Alexey Melnikov and Sujing Zhou provided valuable reviews of this document.

Rhys Smith provided the text for the OID registry section. Sam Hartman's work on this draft has been funded by JANET.

10. References

10.1. Normative References

[GSS-IANA]

IANA, "GSS-API Service Name Registry", <<http://www.iana.org/assignments/gssapi-service-names/gssapi-service-names.xhtml>>.

[I-D.ietf-abfab-eapapplicability]

Winter, S. and J. Salowey, "Update to the EAP Applicability Statement for ABFAB", draft-ietf-abfab-eapapplicability-00 (work in progress), July 2012.

[I-D.ietf-emu-chbind]

Hartman, S., Clancy, T., and K. Hoeper, "Channel Binding Support for EAP Methods", draft-ietf-emu-chbind-16 (work in progress), May 2012.

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

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

[RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, January 2000.

[RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

- [RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.
- [RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 4402, February 2006.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

10.2. Informative References

- [I-D.ietf-abfab-arch]
Howlett, J., Hartman, S., Tschofenig, H., Lear, E., and J. Schaad, "Application Bridging for Federated Access Beyond Web (ABFAB) Architecture", draft-ietf-abfab-arch-03 (work in progress), July 2012.
- [I-D.ietf-emu-eap-tunnel-method]
Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel EAP Method (TEAP) Version 1", draft-ietf-emu-eap-tunnel-method-03 (work in progress), June 2012.
- [I-D.ietf-krb-wg-gss-cb-hash-agility]
Emery, S., "Kerberos Version 5 GSS-API Channel Binding Hash Agility", draft-ietf-krb-wg-gss-cb-hash-agility-10 (work in progress), January 2012.
- [I-D.ietf-radext-radius-extensions]
DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", draft-ietf-radext-radius-extensions-06 (work in progress), June 2012.
- [I-D.ietf-radext-radsec]

- Wierenga, K., McCauley, M., Winter, S., and S. Venaas, "Transport Layer Security (TLS) encryption for RADIUS", draft-ietf-radext-radsec-12 (work in progress), February 2012.
- [RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC4072] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.
- [RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005.
- [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.
- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.
- [RFC5178] Williams, N. and A. Melnikov, "Generic Security Service Application Program Interface (GSS-API) Internationalization and Domain-Based Service Names and Name Type", RFC 5178, May 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

Appendix A. Pre-Publication RADIUS VSA

As described in Section 3.4, RADIUS attributes are used to carry the acceptor name when this family of mechanisms is used with RADIUS. Prior to publication of this specification, a vendor-specific RADIUS attribute was used. This non-normative appendix documents that attribute as it may be seen from older implementations.

Prior to IANA assignment, GSS-EAP used a RADIUS vendor-specific attribute for carrying the acceptor name. The VSA with enterprise ID 25622 is formatted as a VSA according to the recommendation in the RADIUS specification. The following sub-attributes are defined:

Name	Attribute	Description
GSS-Acceptor-Service-Name	128	user-or-service portion of name
GSS-Acceptor-Host-Name	129	host portion of name
GSS-Acceptor-Service-specifics	130	service-specifics portion of name
GSS-Acceptor-Realm-Name	131	Realm portion of name

Authors' Addresses

Sam Hartman (editor)
Painless Security

Email: hartmans-ietf@mit.edu

Josh Howlett
JANET

Email: josh.howlett@ja.net

Kitten Working Group
Internet-Draft
Obsoletes: RFC 2831 (if approved)
(if approved)
Intended status: Informational
Expires: October 24, 2011

A. Melnikov
Isode Limited
April 22, 2011

Moving DIGEST-MD5 to Historic
draft-ietf-kitten-digest-to-historic-04

Abstract

This memo describes problems with the DIGEST-MD5 Simple Authentication and Security Layer (SASL) mechanism as specified in RFC 2831. It marks DIGEST-MD5 as OBSOLETE in the IANA Registry of SASL mechanisms, and moves RFC 2831 to Historic. status.

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 October 24, 2011.

Copyright Notice

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

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

described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Overview	3
2.	Security Considerations	5
3.	IANA Considerations	5
4.	Acknowledgements	5
5.	References	6
5.1.	Normative References	6
5.2.	Informative References	6
	Author's Address	7

1. Overview

[RFC2831] defined how HTTP Digest Authentication [RFC2617] can be used as a Simple Authentication and Security Layer (SASL) [RFC4422] mechanism for any protocol that has a SASL profile. It was intended both as an improvement over CRAM-MD5 [RFC2195] and as a convenient way to support a single authentication mechanism for web, email, LDAP, and other protocols. While it can be argued that it was an improvement over CRAM-MD5, many implementors commented that the additional complexity of DIGEST-MD5 made it difficult to implement fully and securely.

Below is an incomplete list of problems with DIGEST-MD5 mechanism as specified in RFC 2831:

1. The mechanism had too many options and modes. Some of them were not well described and were not widely implemented. For example, DIGEST-MD5 allowed the "qop" directive to contain multiple values, but it also allowed for multiple qop directives to be specified. The handling of multiple options was not specified, which resulted in minor interoperability problems. Some implementations amalgamated multiple qop values into one, while others treated multiple qops as an error. Another example is the use of an empty authorization identity. In SASL an empty authorization identity means that the client is willing to authorize as the authentication identity. The document was not clear on whether the authzid must be omitted or can be specified with the empty value to convey this. The requirement for backward compatibility with HTTP Digest meant that the situation was even worse. For example DIGEST-MD5 required all usernames/passwords which can be entirely represented in ISO-8859-1 charset to be down converted from UTF-8 to ISO-8859-1. Another example is use of quoted strings. Handling of characters that needed escaping was not properly described and the DIGEST-MD5 document had no examples to demonstrate correct behavior.
2. The document used ABNF from RFC 822 [RFC0822], which allows an extra construct and allows for "implied folding whitespace" to be inserted in many places. The difference from ABNF [RFC5234] was confusing for some implementors. As a result, many implementations didn't accept folding whitespace in many places where it was allowed.
3. The DIGEST-MD5 document uses the concept of a "realm" to define a collection of accounts. A DIGEST-MD5 server can support one or more realms. The DIGEST-MD5 document didn't provide any guidance on how realms should be named, and, more importantly, how they can be entered in User Interfaces (UIs). As the result many

DIGEST-MD5 clients had confusing UIs, didn't allow users to enter a realm and/or didn't allow users to pick one of the server supported realms.

4. Use of username in the inner hash. The inner hash of DIGEST-MD5 is an MD5 hash of colon separated username, realm and password. Implementations may choose to store inner hashes instead of clear text passwords. While this has some useful properties, such as protection from compromise of authentication databases containing the same username and password on other servers, if a server with the username and password is compromised, however this was rarely done in practice. Firstly, the inner hash is not compatible with widely deployed Unix password databases, and second, changing the username would invalidate the inner hash.
5. Description of DES/3DES [DES] and RC4 security layers are inadequate to produce independently-developed interoperable implementations. In the DES/3DES case this was partly a problem with existing DES APIs.
6. DIGEST-MD5 outer hash (the value of the "response" directive) didn't protect the whole authentication exchange, which made the mechanism vulnerable to "man in the middle" (MITM) attacks, such as modification of the list of supported qops or ciphers.
7. The following features are missing from DIGEST-MD5, which make it insecure or unsuitable for use in protocols:
 - A. Lack of channel bindings [RFC5056].
 - B. Lack of hash agility (i.e. no easy way to replace the MD5 hash function with another one).
 - C. Lack of support for SASLPrep [RFC4013] or any other type of Unicode character normalization of usernames and passwords. The original DIGEST-MD5 document predates SASLPrep and doesn't recommend any Unicode character normalization.
8. The cryptographic primitives in DIGEST-MD5 are not up to today's standards, in particular:
 - A. The MD5 hash is sufficiently weak to make a brute force attack on DIGEST-MD5 easy with common hardware [RFC6151].
 - B. Using the RC4 algorithm for the security layer without discarding the initial key stream output is prone to attack [RC4].

- C. The DES cipher for the security layer is considered insecure due to its small key space [RFC3766].

Note that most of the problems listed above are already present in the HTTP Digest authentication mechanism.

Because DIGEST-MD5 was defined as an extensible mechanism, it would be possible to fix most of the problems listed above. However this would increase implementation complexity of an already complex mechanism even further, so the effort would not be worth the cost. In addition, an implementation of a "fixed" DIGEST-MD5 specification would likely either not interoperate with any existing implementation of RFC 2831, or would be vulnerable to various downgrade attacks.

Note that despite DIGEST-MD5 seeing some deployment on the Internet, this specification recommends obsoleting DIGEST-MD5 because DIGEST-MD5, as implemented, is not a reasonable candidate for further standardization and should be deprecated in favor of one or more new password-based mechanisms currently being designed.

The SCRAM family of SASL mechanisms [RFC5802] has been developed to provide similar features as DIGEST-MD5 but with a better design.

2. Security Considerations

Security issues are discussed through out this document.

3. IANA Considerations

IANA is requested to change the "Intended usage" of the DIGEST-MD5 mechanism registration in the SASL mechanism registry to OBSOLETE. The SASL mechanism registry is specified in [RFC4422] and is currently available at:

<http://www.iana.org/assignments/sasl-mechanisms>

4. Acknowledgements

The author gratefully acknowledges the feedback provided by Chris Newman, Simon Josefsson, Kurt Zeilenga, Sean Turner and Abhijit Menon-Sen. Various text was copied from other RFCs, in particular from RFC 2831.

5. References

5.1. Normative References

- [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.
- [RFC2831] Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000.

5.2. Informative References

- [DES] National Institute of Standards and Technology, "Data Encryption Standard (DES)", FIPS PUB 46-3, October 1999.
- [RC4] Strombergson, J. and S. Josefsson, "Test vectors for the stream cipher RC4", draft-josefsson-rc4-test-vectors-02.txt (work in progress), June 2010.
- [RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.
- [RFC2195] Klensin, J., Catoe, R., and P. Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.
- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

Author's Address

Alexey Melnikov
Isode Limited
5 Castle Business Village
36 Station Road
Hampton, Middlesex TW12 2BX
UK

Email: Alexey.Melnikov@isode.com
URI: <http://www.melnikov.ca/>

NETWORK WORKING GROUP
Internet-Draft
Intended status: Standards Track
Expires: October 1, 2017

N. Williams
Cryptonector LLC
A. Melnikov
Isode Ltd
March 30, 2017

Namespace Considerations and Registries for GSS-API Extensions
draft-ietf-kitten-gssapi-extensions-iana-11.txt

Abstract

This document describes the ways in which the GSS-API may be extended and directs the creation of an IANA registry for various GSS-API namespaces.

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 October 1, 2017.

Copyright Notice

Copyright (c) 2017 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. Conventions used in this document	2
2. Introduction	2
3. Extensions to the GSS-API	2
4. Generic GSS-API Namespaces	3
5. Language Binding-Specific GSS-API Namespaces	3
6. Extension-Specific GSS-API Namespaces	4
7. Registration Form	4
8. IANA Considerations	6
8.1. Initial Namespace Registrations	7
8.1.1. Example registrations	7
8.2. Registration Maintenance Guidelines	9
8.2.1. Sub-Namespace Symbol Pattern Matching	10
8.2.2. Expert Reviews of Individual Submissions	10
8.2.3. Change Control	11
9. Security Considerations	12
10. References	12
10.1. Normative References	12
10.2. Informative References	12
Authors' Addresses	13

1. Conventions used in this document

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 [RFC2119].

2. Introduction

There is a need for private-use and mechanism-specific extensions to the Generic Security Services Application Programming Interface (GSS-API). As such extensions are designed and standardized (or not), both at the IETF and elsewhere, there is a non-trivial risk of namespace pollution and conflicts. To avoid this we set out guidelines for extending the GSS-API and direct the creation of an IANA registry for GSS-API namespaces.

Registrations of individual items and sub-namespaces are allowed. Each sub-namespace may provide different rules for registration, e.g., for mechanism-specific and private-use extensions.

3. Extensions to the GSS-API

Extensions to the GSS-API can be categorized as follows:

- o Abstract API extensions

- o Implementation-specific
- o Mechanism-specific
- o Language binding-specific

Extensions to the GSS-API may be purely semantic, without effect on the GSS-API's namespaces. Or they may introduce new functions, constants, types, etc...; these clearly affect the GSS-API namespaces.

Extensions that affect the GSS-API namespaces should be registered with the IANA as described herein.

4. Generic GSS-API Namespaces

The abstract API namespaces for the GSS-API are:

- o Type names
- o Function names
- o Constant names for various types
- o Constant values for various types
- o Name types (OID, type name and syntaxes)

Additionally we have namespaces associates with the OBJECT IDENTIFIER (OID) type. The IANA already maintains a registry of such OIDs:

- o Mechanism OIDs
- o Name Type OIDs

5. Language Binding-Specific GSS-API Namespaces

Language binding specific namespaces include, among others:

- o Header/interface module names
- o Object classes and/or types
- o Methods and/or functions
- o Constant names
- o Constant values

6. Extension-Specific GSS-API Namespaces

Extensions to the GSS-API may create additional namespaces. See Section 8.2.

7. Registration Form

Registrations for GSS-API namespaces SHALL take the following form:

Registration Field	Possible Values	Description
Bindings	'Generic', 'C-bindings', 'Java', 'C#', <programming language name>	Indicates the name of the programming language that this registration involves, or, if 'Generic', that this is an entry for the generic abstract GSS-API (i.e., not specific to any programming language).
Registration type	'Instance', 'Sub-Namespace'	Indicates whether this entry reserves a given symbol name (and possibly, constant value), or whether it reserves an entire sub-namespace (the name is a pattern) or constant value range.
Object Type	<Symbol> defined by the binding language (for example 'Data-Type', 'Function', 'Method', 'Integer', 'String', 'OID', 'Context-Flag', 'Name-Type', 'Macro', 'Header-File-Name', 'Module-Name', 'Class')	Indicates the type of the object whose symbolic name or constant value this entry registers. The possible values of this field depend on the programming language in question, therefore they are not all specified here.
Symbol Name/Prefix	<Symbol name or name pattern>	The name of a symbol or symbol sub-namespace being

		registered. See Section 8.2.1
Binding of	<Name of abstract API element of which this object is a binding>	If the registration is for a specific language binding of the GSS-API, then this names the abstract API element of which it is a binding (OPTIONAL).
Constant Value/Range	<Constant value> or <constant value range>	The value of the constant named by the <Symbol Name/Prefix>. This field is present only for Instance and Sub-namespace registrations of Constant object types.
Description	<Text>	Description of the registration. Multiple instances of this field may result (see Section 8.2.3).
Registration Rules	<Reference> to an IANA registration Policy defined in [RFC5226] (or an RFC that updates it), for instance 'IESG Approval', 'Expert Review', 'First Come First Served', 'Private Use'.	Describes the rules for allocation of items that fall in this sub-namespace, for entries with Registration Type of Sub-namespace (OPTIONAL). For private use sub-namespaces the submitter MUST provide the e-mail address of a responsible contact. If this field is not specified for a sub-namespace, the default registration rules specified in Section 8.2 apply.
Reference	<Reference>	Reference to a document that describes the registration, if any (OPTIONAL). Multiple instances of this field are allowed, with one reference each.
Expert Reviewer	<Name of expert reviewers, possibly	OPTIONAL, see Section 8.2.2. Multiple instances of this

	WG names>	field are allowed, with one expert reviewer per-instance. Leave this field blank when requesting a registration. It will be filled in by the Expert who reviews the registration.
Expert Review Notes	<Notes from the expert review>	Expert reviewers may request that some comments be included with the registration, e.g., regarding security considerations of the registered extension.
Status	'Registered' or 'Obsoleted'	Status of the registration.
Obsoleting Reference	<Reference>	Reference to a document, if any, that obsoletes this registration. Multiple instances of this field are allowed, with one reference each. (OPTIONAL)

The IANA should create a single GSS-API namespace registry, or multiple registries, one for symbolic names and one for constant values, and/or it may create a registry per-programming language, at its convenience.

Entries in these registries should consist of all the fields from their corresponding registration entries.

Entries should be sorted by: programming language, registration type, object type, and symbol name/pattern.

8. IANA Considerations

This document deals with IANA considerations throughout. Specifically it creates a single registry of various kinds of things, though the IANA may instead create multiple registries, each for one of those kinds of things. Of particular interest may be that IANA will now be the registration authority for the GSS-API name type OID space.

8.1. Initial Namespace Registrations

Initial registry content corresponding to the items defined in [RFC2743], [RFC2744], [RFC2853], [RFC1964] and [RFC4121] and others will be supplied during the IANA review portion of the RFC publishing process. [[Note to RFC Editor: Delete the following sentence before publication:]] The KITTEN WG chairs MUST indicate that such content has been reviewed by the WG and that there is WG consensus that the entries are in agreement with those RFCs.

8.1.1. Example registrations

In order to sanity check recommended IANA registration templates, this section registers several entries.

Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Function
Symbol Name	gss_init_sec_context
Binding of	GSS_Init_sec_context
Constant Value/Range	N/A
Description	Create a security context by initiator
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A

Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Function
Symbol Name	gss_accept_sec_context
Binding of	GSS_Accept_sec_context
Constant Value/Range	N/A
Description	Accept a security context from initiator
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A

Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Context-Flag
Symbol Name	GSS_C_DELEG_FLAG
Binding of	deleg_state or deleg_req_flag
Constant Value/Range	1
Description	On output (if set): Delegated credentials are available via the delegated_cred_handle parameter of GSS_Accept_sec_context. On input (if set): With the call to GSS_Init_sec_context, delegate credentials to the acceptor.
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A

8.2. Registration Maintenance Guidelines

Standards-Track RFCs can create new items with any non-conflicting Symbol Name/Prefix value for this registry by virtue of IESG approval to publish as a Standards-Track RFC -- that is, without additional expert review.

Standards-Track RFCs can mark existing entries as obsolete, and can even create conflicting entries if explicitly stated (the IESG, of course, should review conflicts carefully, and may reject them).

IANA shall also consider submissions from individuals, and via Informational and Experimental RFCs, subject to Expert Review. IANA SHALL allow such registrations if a) they are not conflicting, b) provided that the registration is for object types other than Context-Flags, and c) subject to expert review. Guidelines for expert reviews are given below.

8.2.1. Sub-Namespace Symbol Pattern Matching

Sub-namespace registrations must provide a pattern for matching symbols for which the sub-namespace's registration rules apply. The pattern consists of a string with the following special tokens:

- o `'*'`, meaning "match any string."
- o `%m`, meaning "match any mechanism family short-hand name."
- o `%i`, meaning "match any implementor vanity short-hand name."

For example, `"GSS_%m_*` matches `"GSS_krb5_foo"` since `"krb5"` is a common short-hand for the Kerberos V GSS-API mechanism [RFC1964]. But `"GSS_%m_*` does not match `"GSS_foo_bar"` unless `"foo"` is asserted to be a short-hand for some mechanism.

8.2.2. Expert Reviews of Individual Submissions

[[The following paragraph should be deleted from the document before publication, as it will not age well. It should be moved to the shepherding write-up.]]

Expert review selection SHALL be done as follows. If, at the time that the IANA receives an individual submission for registration in this registry, there are any IETF Working Groups chartered to produce GSS-API-related documents, then the IANA SHALL ask the chairs of such WGs to be expert reviewers or to name one. If there are no such WGs at that time, then the IANA SHALL ask past chairs of the KITTEN WG and the author/editor of this RFC to act as expert reviewers or name an alternate.

Expert reviewers of individual registration submissions with Registration Type == Sub-namespace should check that the registration request has a suitable description (which doesn't need to be sufficiently detailed for others to implement) and that the Symbol Name/Prefix is sufficiently descriptive of the purpose of the sub-namespace or reflective of the name of the submitter or associated company.

Expert reviewers of individual registration submissions with

Registration Type == Instance should check that the Symbol Name falls under a sub-namespace controlled by the submitter. Registration of such entries which do not fall under such a sub-namespace may be allowed provided that they correspond to long existing non-standard extensions to the GSS-API and this can be easily checked or demonstrated, otherwise IESG Protocol Action is REQUIRED (see previous section). Also, reviewers should check that any registration of constant values have a detailed description that is suitable for other implementors to reproduce, and that they don't conflict with other usages or are otherwise dangerous in the reviewers estimation.

Expert reviewers should review impact on mechanisms, security and interoperability, and may reject or annotate registrations which can have mechanism impact that requires IESG protocol action. Consider, for example, new versions of `GSS_Init_sec_context()` and/or `GSS_Accept_sec_context` which have new input and/or output parameters which imply changes on the wire or in behaviour that may result in interoperability issues. A reviewer could choose to add notes to the registration describing such issues, or the reviewer might conclude that the danger to Internet interoperability is sufficient to warrant rejecting the registration.

8.2.3. Change Control

Registered entries may be marked obsoleted using the same expert review process as for registering new entries. Obsoleted entries are not, however, to be deleted, but merely marked having Obsoleted Status. Note that entries may be created as obsoleted to record the fact that the given symbol(s) have been used before, even though continued use of them is discouraged.

Registered entries may also be updated in two other ways: additional references, obsoleting references, and descriptions may be added.

All changes are subject to expert review, except for changes to registrations in a sub-namespace which are subject to the rules of the relevant sub-namespace. The submitter of a change request need not be the same as the original submitter.

Registrations may be modified by addition, but under no circumstance may any fields be modified except for the Status field or Contact Address, or to correct for transcription errors in filing or processing registration requests.

The IANA SHALL add a field describing the date that a an addition or modification was made, and a description of the change.

9. Security Considerations

General security considerations relating to IANA registration services apply; see [RFC5226].

Also, expert reviewers should look for and may document security related issues with submitters' GSS-API extensions, to the best of the reviewers' ability given the information furnished by the submitter. Reviewers may add comments regarding their limited ability to review a submission for security problems if the submitter is unwilling to provide sufficient documentation.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <<http://www.rfc-editor.org/info/rfc2743>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

10.2. Informative References

- [RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, DOI 10.17487/RFC1964, June 1996, <<http://www.rfc-editor.org/info/rfc1964>>.
- [RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, DOI 10.17487/RFC2744, January 2000, <<http://www.rfc-editor.org/info/rfc2744>>.
- [RFC2853] Kabat, J. and M. Upadhyay, "Generic Security Service API Version 2 : Java Bindings", RFC 2853, DOI 10.17487/RFC2853, June 2000, <<http://www.rfc-editor.org/info/rfc2853>>.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, DOI 10.17487/RFC4121, July 2005, <<http://www.rfc-editor.org/info/rfc4121>>.

Authors' Addresses

Nicolas Williams
Cryptonector LLC

Email: nico@cryptonector.com

Alexey Melnikov
Isode Ltd
5 Castle Business Village
36 Station Road
Hampton, Middlesex TW12 2BX
UK

Email: Alexey.Melnikov@isode.com

KITTEN WORKING GROUP
Internet-Draft
Intended status: Standards Track
Expires: December 2, 2012

N. Williams
Cryptonector, LLC
L. Johansson
SUNET
S. Hartman
Painless Security
S. Josefsson
SJD AB
May 31, 2012

GSS-API Naming Extensions
draft-ietf-kitten-gssapi-naming-exts-15

Abstract

The Generic Security Services API (GSS-API) provides a simple naming architecture that supports name-based authorization. This document introduces new APIs that extend the GSS-API naming model to support name attribute transfer between GSS-API peers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on December 2, 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Conventions used in this document	4
2.	Introduction	4
3.	Name Attribute Authenticity	5
4.	Name Attributes/Values as ACL Subjects	5
5.	Naming Contexts	5
6.	Representation of Attribute Names	7
7.	API	8
7.1.	SET OF OCTET STRING	8
7.2.	Const types	8
7.3.	GSS_Display_name_ext()	9
7.3.1.	C-Bindings	9
7.4.	GSS_Inquire_name()	10
7.4.1.	C-Bindings	10
7.5.	GSS_Get_name_attribute()	11
7.5.1.	C-Bindings	12
7.6.	GSS_Set_name_attribute()	12
7.6.1.	C-Bindings	14
7.7.	GSS_Delete_name_attribute()	14
7.7.1.	C-Bindings	15
7.8.	GSS_Export_name_composite()	15
7.8.1.	C-Bindings	16
8.	IANA Considerations	16
9.	Security Considerations	16
10.	References	17
10.1.	Normative References	17
10.2.	Informative References	17
	Authors' Addresses	18

1. Conventions used in this document

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 [RFC2119] .

2. Introduction

As described in [RFC4768] the GSS-API's naming architecture suffers from certain limitations. This document defines concrete GSS-API extensions.

A number of extensions to the GSS-API [RFC2743] and its C Bindings [RFC2744] are described herein. The goal is to make information modeled as "name attributes" available to applications. Such information MAY for instance be used by applications to make authorization-decisions. For example, Kerberos V authorization data elements, both in their raw forms, as well as mapped to more useful value types, can be made available to GSS-API applications through these interfaces.

The model is that GSS names have attributes. The attributes of a name may be authenticated (e.g., an X509 attribute certificate or signed SAML attribute assertion), or may have been set on a GSS name for the purpose of locally "asserting" the attribute during credential acquisition or security context exchange. Name attributes' values are network representations thereof (e.g., the actual value octets of the contents of an X.509 certificate extension, for example) and are intended to be useful for constructing portable access control facilities. Applications may often require language- or platform-specific data types, rather than network representations of name attributes, so a function is provided to obtain objects of such types associated with names and name attributes.

Future updates of this specification may involve adding an attribute namespace for attributes that only have application-specific semantics. Note that mechanisms will still need to know how to transport such attributes. The IETF may also wish to add functions by which to inquire whether a mechanism(s) understands a given attribute name or namespace, and to list which attributes or attribute namespaces a mechanism understands. Finally, the IETF may want to consider adding a function by which to determine the name of the issuer of a name attribute.

3. Name Attribute Authenticity

An attribute is 'authenticated' if and only if there is a secure association between the attribute (and its values) and the trusted source of the peer credential. Examples of authenticated attributes are (any part of) the signed portion of an X.509 certificate or AD-KDCIssued authorization-data elements in Kerberos V Tickets provided of course that the authenticity of the respective security associations (e.g., signatures) have been verified.

Note that the fact that an attribute is authenticated does not imply anything about the semantics of the attribute nor that the trusted credential source was authorized to assert the attribute. Such interpretations SHOULD be the result of applying local policy to the attribute.

An un-authenticated attribute is called _asserted_ in what follows. This is not to be confused with other uses of the word asserted or assertion such as "SAML attribute assertion", the attributes of which may be authenticated in the sense of this document for instance if the SAML attribute assertion was signed by a key trusted by the peer.

4. Name Attributes/Values as ACL Subjects

To facilitate the development of portable applications that make use of name attributes to construct and evaluate portable ACLs the GSS-API makes name attribute values available in canonical network encodings thereof.

5. Naming Contexts

Several factors influence the context in which a name attribute is interpreted. One is the trust context.

As discussed previously, applications apply local policy to determine whether a particular peer credential issuer is trusted to make a given statement. Different GSS-API mechanisms and deployments have different trust models surrounding attributes they provide about a name.

For example, Kerberos deployments in the enterprise typically trust a KDC to make any statement about principals in a realm. This includes attributes such as group membership.

In contrast, in a federated SAML environment, the identity provider typically exists in a different organization than the acceptor. In this case, the set of group memberships or entitlements that the IDP is permitted to make needs to be filtered by the policy of the

acceptor and federation.

So even an attribute containing the same information such as e-mail address would need to be treated differently by the application in the context of an enterprise deployment from the context of a federation.

Another aspect related to trust is the role of the credential issuer in providing the attribute. Consider Kerberos PKINIT [RFC4556]. In this protocol, a public key and associated certificate are used to authenticate to a Kerberos KDC. Consider how attributes related to a pkinit certificate should be made available in GSS-API authentications based on the Kerberos ticket. In some deployments the certificate may be fully trusted; in including the certificate information in the ticket, the KDC permits the acceptor to trust the information in the certificate just as if the KDC itself had made these statements. In other deployments, the KDC may have authorized a hash of the certificate without evaluating the content of the certificate or generally trusting the issuing certification authority. In this case, if the certificate were included in the issued ticket, the KDC would only be making the statement that the certificate was used in the authentication. This statement would be authenticated, but would not imply that the KDC stated particular attributes of the certificate described the initiator.

Another aspect of context is encoding of the attribute information. An attribute containing an ASCII [ANSI.X3-4.1986] or UTF-8 [RFC3629] version of an e-mail address could not be interpreted the same as a ASN.1 Distinguished Encoding Rules e-mail address in a certificate.

All of these contextual aspects of a name attribute affect whether two attributes can be treated the same by an application and thus whether they should be considered the same name attribute. In the GSS-API naming extensions, attributes that have different contexts MUST have different names so they can be distinguished by applications. As an unfortunate consequence of this requirement, multiple attribute names will exist for the same basic information. That is, there is no single attribute name for the e-mail address of an initiator. Other aspects of how mechanisms describe information about subjects would already make this true. For example, some mechanisms use OIDs to name attributes; others use URIs.

Local implementations or platforms are likely to have sufficient policy and information to know when contexts can be treated as the same. For example the GSS-API implementation may know that a particular certification authority can be trusted in the context of a pkinit authentication. The local implementation may have sufficient policy to know that a particular credential issuer is trusted to make

a given statement. In order to take advantage of this local knowledge within the GSS-API implementation, naming extensions support the concept of local attributes in addition to standard attributes. For example, an implementation might provide a local attribute for e-mail address. The implementation would specify the encoding and representation of this attribute; mechanism-specific standards attributes would be re-encoded if necessary to meet this representation. Only e-mail addresses in contexts that meet the requirements of local policy would be mapped into this local attribute.

Such local attributes inherently expose a tradeoff between interoperability and usability. Using a local attribute in an application requires knowledge of the local implementation. However using a standardized attribute in an application requires more knowledge of policy and more validation logic in the application. Sharing this logic in the local platform provides more consistency across applications as well as reducing implementation costs. Both options are needed.

6. Representation of Attribute Names

Different underlying mechanisms (e.g., SAML or X.509 certificates) provide different representations for the names of their attribute. In X.509 certificates, most objects are named by object identifiers (OIDs). The type of object (certificate extension, name constraint, keyPurposeID, etc) along with the OID is sufficient to identify the attribute. By contrast, according to Section 8.2 and 2.7.3.1 of [OASIS.saml-core-2.0-os], the name of an attribute has two parts. The first is a URI describing the format of the name. The second part, whose form depends on the format URI, is the actual name. In other cases an attribute might represent a certificate that plays some particular role in a GSS-API mechanism; such attributes might have a simple mechanism-defined name.

Attribute names MUST support multiple components. If there are more than one component in an attribute name, the more significant components define the semantics of the less significant components.

Attribute names are represented as OCTET STRING elements in the API described below. These attribute names have syntax and semantics that are understood by the application and by the lower-layer implementations (some of which are described below).

If an attribute name contains a space (ASCII 0x20), the first space separates the most significant or primary component of the name from the remainder. We may refer to the primary component of the

attribute name as the attribute name's "prefix". If there is no space, the primary component is the entire name, otherwise it defines the interpretation of the remainder of the name.s

If the primary component contains an ASCII : (0x3a), then the primary component is a URI. Otherwise, the attribute is a local attribute and the primary component has meaning to the implementation of GSS-API or to the specific configuration of the application. Local attribute names with an at-sign ('@') in them are reserved for future allocation by the IETF.

Since attribute names are split at the first space into prefix and suffix, there is a potential for ambiguity if a mechanism blindly passes through a name attribute whose name it does not understand. In order to prevent such ambiguities the mechanism MUST always prefix raw name attributes with a prefix that reflects the context of the attribute.

Local attribute names under the control of an administrator or a sufficiently trusted part of the platform need not have a prefix to describe context.

7. API

7.1. SET OF OCTET STRING

The construct SET OF OCTET STRING occurs once in RFC 2743 [RFC2743] where it is used to represent a set of status strings in the GSS_Display_status call. The Global Grid Forum has defined SET OF OCTET STRING as a buffer-set type in GFD.024 [GFD.024] which also provides one API for memory management of these structures. The normative reference to GFD.024 [GFD.024] is for the buffer set functions defined in section 2.5 and the associated buffer set C types defined in section 6 (namely gss_buffer_set_desc, gss_buffer_set_t, gss_create_empty_buffer_set, gss_add_buffer_set_member, gss_release_buffer_set). Nothing else from GFD.024 is required to implement this document. In particular, that document specify changes in behaviour existing GSS-API functions in section 3: implementing those changes are not required to implement this document. Any implementation of SET OF OCTET STRING for use by this specification MUST preserve order.

7.2. Const types

The C bindings for the new APIs uses some types from [RFC5587] to avoid issues with the use of "const". The normative reference to [RFC5587] is for the C types specified in Figure 1 of 3.4.6, nothing

else from that document is required to implement this document.

7.3. GSS_Display_name_ext()

Inputs:

- o name INTERNAL NAME,
- o display_as_name_type OBJECT IDENTIFIER

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o display_name OCTET STRING -- caller must release with GSS_Release_buffer()

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.
- o GSS_S_UNAVAILABLE indicates that the given name could not be displayed using the syntax of the given name type.
- o GSS_S_FAILURE indicates a general error.

This function displays a given name using the given name syntax, if possible. This operation may require mapping Mechanism Names (MNs) to generic name syntaxes or generic name syntaxes to mechanism-specific name syntaxes; such mappings may not always be feasible and MAY be inexact or lossy, therefore this function may fail.

7.3.1. C-Bindings

The display_name buffer is de-allocated by the caller with gss_release_buffer.

```
OM_uint32 gss_display_name_ext(
    OM_uint32                *minor_status,
    gss_const_name_t          name,
    gss_const_OID             display_as_name_type,
    gss_buffer_t              display_name
);
```

7.4. GSS_Inquire_name()

Inputs:

- o name INTERNAL NAME

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o name_is_MN BOOLEAN,
- o mn_mech OBJECT IDENTIFIER,
- o attrs SET OF OCTET STRING -- the caller is responsible for de-allocating memory using GSS_Release_buffer_set

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.
- o GSS_S_FAILURE indicates a general error.

This function outputs the set of attributes of a name. It also indicates if a given name is an Mechanism Name (MN) or not and, if it is, what mechanism it's an MN of.

7.4.1. C-Bindings

```
OM_uint32 gss_inquire_name(  
    OM_uint32                *minor_status,  
    gss_const_name_t         name,  
    int                      *name_is_MN,  
    gss_OID                  *MN_mech,  
    gss_buffer_set_t         *attrs  
);
```

The gss_buffer_set_t is used here as the C representation of SET OF OCTET STRING. This type is used to represent a set of attributes and is a NULL-terminated array of gss_buffer_t. The gss_buffer_set_t type and associated API is defined in GFD.024 [GFD.024]. The "attrs" buffer set is de-allocated by the caller using gss_release_buffer_set().

7.5. GSS_Get_name_attribute()

Inputs:

- o name INTERNAL NAME,
- o attr OCTET STRING

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o authenticated BOOLEAN, -- TRUE if and only if authenticated by the trusted peer credential source.
- o complete BOOLEAN -- TRUE if and only if this represents a complete set of values for the name.
- o values SET OF OCTET STRING -- the caller is responsible for de-allocating memory using GSS_Release_buffer_set.
- o display_values SET OF OCTET STRING -- the caller is responsible for de-allocating memory using GSS_Release_buffer_set

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.
- o GSS_S_UNAVAILABLE indicates that the given attribute OID is not known or set.
- o GSS_S_FAILURE indicates a general error.

This function outputs the value(s) associated with a given GSS name object for a given name attribute.

The complete flag denotes that (if TRUE) the set of values represents a complete set of values for this name. The peer being an authoritative source of information for this attribute is a sufficient condition for the complete flag to be set by the peer.

In the federated case when several peers may hold some of the attributes about a name this flag may be highly dangerous and SHOULD NOT be used.

NOTE: This function relies on the GSS-API notion of "SET OF" allowing for order preservation; this has been discussed on the KITTEN WG mailing list and the consensus seems to be that, indeed, that was always the intention. It should be noted however that the order presented does not always reflect an underlying order of the mechanism specific source of the attribute values.

7.5.1. C-Bindings

The C-bindings of `GSS_Get_name_attribute()` requires one function call per-attribute value, for multi-valued name attributes. This is done by using a single `gss_buffer_t` for each value and an input/output integer parameter to distinguish initial and subsequent calls and to indicate when all values have been obtained.

The 'more' input/output parameter should point to an integer variable whose value, on first call to `gss_get_name_attribute()` MUST be -1, and whose value upon function call return will be non-zero to indicate that additional values remain, or zero to indicate that no values remain. The caller should not modify this parameter after the initial call. The status of the complete and authenticated flags MUST NOT change between multiple calls to iterate over values for an attribute.

The output buffers "value" and "display_value" are de-allocated by the caller using `gss_release_buffer()`.

```
OM_uint32 gss_get_name_attribute(
    OM_uint32                *minor_status,
    gss_const_name_t         name,
    gss_const_buffer_t       attr,
    int                      *authenticated,
    int                      *complete,
    gss_buffer_t             value,
    gss_buffer_t             display_value,
    int                      *more
);
```

7.6. GSS_Set_name_attribute()

Inputs:

- o name INTERNAL NAME,
- o complete BOOLEAN, -- TRUE if and only if this represents a complete set of values for the name.

- o attr OCTET STRING,
- o values SET OF OCTET STRING

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.
- o GSS_S_UNAVAILABLE indicates that the given attribute NAME is not known or could not be set.
- o GSS_S_FAILURE indicates a general error.

When the given NAME object is an MN this function MUST fail (with GSS_S_FAILURE) if the mechanism for which the name is an MN does not recognize the attribute name or the namespace it belongs to. This is because name attributes generally have some semantics that mechanisms must understand.

On the other hand, when the given name is not an MN this function MAY succeed even if none of the available mechanisms understand the given attribute, in which subsequent credential acquisition attempts (via GSS_Acquire_cred() or GSS_Add_cred()) with the resulting name MUST fail for mechanisms that do not understand any one or more name attributes set with this function. Applications may wish to use a non-MN, then acquire a credential with that name as the desired name. The acquired credentials will have elements only for the mechanisms that can carry the name attributes set on the name.

Note that this means that all name attributes are locally critical: the mechanism(s) must understand them. The reason for this is that name attributes must necessarily have some meaning that the mechanism must understand, even in the case of application-specific attributes (in which case the mechanism must know to transport the attribute to any peer). However, there is no provision to ensure that peers understand any given name attribute. Individual name attributes may be critical with respect to peers, and the specification of the attribute will have to indicate which of the mechanism's protocol or the application is expected to enforce criticality.

The complete flag denotes that (if TRUE) the set of values represents

a complete set of values for this name. The peer being an authoritative source of information for this attribute is a sufficient condition for the complete flag to be set by the peer.

In the federated case when several peers may hold some of the attributes about a name this flag may be highly dangerous and SHOULD NOT be used.

NOTE: This function relies on the GSS-API notion of "SET OF" allowing for order preservation; this has been discussed on the KITTEN WG mailing list and the consensus seems to be that, indeed, that was always the intention. It should be noted that underlying mechanisms may not respect the given order.

7.6.1. C-Bindings

The C-bindings of `GSS_Set_name_attribute()` requires one function call per-attribute value, for multi-valued name attributes -- each call adds one value. To replace an attribute's every value delete the attribute's values first with `GSS_Delete_name_attribute()`.

```
OM_uint32 gss_set_name_attribute(  
    OM_uint32                *minor_status,  
    gss_const_name_t         name,  
    int                      complete,  
    gss_const_buffer_t       attr,  
    gss_const_buffer_t       value  
);
```

7.7. `GSS_Delete_name_attribute()`

Inputs:

- o name INTERNAL NAME,
- o attr OCTET STRING,

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.

- o GSS_S_UNAVAILABLE indicates that the given attribute NAME is not known.
- o GSS_S_UNAUTHORIZED indicates that a forbidden delete operation was attempted, such as deleting a negative attribute.
- o GSS_S_FAILURE indicates a general error.

Deletion of negative authenticated attributes from NAME objects MUST NOT be allowed and must result in a GSS_S_UNAUTHORIZED.

7.7.1. C-Bindings

```
OM_uint32 gss_delete_name_attribute(  
    OM_uint32                *minor_status,  
    gss_const_name_t         name,  
    gss_const_buffer_t       attr  
);
```

7.8. GSS_Export_name_composite()

Inputs:

- o name INTERNAL NAME

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o exp_composite_name OCTET STRING -- the caller is responsible for de-allocating memory using GSS_Release_buffer

Return major_status codes:

- o GSS_S_COMPLETE indicates no error.
- o GSS_S_FAILURE indicates a general error.

This function outputs a token which can be imported with GSS_Import_name(), using GSS_C_NT_COMPOSITE_EXPORT as the name type and which preserves any name attribute information (including the authenticated/complete flags) associated with the input name (which GSS_Export_name() may well not). The token format is not specified here as this facility is intended for inter-process communication

only; however, all such tokens MUST start with a two-octet token ID, hex 04 02, in network byte order.

The OID for GSS_C_NT_COMPOSITE_EXPORT is <TBD>.

7.8.1. C-Bindings

The "exp_composite_name" buffer is de-allocated by the caller with gss_release_buffer.

```
OM_uint32 gss_export_name_composite(  
    OM_uint32                *minor_status,  
    gss_const_name_t         name,  
    gss_buffer_t              exp_composite_name  
);
```

8. IANA Considerations

This specification has no actions for IANA.

This document creates a namespace of GSS-API name attributes. Attributes are named by URIs, so no single authority is technically needed for allocation. However future deployment experience may indicate the need for an IANA registry for URIs used to reference names specified by IETF standards. It is expected that this will be a registry of URNs but this document provides no further guidance on this registry.

9. Security Considerations

This document extends the GSS-API naming model to include support for name attributes. The intention is that name attributes are to be used as a basis for (among other things) authorization decisions or personalization for applications relying on GSS-API security contexts.

The security of the application may be critically dependent on the security of the attributes. This document classifies attributes as asserted or authenticated. Asserted (non-authenticated) attributes MUST NOT be used if the attribute has security implications for the application (e.g., authorization decisions) since asserted attributes may easily be controlled by the peer directly.

It is important to understand the meaning of 'authenticated' in this setting. Authenticated does not imply that any semantic of the attribute is claimed to be true. The only implication is that a

trusted third party has asserted the attribute as opposed to the attribute being asserted by the peer itself. Any additional semantics are always the result of applying policy. For instance in a given deployment the mail attribute of the subject may be authenticated and sourced from an email system where 'authoritative' values are kept. In another situation users may be allowed to modify their mail addresses freely. In both cases the 'mail' attribute may be authenticated by virtue of being included in signed SAML attribute assertions or by other means authenticated by the underlying mechanism.

When the underlying security mechanism does not provide a permanent unique identity (e.g., anonymous kerberos), GSS-API naming extensions may be used to provide a permanent unique identity attribute. This may be a globally unique identifier, a value unique within the namespace of the attribute issuer, or a "directed" identifier that is unique per peer acceptor identity. SAML, to use one example technology, offers a number of built-in constructs for this purpose, such as a <NameID> with a Format of "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent". SAML deployments also typically make use of domain-specific attribute types that can serve as identifiers.

10. References

10.1. Normative References

- [GFD.024] Argonne National Laboratory, National Center for Supercomputing Applications, Argonne National Laboratory, and Argonne National Laboratory, "GSS-API Extensions", GFD GFD.024, June 2004.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, January 2000.
- [RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.

10.2. Informative References

- [ANSI.X3-4.1986]

American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[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.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.

[RFC4768] Hartman, S., "Desired Enhancements to Generic Security Services Application Program Interface (GSS-API) Version 3 Naming", RFC 4768, December 2006.

Authors' Addresses

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com

Leif Johansson
Swedish University Network
Thulegatan 11
Stockholm
Sweden

Email: leifj@sunet.se
URI: <http://www.sunet.se>

Sam Hartman
Painless Security

Phone:
Fax:
Email: hartmans-ietf@mit.edu
URI:

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2012

E. Lear
Cisco Systems GmbH
H. Tschofenig
Nokia Siemens Networks
H. Mauldin
Cisco Systems, Inc.
S. Josefsson
SJD AB
February 24, 2012

A SASL & GSS-API Mechanism for OpenID
draft-ietf-kitten-sasl-openid-08

Abstract

OpenID 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 and GSS-API mechanism for OpenID that allows the integration of existing OpenID 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 August 25, 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	2
1.1. Terminology	4
1.2. Applicability	4
2. Applicability for application protocols other than HTTP	4
2.1. Binding SASL to OpenID in the Relying Party	7
2.2. Discussion	7
3. OpenID SASL Mechanism Specification	8
3.1. Initiation	9
3.2. Authentication Request	9
3.3. Server Response	9
3.4. Error Handling	10
4. OpenID GSS-API Mechanism Specification	10
4.1. GSS-API Principal Name Types for OpenID	11
5. Example	12
6. Security Considerations	13
6.1. Binding OpenIDs to Authorization Identities	14
6.2. RP redirected by malicious URL to take an improper action	14
6.3. User Privacy	14
7. IANA Considerations	14
8. Acknowledgments	15
9. References	15
9.1. Normative References	15
9.2. Informative References	16
Appendix A. Changes	17
Authors' Addresses	17

1. Introduction

OpenID [OpenID] is a web-based three-party protocol that provides a means for a user to offer identity assertions and other attributes to a web server (Relying Party) via the help of an identity provider. The purpose of this system is to provide a way to verify that an end user controls an identifier.

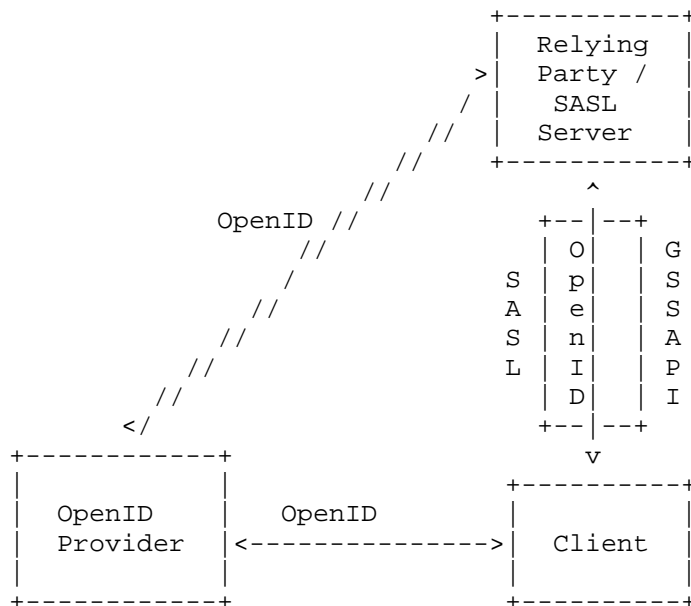
Simple Authentication and Security Layer (SASL) [RFC4422] (SASL) is used by application protocols such as IMAP [RFC3501], POP [RFC1939] and XMPP [RFC6120], with the goal of modularizing authentication and security layers, so that newer mechanisms can be added as needed. This memo specifies just such 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 interface. This document defines a pure SASL mechanism for OpenID, 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. Implementors of the SASL component MAY implement the GSS-API interface as well.

This mechanism specifies interworking between SASL and OpenID in order to assert identity and other attributes to relying parties. As such, while SASL servers (as relying parties) will advertise SASL mechanisms, clients will select the OpenID mechanism.

The OpenID mechanism described in this memo aims to re-use the OpenID mechanism to the maximum extent 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) [RFC5246], continue to be used. Minimal changes are required to non-web applications, as most of the transaction occurs through a normal web browser. Hence, this specification is only appropriate for use when such a browser is available.

Figure 1 describes the interworking between OpenID and SASL. This document requires enhancements to the Relying Party and to the Client (as the two SASL communication end points) but no changes to the OpenID Provider (OP) are necessary. To accomplish this goal indirect messaging required by the OpenID specification is tunneled through the SASL/GSS-API mechanism.



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 OpenID 2.0 specification.

1.2. Applicability

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

2. Applicability for application protocols other than HTTP

OpenID was originally envisioned for HTTP [RFC2616] and HTML [W3C .REC-html401-19991224] based communications, and with the associated semantic, the idea being that the user would be redirected by the Relying Party to an identity provider who authenticates the user, and then sends identity information and other attributes (either directly or indirectly) to the Relying Party. The identity provider in the OpenID specifications is referred to as an OpenID Provider (OP). The actual protocol flow can be found in Section 3 of the OpenID 2.0

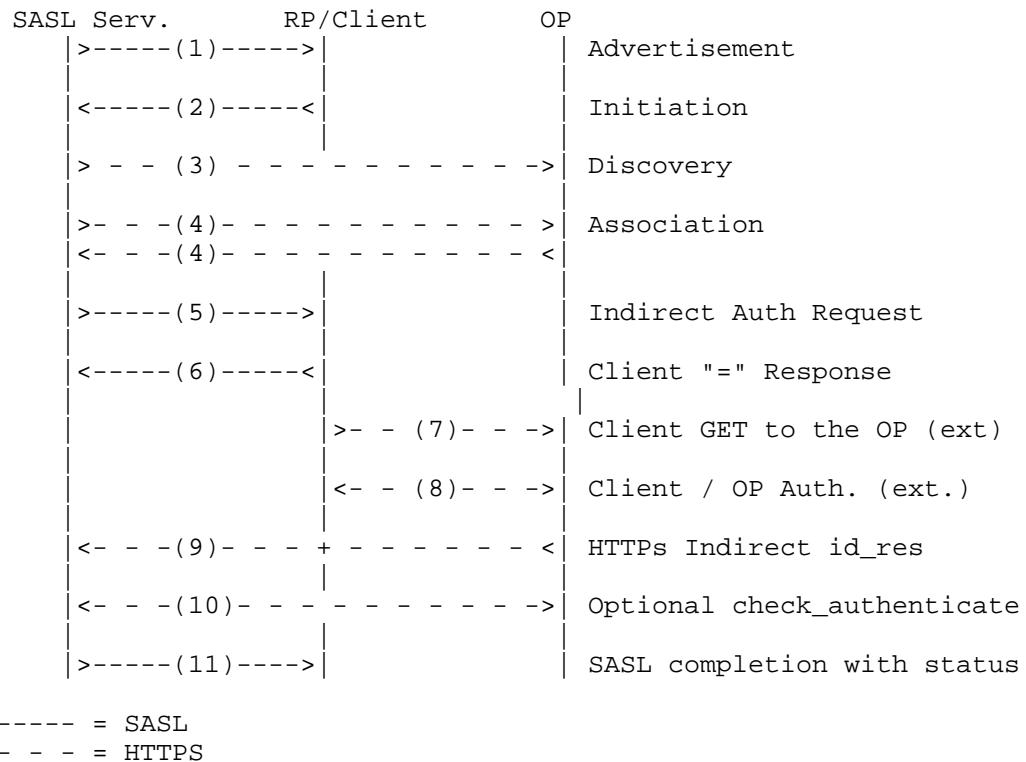
specification [OpenID]. The reader is strongly encouraged to be familiar with the specification before continuing.

When considering that flow in the context of SASL, we note that while the RP and the client both need to change their code to implement this SASL mechanism, it is a design constraint that the OP behavior remain untouched, in order for implementations to interoperate with existing IdPs. Hence, an analog flow that interfaces the three parties needs to be created. In the analog, we note that unlike a web server, the SASL server already has some sort of session (probably a TCP connection) established with the client. However, it may be necessary for a SASL client to invoke to another application. This will be discussed below. By doing so, we externalize much of the authentication from SASL.

The steps are listed below:

1. The SASL server advertises support for the SASL OpenID mechanism to the client.
2. The client initiates a SASL authentication and transmits the User-Supplied Identifier as its first response. The SASL mechanism is client-first, and as explained in [RFC4422] the server will send an empty challenge if needed.
3. After normalizing the User-Supplied Identifier as discussed in [OpenID], the Relying Party performs discovery on it and establishes the OP Endpoint URL that the end user uses for authentication.
4. The Relying Party and the OP optionally establish an association -- a shared secret established using Diffie-Hellman Key Exchange. The OP uses an association to validate those messages through the use of an HMAC; this removes the need for subsequent direct requests to verify the signature after each authentication request/response.
5. The Relying Party transmits an authentication request to the OP to obtain an assertion in the form of an indirect request. These messages are passed through the client rather than directly between the RP and the OP. OpenID defines two methods for indirect communication, namely HTTP redirects and HTML form submission. Both mechanisms are not directly applicable for usage with SASL. To ensure that a standard OpenID 2.0 capable OP can be used a new method is defined in this document that requires the OpenID message content to be encoded using a Universal Resource Identifier (URI). [RFC3986] Note that any Internationalized Resource Identifiers (IRIs) must be normalized to URIs by the SASL client, as specified in [RFC3987], prior to transmitting them to the SASL server.
6. The SASL client now sends an response consisting of "=", to indicate that authentication continues via the normal OpenID flow.

7. At this point the client application **MUST** construct a URL containing the content received in the previous message from the RP. This URL is transmitted to the OP either by the SASL client application or an appropriate handler, such as a browser.
8. Next the client optionally authenticates to the OP and then approves or disapproves authentication to the Relying Party. For reasons of its own the OP has the option of not authenticating a request. The manner in which the end user is authenticated to their respective OP and any policies surrounding such authentication is out of scope of OpenID and hence also out of scope for this specification. This step happens out of band from SASL.
9. The OP will convey information about the success or failure of the authentication phase back to the RP, again using an indirect response via the client browser or handler. The client transmits over HTTP/TLS the redirect of the OP result to the RP. This step happens out of band from SASL.
10. The RP **MAY** send an OpenID check_authentication request directly to the OP, if no association has been established, and the OP should respond. Again this step happens out of band from SASL.
11. The SASL server sends an appropriate SASL response to the client, with optional Open Simple Registry (SREG) attributes.



Note the directionality in SASL is such that the client MUST send the "=" response. Specifically, the SASL client processes the redirect and then awaits a final SASL decision, while the rest of the OpenID authentication process continues.

2.1. Binding SASL to OpenID in the Relying Party

OpenID is meant to be used in serial within the web, where browser cookies are easily accessible. As such, there are no transaction-ids within the protocol. To ensure that a specific request is bound, and in particular to ease interprocess communication, the relying party MUST encode a nonce or transaction-id in the URIs it transmits through the client for success or failure, either as a base URI or fragment component to the "return_to" URI. This value is to be used to uniquely identify each authentication transaction. The nonce value MUST be at least 2^{32} large and large enough to handle well in excess of the number of concurrent transactions a SASL server shall see.

2.2. Discussion

As mentioned above OpenID is primarily designed to interact with web-based applications. Portions of the authentication stream are only defined in the crudest sense. That is, when one is prompted to approve or disapprove an authentication, anything that one might find on a browser is allowed, including JavaScript, fancy style-sheets, etc. Because of this lack of structure, implementations will need to invoke a fairly rich browser in order to ensure that the authentication can be completed.

Once there is an outcome, the SASL server needs to know about it. The astute will hopefully by now have noticed an "=" client SASL response. This is not to say that nothing is happening, but rather that authentication flow has shifted from SASL and the client application to OpenID within the browser, and will return to the client application when the server has an outcome to hand to the client. The alternative to this flow would be some sort of signal from the HTML browser to the SASL client of the results that would in turn be passed to the SASL server. The inter-process communication issue this raises is substantial. Better, we conclude, to externalize the authentication to the browser, and have an "=" client response.

3. OpenID SASL Mechanism Specification

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

The name of this mechanism "OPENID20". 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, described below as "outcome_data" (with SREG attributes), sent as additional data when indicating a successful outcome.

3.1. Initiation

A client initiates an OpenID authentication with SASL by sending the GS2 header followed by the URI, as specified in the OpenID specification.

```
initial-response = gs2-header Auth-Identifier
Auth-Identifier = Identifier ; authentication identifier
Identifier = URI           ; Identifier is specified in
                           ; Sec. 7.2 of the OpenID 2.0 spec.
```

The syntax and semantics of the "gs2-header" are specified in [RFC5801], and we use it here with the following limitations: The "gs2-nonstd-flag" MUST NOT be present. The "gs2-cb-flag" MUST be "n" because channel binding is not supported by this mechanism.

URI is specified in [RFC3986]. XRIs MUST NOT be used. [XRI2.0]

3.2. Authentication Request

The SASL Server sends the URL resulting from the OpenID authentication request, containing an "openid.mode" of either "checkid_immediate" or "checkid_setup", as specified in Section 9.1 of the OpenID 2.0 specification.

```
authentication-request = URI
```

As part of this request, the SASL server MUST append a unique transaction id to the "return_to" portion of the request. The form of this transaction is left to the RP to decide, but SHOULD be large enough to be resistant to being guessed or attacked.

The client now sends that request via an HTTP GET to the OP, as if redirected to do so from an HTTP server.

The client MUST handle both user authentication to the OP and confirmation or rejection of the authentication by the RP via this SASL mechanism.

After all authentication has been completed by the OP, and after the response has been sent to the client, the client will relay the response to the Relying Party via HTTP/TLS, as specified previously in the transaction ("return_to").

3.3. Server Response

The Relying Party now validates the response it received from the client via HTTP/TLS, as specified in the OpenID specification, using the "return_to" URI given previously in the transaction.

The response by the Relying Party constitutes a SASL mechanism outcome, and SHALL be used to set state in the server accordingly, and it SHALL be used by the server to report that state to the SASL client as described in [RFC4422] Section 3.6. In the additional data, the server MAY include OpenID Simple Registry (SREG) attributes that are listed in Section 4 of [SREG1.0]. SREG attributes are encoded as follows:

1. Strip "openid.sreg." from each attribute name.
2. Treat the concatenation of results as URI parameters that are separated by an ampersand (&) and encode as one would a URI, absent the scheme, authority, and the question mark.

For example: email=lear@example.com&fullname=Eliot%20Lear

More formally:

```
outcome-data = [ sreg-avp *( "," sreg-avp ) ]
sreg-avp      = sreg-attr "=" sreg-val
sreg-attr     = sreg-word
sreg-val      = sreg-word
sreg-word     = 1*( unreserved / pct-encoded )
               ; pct-encoded from Section 2.1 of RFC 3986
               ; unreserved from Section 2.3 of RFC 3986
```

A client who does not support SREG MUST ignore SREG attributes sent by the server. Similarly, a client MUST ignore unknown attributes.

In the case of failures, the response MUST follow this syntax:

```
outcome_data = "openid.error" "=" sreg_val *( "," sregp_avp )
```

3.4. Error Handling

[RFC4422] Section 3.6 explicitly prohibits additional information in an unsuccessful authentication outcome. Therefore, the openid.error and openid.error_code are to be sent as an additional challenge in the event of an unsuccessful outcome. In this case, as the protocol is lock step, the client will follow with an additional exchange containing "=", after which the server will respond with an application-level outcome.

4. OpenID 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 OpenID SASL mechanism is actually also a GSS-API mechanism. The OpenID user takes the role of the GSS-API Initiator and the OpenID Relying Party takes the role of the GSS-API Acceptor. The OpenID Provider does not have a role in GSS-API, and is considered an internal matter for the OpenID mechanism. The messages are the same, but a) the GS2 header on the client's first message and channel binding data is excluded when OpenID 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 OpenID is OID-TBD (IANA to assign: see IANA considerations).

OpenID security contexts MUST have the `mutual_state` flag (`GSS_C_MUTUAL_FLAG`) set to TRUE. OpenID does not support credential delegation, therefore OpenID 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 OpenID mechanism does not support per-message tokens or `GSS_Pseudo_random`.

The [RFC5587] mechanism attributes for this mechanism are `GSS_C_MA_MECH_CONCRETE`, `GSS_C_MA_ITOK_FRAMED`, and `GSS_C_MA_AUTH_INIT`.

4.1. GSS-API Principal Name Types for OpenID

OpenID supports standard generic name syntaxes for acceptors such as `GSS_C_NT_HOSTBASED_SERVICE` (see [RFC2743], Section 4.1).

OpenID supports only a single name type for initiators: `GSS_C_NT_USER_NAME`. `GSS_C_NT_USER_NAME` is the default name type for OpenID.

OpenID name normalization is covered by the OpenID specification, see [OpenID] section 7.2.

The query, display, and exported name syntaxes for OpenID principal names are all the same. There are no OpenID-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 OpenID name normalization rules. For example, the OpenID identifier "https://openid.example/" will have a GSS_C_NT_USER_NAME value of "https://openid.example/".

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

5. Example

Suppose one has an OpenID of https://openid.example, and wishes to authenticate his IMAP connection to mail.example (where .example is the top level domain specified in [RFC2606]). The user would input his Openid into his mail user agent, when he configures the account. In this case, no association is attempted between the OpenID RP and the OP. The client will make use of the return_to attribute to capture results of the authentication to be redirected to the server. Note the use of [RFC4959] for initial response. The authentication on the wire would then look something like the following:


```
(S = IMAP server; C = IMAP client)

C: < connects to IMAP port>
S: * OK
C: C1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SASL-IR SORT [...] AUTH=OPENID20
S: C1 OK Capability Completed
C: C2 AUTHENTICATE OPENID biwsaHR0cHM6Ly9vcGVuaWQuZXhhbXBsZS8=
[ This is the base64 encoding of "n,,https://openid.example/" .
  Server performs discovery on http://openid.example/ ]
S: + aHR0cHM6Ly9vcGVuaWQuZXhhbXBsZS9vcGVuaWQvP29wZW5pZC5ucz1
    odHRwOi8vc3BlY3Mub3BlbmklmLm5ldC9hdXRoLzIuMCZvcGVuaWQucm
    V0dXJuX3RvPWw0dHBzOi8vbWFpbC5leGFtcGxlL2NvbnNlbWVYLzFlZ
    jg4OGMmb3BlbmklmLmNsYWltZWRFaWQ9aHR0cHM6Ly9vcGVuaWQuZXhh
    bXBsZS8mb3BlbmklmLmlkZW50aXR5PWh0dHBzOi8vb3BlbmklmLmV4YW1
    wbGUvJm9wZW5pZC5yZWZfbTlpbWFWb0i8vbWFPbC5leGFtcGxlJm9wZW
    5pZC5tb2RlPWNoZWNaWRfc2V0dXA=

[ This is the base64 encoding of "https://openid.example/openid/
?openid.ns=http://specs.openid.net/auth/2.0
&openid.return_to=https://mail.example/consumer/lef888c
&openid.claimed_id=https://openid.example/
&openid.identity=https://openid.example/
&openid.realm=imap://mail.example
&openid.mode=checkid_setup"
with line breaks and spaces added here for readability.
]
C: PQ==
[ The client now sends the URL it received to a browser for
processing. The user logs into https://openid.example, and
agrees to authenticate imap://mail.example. A redirect is
passed back to the client browser who then connects to
https://imap.example/consumer via SSL with the results.
From an IMAP perspective, however, the client sends the "="
response, and awaits mail.example.
Server mail.example would now contact openid.example with an
openid.check_authenticate message. After that...
]
S: + ZWlhawW9bGVhckBtYWlsLmV4YW1wbGUSznVsbg5hbWU9RWxp
    b3QlMjBMZWfY
[ Here the IMAP server has returned an SREG attribute of
email=lear@mail.example,fullname=Eliot%20Lear.
Line break in response added in this example for clarity. ]
C:
[ In IMAP client must send a blank response after receiving the
SREG data. ]
S: C2 OK
```

In this example, the SASL server / RP has made use of a transaction id 1ef888c.

6. Security Considerations

This section will address only security considerations associated with the use of OpenID with SASL and GSS-API. For considerations relating to OpenID in general, the reader is referred to the OpenID specification and to other literature [1]. Similarly, for general SASL [RFC4422] and GSS-API [RFC5801] Security Considerations, the reader is referred to those specifications.

6.1. Binding OpenIDs to Authorization Identities

As specified in [RFC4422], the server is responsible for binding credentials to a specific authorization identity. It is therefore necessary that a registration process takes place in advance that binds specific OpenIDs to specific authorization identities, or that only specific trusted OpenID Providers be allowed, where a mapping is predefined. For example, it could be pre-arranged between an IdP and RP that "https://example.com/user" maps to "user" for purposes of authorization.

6.2. RP redirected by malicious URL to take an improper action

In the initial SASL client response a user or host can transmit a malicious response to the RP for purposes of taking advantage of weaknesses in the RP's OpenID implementation. It is possible to add port numbers to the URL so that the outcome is the RP does a port scan of the site. The URL could contain an unauthorized host or even the local host. The URL could contain a protocol other than http or https, such as file or ftp.

One mitigation would be for RPs to have a list of authorized URI bases. OPs SHOULD only redirect to RPs with the same domain component of the base URI. RPs MUST NOT automatically retry on failed attempts. A log of those sites that fail SHOULD be kept, and limitations on queries from clients SHOULD be imposed, just as with any other authentication attempt. Applications SHOULD NOT invoke browsers to communicate with OPs that they are not themselves configured with.

6.3. User Privacy

The OP is aware of each RP that a user logs into. There is nothing in the protocol to hide this information from the OP. 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 OpenID Providers will be able to track - to some extent - user access to their services and any additional information that OP provides.

7. IANA Considerations

The IANA is requested to update the SASL Mechanism Registry using the following template, as described in [RFC4422].

SASL mechanism name: OPENID20

Security Considerations: See this document

Published specification: See this document

Person & email address to contact for further information: Authors of this document

Intended usage: COMMON

Owner/Change controller: 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.

8. Acknowledgments

The authors would like to thank Alexey Melnikov, Joe Hildebrand, Mark Crispin, Chris Newman, Leif Johansson, Sam Hartman, Nico Williams, Klaas Wierenga, Stephen Farrell, and Stephen Kent for their review and contributions.

9. References

9.1. Normative References

- [OpenID] OpenID Foundation, "OpenID Authentication 2.0 - Final", December 2007.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2606] Eastlake, D.E. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, June 1999.
- [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.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [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.
- [RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.
- [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.
- [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.
- [SREG1.0] OpenID Foundation, "OpenID Simple Registration Extension version 1.0", June 2006.
- [XRI2.0] Reed, D. and D. McAlpin, "Extensible Resource Identifier (XRI) Syntax V2.0", OASIS Standard xri-syntax-V2.0-cs, September 2005.

9.2. Informative References

- [RFC1939] Myers, J.G. and M.T. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.
- [RFC4959] Siemborski, R. and A. Gulbrandsen, "IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response", RFC 4959, September 2007.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [W3C.REC-html401-19991224] Hors, A., Raggett, D. and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<http://www.w3.org/TR/1999/REC-html401-19991224>>.

Appendix A. Changes

This section to be removed prior to publication.

- o 04 - 07 04 - 07 address LC and review comments, including those of Stephen Farrell, Steve Kent, and Brian Carpenter.
- o 03 Clarifies messages and ordering, and replace the empty message with a "=" message.
- o 02 Address all WGLC comments.
- o 01 Specific text around possible improvements for OOB browser control in security considerations. Also talk about transaction id.
- o 00 WG -00 draft. Slight wording modifications about design constraints per Alexey.
- o 02 Correct single (significant) error on mechanism name.
- o 01 Add nonce discussion, add authorized identity, explain a definition. Add gs2 support.
- o 00 Initial Revision.

Authors' Addresses

Eliot Lear
Cisco Systems GmbH
Richtistrasse 7
Wallisellen, ZH CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo, 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Henry Mauldin
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 (800) 553-6387
Email: hmauldin@cisco.com

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm, 113 47
SE

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 23, 2012

K. Wierenga
Cisco Systems, Inc.
E. Lear
Cisco Systems GmbH
S. Josefsson
SJD AB
February 20, 2012

A SASL and GSS-API Mechanism for SAML
draft-ietf-kitten-sasl-saml-09.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 August 23, 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	3
1.1. Terminology	4
1.2. Applicability	4
2. Authentication flow	6
3. SAML SASL Mechanism Specification	9
3.1. Initial Response	9
3.2. Authentication Request	10
3.3. Outcome and parameters	11
4. SAML GSS-API Mechanism Specification	12
4.1. GSS-API Principal Name Types for SAML	13
5. Examples	14
5.1. XMPP	14
5.2. IMAP	18
6. Security Considerations	22
6.1. Man in the middle and Tunneling Attacks	22
6.2. Binding SAML subject identifiers to Authorization Identities	22
6.3. User Privacy	22
6.4. Collusion between RPs	22
6.5. GSS-API specific security considerations	22
7. IANA Considerations	24
7.1. IANA mech-profile	24
7.2. IANA OID	24
8. References	25
8.1. Normative References	25
8.2. Informative References	26
Appendix A. Acknowledgments	28
Appendix B. Changes	29
Authors' Addresses	30

1. Introduction

Security Assertion Markup Language (SAML) 2.0

[OASIS.saml-core-2.0-os] is a set of specifications that provide 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 enables interworking between SASL and SAML in order to assert the identity of the user 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 4.1 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 instance is available. In the absence of a browser instance, SAML profiles that don't require a browser such as the Enhanced Client or Proxy profile (as defined in section 4.2 of the SAML profiles 2.0 specification [OASIS.saml-profiles-2.0-os]) may be used, but that is outside the scope of this specification.

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.

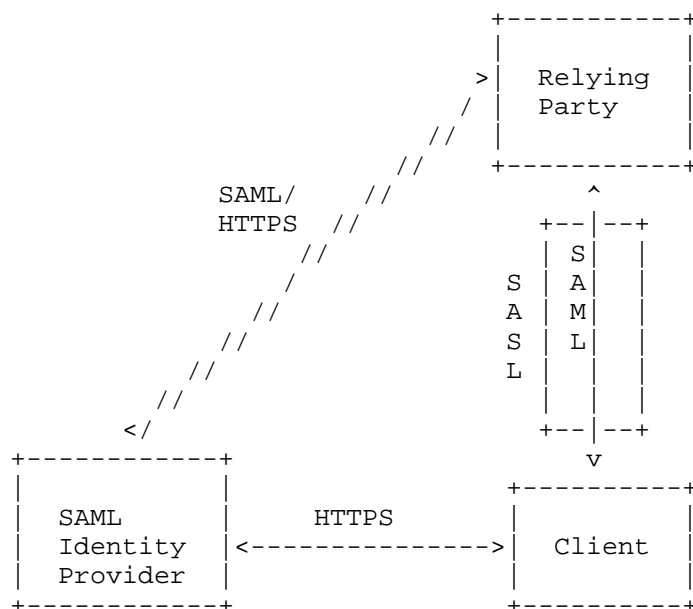


Figure 1: Interworking Architecture

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, or over similar integrity protected and authenticated channels. In addition, when TLS is used the client MUST successfully validate the server certificate ([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. When it does so, it includes the following parameters: (1) an authentication request that contains the name of resource being requested, (2) a browser cookie, and (3) 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 release of user attributes 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 Uniform 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 and the SASL server will receive the answer to the challenge out-of-band from the SASL conversation.
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 user 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 is 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 "=" (i.e., an empty response). 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:

```
initial-response = gs2-header Idp-Identifier
IdP-Identifier = domain ; domain name with corresponding IdP
```

The "gs2-header" is used as follows:

- The "gs2-nonstd-flag" MUST NOT be present.
- The "gs2-cb-flag" MUST be set to "n" because channel binding [RFC5056] 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.)
- The "gs2-authzid" carries the optional authorization identity as specified in [RFC5801] (not to be confused with the IdP-Identifier).

Domain name is specified in [RFC1035]. A domain name is either a "traditional domain name" as described in [RFC1035] or an "internationalized domain name" as described in [RFC5890]. Clients

and servers MUST treat the IdP-Identifier as a domain name slot [RFC5890]. They also SHOULD support internationalized domain names (IDNs) in the Idp-Identifier field; if they do so, all of the domain name's labels MUST be A-labels or NR-LDH labels [RFC5890], if necessary internationalized labels MUST be converted from U-labels to A-labels by using the Punycode encoding [RFC3492] for A-labels prior to sending them to the SASL-server as described in the protocol specification for Internationalized Domain Names in Applications [RFC5891].

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 that 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]. Should the client support Internationalized Resource Identifiers (IRIs) [RFC3987] it MUST first convert the IRI to a URI before transmitting it to the server [RFC5890].

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 session it has with the SASL client with the appropriate SAML authentication result. It can do so by comparing the ID of the SAML authentication request it has issued with the one it receives in the SAML authentication statement.

3.3. Outcome and parameters

The SASL server (in its capacity as a SAML Relying Party) now validates the SAML authentication response it received from the SAML client via HTTP or HTTPS.

The outcome of that validation by the SASL server constitutes a SASL mechanism outcome, and therefore (as stated in [RFC4422]) 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 8 in Figure 2).

4. SAML GSS-API Mechanism Specification

This section and its sub-sections are not required for SASL implementors, but this section **MUST** be observed to implement the GSS-API mechanism discussed below.

This section specifies a GSS-API mechanism that when used via the GS2 bridge to SASL behaves like the SASL mechanism defined in this document. Thus, it can loosely be said that the SAML SASL mechanism is 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]. More precisely, to pass identity validation the client uses the securely negotiated `targ_name` as the reference identifier and match it to the DNS-ID of the server certificate, and **MUST** reject the connection if there is a mismatch. For compatibility with deployed certificate hierarchies, the client **MAY** also perform a comparison with the CN-ID when there is no DNS-ID present. Wildcard matching is permitted. The `targ_name` reference identifier is a "traditional domain names" thus the comparison is made using case-insensitive ASCII comparison.

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. Examples

5.1. 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 according to the definition in Section 4 of 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:

```
aHR0cHM6Ly9zYWlsLmV4YW1wbGUub3JnL1NBTVwvQnJvd3Nlcj9TQU1MUwVx
dWVzdD1QSE5oYld4d09rRjFkR2h1VW1WeGRXVnpkQ0I0Yld4dWN6cHpZVzFz
Y0QwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUUVxc2TWk0d09uQnli
MlJ2WTI5c0lnMETJQ0FnSUVsRVBTSmZzbVZqTkRJMFPtRTFNVEF6TkRJNE9U
QTVZVE13Wm1ZeFpUTXhNVFk0TXpJm1pqYzVORGmWt1RnMElpQldaWEp6YVc5
dVBTSXlMakFpRFFvZ0lDQWdTWE56ZFdwSmJuTjBZVzUwUfNJeU1EQTNMVEV5
TFRFd1ZERXhPak01T2pNMfdpSWdSbTl5WTJWQmRYUm9iajBpWm1Gc2MyVW1E
UW9nSUNBZ1NYTlFZWE56YVhabFBTSm1ZV3h6W1NJTkNpQWdJQ0JRY205MGly
TnZiRUpwYm1ScGJtYz1JblZ5YmpwdllYtNbjenB1WVcxbGN6cDBZenBUUVUx
TU9qSXVNRHBpYVc1a2FXNW5jenBJVkJZSUUxWQ1BVMVFpRFFvZ0lDQWdRWE56
WlhKMGFxOXVrMj1lYzNWdFpYS1RaWEoyYVdObFZWSk1QUTBLSUNBZ0lDQWdJ
Q0FpYUhSMGNITtZMeTk0YlhCd0xtVjRZVzF3YkdVdVkyOXRMmU5CVFV3dlFY
TnpaWEowYVc5dVEyOXVjMlZ0WlhKVFPySjJhV05sSWo0TkNpQThjMkZ0YkRw
SmMzTjFawe1nZUcxc2JuTTZjMkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6
T25Sak9sTkJUUVxc2TWk0d09tRnpjMlZ5ZEdsdmJpSSStEUW9nSUNBZ0lHaDBk
SEJ6T2k4dmVHMxdjQzVsZUdGdGNHeGxMbU52Y1EwS01Ed3ZjMkZ0YkRwSmMz
TjFaweKrrFFvZ1BITmhiV3h3T2s1aGJXVkpSRKJ2YkdsamVTQjRiV3h1Y3pw
ellXMXNjRDBpZFhKdU9tOWhjMmx6T201aGJXVnpPblJqT2xOQlRVdzZNaTR3
T25CeW1zUnZmZj1zSWcwS0lDQWdJQ0JHYjNkdFlYUTlJblZ5YmpwdllYtNbJ
enB1WVcxbGN6cDBZenBUUVUxTU9qSXVNRHB1WVcxbGFxUXRabTl5YldGME9u
Qmxjbk5wYzNSbGJuUW1EUW9nSUNBZ0lGTLFUbUZ0WlZGMV1XeHBabWxsY2ow
aWVHMxdjQzVsZUdGdGNHeGxMbU52Y1lNJZlFXeHNiM2REY21WaGRHVt1JblJ5
ZFdVaUlDOCTEUW9nUEhOaGJXeHdPbEpsY1hWbGMzUmxaRUyxZEdodVEyOXVk
R1Y0ZEEwS0lDQWdJQ0I0Yld4dWN6cHpZVzFzY0QwaWRYSnVPbTloYzJsek9t
NWhiV1Z6T25Sak9sTkJUUVxc2TWk0d09uQnliMlJ2WTI5c0lpQU5DaUfnSUNB
Z0lDQWdRMj10Y0dGeWfYTnZiajBpWlhoaFkzUWlQZzBLSUNBOGMyRnRiRHBC
ZFhSb2JrTnZib1JsZUhsSRGJHRnpjMUpsWmcwS0lDQWdJQ0FnZUcxc2JuTTZj
MkZ0YkQwaWRYSnVPbTloYzJsek9tNWhiV1Z6T25Sak9sTkJUUVxc2TWk0d09t
RnpjMlZ5ZEdsdmJpSSStEUW9nb0NBZ0lDQjFjbTQ2YjJGemFYTTZibUZ0WlhN
NmRHTTtZVMEZOVERveUxqQTZZV002WTJ4aGMzTmxjenBRWVhOemQyOXlaRkKJ5
YjNSbFkzUmxaRlJ5WVclemNH0X1kQTBLsUNBOEwzTmhiV3c2UVhWMGFHNURi
MjUwWlhoMFEyeGhjM05TWldZKORRb2dQQz16WVcxc2NEcFNaWEYxWlhoMFPX
UkJkWFJvYmtOdmJuUmxlSFERsUEwS1BDOXpZVzFzY0RwQmRYUm9ibEpsY1hW
bGMzUSs=
```

The decoded challenge is:

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

```

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</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 following steps between brackets are out of scope for this

document but included to better illustrate the entire flow.

[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:

```
<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'>
  <not-authorized/>
</failure>
</stream:stream>
```

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

5.2. 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: biwsZXhhbXBsZS5vcmc
S: + aHR0cHM6Ly9zYWlsLmV4YW1wbGUub3JnL1NBTVwvQnJvd3Nlcj9TQU1M
UmVxdWVzdD1QSE5oYld4d09rRg0KMWRHaHVVbVZ4ZFdWemRDQjRiV3h1Y3pwe
l1lXMxNjRDBpZFhkdU9tOWhjMmx6T201aGJXVnpPblJqT2x0Qg0KVfV3Nk1pNH
dPbkJ5YjNSdlkyOXNJZzBLSUNBZ0lFbEVQU0pmWW1Wak5ESTBabUUxTVRBek5
ESTRPVEE1WQ0KVE13WmlZeFpUTXhNVFk0TXpJm1pqYzVORGMwTlRnMElpQlda
WEp6YVc5dVBTSXlMakFpRFFvZ0lDQWdTWA0KTnpkVlZKYm5OMFlXNTBQU0l5T
URBM0xURXlMVEV3VkrFeE9qTTVPak0wV2lJZlJtOXlZMlZCZFhSb2JqMA0KaV
ptRnNjMlVpRFFvZ0lDQWdTWE5RWVhOemFYWmxQU0ptWVd4elpTSU5DaUFnSUN
CUWNTOTBiMk52YkVKcA0KYm1ScGJtYz1JblZ5YmpwdllYTnBjenBlWVcxbGN6
cDBZenBUUVUxTU9qSXVNRHBpYVc1a2FXNW5jenBJVg0KRlJRTFZCUFUxUWlEU
W9nSUNBZlFYTnpaWEowYVc5dVEyOXVjMlZ0WlhKVfPYSjJhV05sVlZKTvBRME
tJQw0KQWdJQ0FnSUNBawFIUjBjSE02THk5dFlXbHNmbVY0WVcxd2JHVXVZMj1
0TDFoQlRvd3ZrWE56WlhkMGFXOQ0KdVEyOXVjMlZ0WlhKVfPYSjJhV05sSWo0
TkNpQThjMkZ0YkRwSmMzTjFfaWElnZUcxc2JuTTZjMkZ0YkQwaQ0KZFhkdU9tO
WhjMmx6T201aGJXVnpPblJqT2x0QlRvdzZNaTR3T2lGemMyVnlkR2x2YmlJK0
RRb2dJQ0FnSQ0KR2gwZEhCek9pOHZlRzF3Y0M1bGVHRnRjR3hsTG1OdmJRMEt
JRhd2YzJGdGJEcEpjM04xWlhJK0RRb2dQSA0KTmhiV3h3T2s1aGJXVkpSRkJ2
YkdsamVTQjRiV3h1Y3pwe1lXMxNjRDBpZFhkdU9tOWhjMmx6T201aGJXVg0Ke
k9uUmpPbE5CVFV3Nk1pNHdPbkJ5YjNSdlkyOXNJZzBLSUNBZ0lDQkdIM0p0WV
hROUluVnlianB2WVhOcA0KY3pwdVlXMWxjenAwWXpwVFFVMU1Pak1lTURwdVl
XMWxhVlF0Wm05eWJXRjBPbkJsY25OcGMzMmxib1FpRA0KUW9nSUNBZ0lGTLFU
bUZ0WlZGMVlXeHBabWxsY2owaWVHMxdjQzVsZUdGdGNHeGxMbU52YlNjZlFXe
HNiMw0KZERjbVZoZEdVOUluUnlkVlVpSUM4K0RRb2dQSE5oYld4d09sSmxjWF
ZsYzNSbFpFRjFkR2hlUTI5dWRHVg0KNGRBMETJQ0FnSUNCNGJXehVjenB6WVc
xc2NEMGlkWEp1T205aGMybHpPbTVoYldWek9uUmpPbE5CVFV3Ng0KTWk0d09u
QnliMlJ2WTI5c0lpQU5DaUFnSUNBZ0lDQWdRMj10Y0dGeWFFYTnZiajBpWlhoa
FkzUWlQZzBLSQ0KQ0E4YzJGdGJEcEJkWFJvYmtOdmJuUmx1SFJEYkdGemMxSm
xaZzBLSUNBZ0lDQWdlRzFzYm5NNmMyRnRiRA0KMGlkWEp1T205aGMybHpPbTV
oYldWek9uUmpPbE5CVFV3Nk1pNHdPbUZ6YzJWeWRHbHZiaUkrRFFvZ0lDQQ0K
Z0lDQjFjbTQ2YjJGemFYTTZibUZ0WlhNNmRHTTZVMEZOVERveUxqQTZZV002W
TJ4aGMzTmxjenBRWVhOeg0KZDI5eVpGQnliMlJsWTNSbFpGUnlZVzV6Y0c5eW
RBMETJQ0E4TDNOaGJXdzZRWfYwYUc1RGiYNTBaWGgwUQ0KMnhoYzNOU1pXWSt
EUW9nUEM5e1lXMxNjRHBTWlhGMVpYTjBaVlJCZFhSb2JrTnZib1JsZUhRK0lB
METQqW0K0XpZVzFzY0RwQmRYUm9ibEpsYlhWbGMzUSs=
C:
S: . OK Success (tls protection)
```

The decoded challenge is:

```
https://saml.example.org/SAML/Browser?SAMLRequest=PHNhbWxwOkF1dGhuUmVxdWVzdCB4bWxuczpZbWlscD0idXJuOm9hc2lzOm5hbWVzOnRjOlNB
TUw6Mi4wOnByb3RvY29sIg0KICAgIElEPSJfYmVjNDI0ZmElMTAzNDI4OTA5Y
TMwZmYxZTMxMTY4MzI3Zjc5NDc0OTg0IiBWZXJzaW9uPSIyLjAiDQogICAgSX
NzdWVJbnN0YW50PSIyMDA3LTEyLTEwVDEwOjM5OjM0WiIgRm9yY2VBdXRobj0i
iZmFsc2UiDQogICAgSXNQYXNzaXZlPSJmYWxzZSINCiAgICBQcm90b2NvbEJp
bmRpbmc9InVybjpYXNpczpuYW1lc3p0YzptQU1MOjIuMDpiaW5kaW5nczpiV
FRQLVBPU1QiDQogICAgQXNzZXJ0aW9uQ29uc3VtZXJTZXJ2aWNlVVJMPQ0KIC
AgICAgICAiaHR0cHM6Ly9tYWlsLmV4YW1wbGUuY29tLlNBTVwvQXNzZXJ0aW9u
Q29uc3VtZXJTZXJ2aWNlIj4NCiA8c2FtbDpJc3NlZXIgeGlsbnM6c2FtbD0i
dXJuOm9hc2lzOm5hbWVzOnRjOlNBTVw6Mi4wOmFzc2VydGlvbiI+DQogICAgI
Gh0dHBzOi8veG1wcC5leGFtcGxlLmNvbQ0KIDwvc2FtbDpJc3NlZXI+DQogPH
NhbWxwOk5hbWVJRFBvbGljeSB4bWxuczpZbWlscD0idXJuOm9hc2lzOm5hbWV
zOnRjOlNBTVw6Mi4wOnByb3RvY29sIg0KICAgICBGB3JtYXQ9InVybjpYXNp
czpuYW1lc3p0YzptQU1MOjIuMDpuYW1laWQtZm9ybWFOOnBlcnNpc3RlbnQiD
QogICAgIFNQtmFtZVF1YWxpZmllcj0ieG1wcC5leGFtcGxlLmNvbSIgQWxs3
dDcmVhdGU9InRydWUiIC8+DQogPHNhbWxwOlJlcXVlc3RlZEF1dGhuQ29udGV
4dA0KICAgICB4bWxuczpZbWlscD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTVw6
Mi4wOnByb3RvY29sIiANCiAgICAgICAgQ29tcGFyaXNvbjo0iZXhhY3QiPg0KI
CA8c2FtbDpBdXRobjNvbRleHRDbGFzc1JlZg0KICAgICAgeGlsbnM6c2FtbD
0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTVw6Mi4wOmFzc2VydGlvbiI+DQogICA
gICBlcm46b2FzaXM6bmFtZXNM6dGM6U0FNTDoyLjA6YWM6Y2xhc3Nlc3pQYXNz
d29yZFBYb3RlY3RlZFRyYW5zcG9ydA0KICA8L3NhbWw6QXV0aG5Db250ZXh0Q
2xhc3NSZWY+DQogPC9zYW1scDpSZXF1ZXN0ZWRBdXRobjNvbRleHQ+IA0KPC
9zYW1scDpBdXRobjJlcXVlc3Q+
```

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://mail.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>
```

6. 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.

6.1. 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.

6.2. 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.

6.3. 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.

6.4. 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 a Relying Party specific opaque identifier. This way the Relying Party would never see the actual user identifier, but a randomly generated identifier.

6.5. GSS-API specific security considerations

Security issues inherent in GSS-API (RFC 2743) and GS2 (RFC 5801) apply to the SAML GSS-API mechanism defined in this document. Further, and as discussed in section 4, proper TLS server identity

verification is critical to the security of the mechanism.

7. IANA Considerations

7.1. 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

Intended usage: COMMON

Note: None

7.2. IANA OID

The IANA is further requested to assign a new entry for this GSS mechanism in the sub-registry for SMI Security for Mechanism Codes, whose prefix is `iso.org.dod.internet.security.mechanisms` (1.3.6.1.5.5) and to reference this specification in the registry.

8. References

8.1. 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.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [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.
- [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.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [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.
- [W3C.REC-html401-19991224]
Hors, A., Raggett, D., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<http://www.w3.org/TR/1999/REC-html401-19991224>>.

8.2. Informative References

- [RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

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.

- o 09 Fixed text per IESG review
- o 08 Fixed text per Gen-Art review
- o 07 Fixed text per comments Alexey Melnikov
- o 06 Fixed text per AD comments
- o 05 Fixed references per ID-nits
- o 04 Added request for IANA assignment, few text clarifications
- o 03 Number of cosmetic changes, fixes per comments Alexey Melnikov
- o 02 Changed IdP URI to domain per Joe Hildebrand, fixed some typos
- o 00 WG -00 draft. Updates GSS-API section, some fixes per Scott Cantor
- o 01 Added authorization identity, added GSS-API specifics, added client supplied IdP
- o 00 Initial Revision.

Authors' Addresses

Klaas Wierenga
Cisco Systems, Inc.
Haarlerbergweg 13-19
Amsterdam, Noord-Holland 1101 CH
Netherlands

Phone: +31 20 357 1752
Email: klaas@cisco.com

Eliot Lear
Cisco Systems GmbH
Richtistrasse 7
Wallisellen, ZH CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

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

KITTEN
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

W. Mills
T. Showalter
Yahoo! Inc.
H. Tschofenig
Nokia Siemens Networks
October 31, 2011

A SASL and GSS-API Mechanism for OAuth
draft-mills-kitten-sasl-oauth-04.txt

Abstract

OAuth enables a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf.

This document defines how an application client uses OAuth over the Simple Authentication and Security Layer (SASL) or the Generic Security Service Application Program Interface (GSS-API) to access a protected resource at a resource serve, and additionally defines authorization and token issuing endpoint discovery. Thereby, it enables schemes defined within the OAuth framework for non-HTTP-based application protocols.

Clients typically store the user's long term credential. This does, however, lead to significant security vulnerabilities, for example, when such a credential leaks. A significant benefit of OAuth for usage in those clients is that the password is replaced by a token. Tokens typically provided limited access rights and can be managed and revoked separately from the user's long-term credential (password).

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 May 3, 2012.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
2. Terminology	7
3. OAuth SASL Mechanism Specification	8
3.1. Channel Binding	8
3.2. Initial Client Response	8
3.2.1. Query String in OAUTH-PLUS	9
3.3. Server's Response	9
3.4. Mapping to SASL Identities	10
3.5. Discovery Information	10
3.6. Use of Signature Type Authorization	12
4. GSS-API OAuth Mechanism Specification	14
5. Examples	15
5.1. Successful Bearer Token Exchange	15
5.2. MAC Authentication with Channel Binding	15
5.3. Failed Exchange	16
5.4. Failed Channel Binding	17
6. Security Considerations	18
7. IANA Considerations	19
7.1. SASL Registration	19
7.2. GSS-API Registration	19
7.3. Link Type Registration	19
7.3.1. OAuth 2 Authentication Endpoint	19
7.3.2. OAuth 2 Token Endpoint	20
7.3.3. OAuth 1.0a Request Initiation Endpoint	20
7.3.4. OAuth 1.0a Authorization Endpoint	21
7.3.5. OAuth 1.0a Token Endpoint	21
8. Appendix A -- Document History	22
9. References	23
9.1. Normative References	23
9.2. Informative References	24
Authors' Addresses	25

1. Introduction

OAuth [I-D.ietf-oauth-v2] enables a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf. The core OAuth specification [I-D.ietf-oauth-v2] does not define the interaction between the client and the resource server with the access to a protected resource using an Access Token. This functionality is described in two separate specifications, namely [I-D.ietf-oauth-v2-bearer], and [I-D.ietf-oauth-v2-http-mac], whereby the focus is on an HTTP-based environment only.

Figure 1 shows the abstract message flow as shown in Figure 1 of [I-D.ietf-oauth-v2].

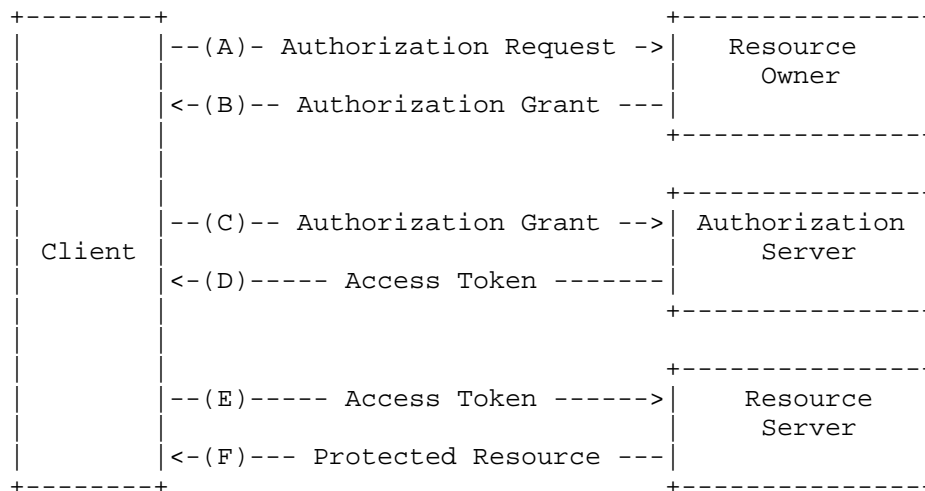


Figure 1: Abstract OAuth 2.0 Protocol Flow

This document takes advantage of the OAuth protocol and its deployment base to provide a way to use SASL [RFC4422] as well as the GSS-API [RFC2743] to gain access to resources when using non-HTTP-based protocols, such as the Internet Message Access Protocol (IMAP) [RFC3501], which is what this memo uses in the examples.

The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing

mechanisms and allows old protocols to make use of new mechanisms. The framework also provides a protocol for securing subsequent protocol exchanges within a data security layer.

The Generic Security Service Application Program Interface (GSS-API) [RFC2743] provides a framework for applications to support multiple authentication mechanisms through a unified interface.

This document defines a SASL mechanism for OAuth, 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. Implementers may be interested in either the SASL, the GSS-API, or even both mechanisms. To facilitate these two variants, the description has been split into two parts, one part that provides normative references for those interested in the SASL OAuth mechanism (see Section 3), and a second part for those implementers that wish to implement the GSS-API portion (see Section 4).

When OAuth is integrated into SASL and the GSS-API the high-level steps are as follows:

- (A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary.
- (B) The client receives an authorization grant which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.
- (C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
- (D) The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token.
- (E) The client requests the protected resource from the resource server and authenticates by presenting the access token.
- (F) The resource server validates the access token, and if valid, serves the request.

Steps (E) and (F) are not defined in [I-D.ietf-oauth-v2] and are the

main functionality specified within this document. Additionally, an optional discovery exchange is defined. Consequently, the message exchange shown in Figure 2 is the result of this specification. (1) and (2) denote the optional discovery exchange steps that may happen before the OAuth 2.0 protocol exchange messages in steps (A)-(D) are executed. Steps (E) and (F) also defined in this specification.

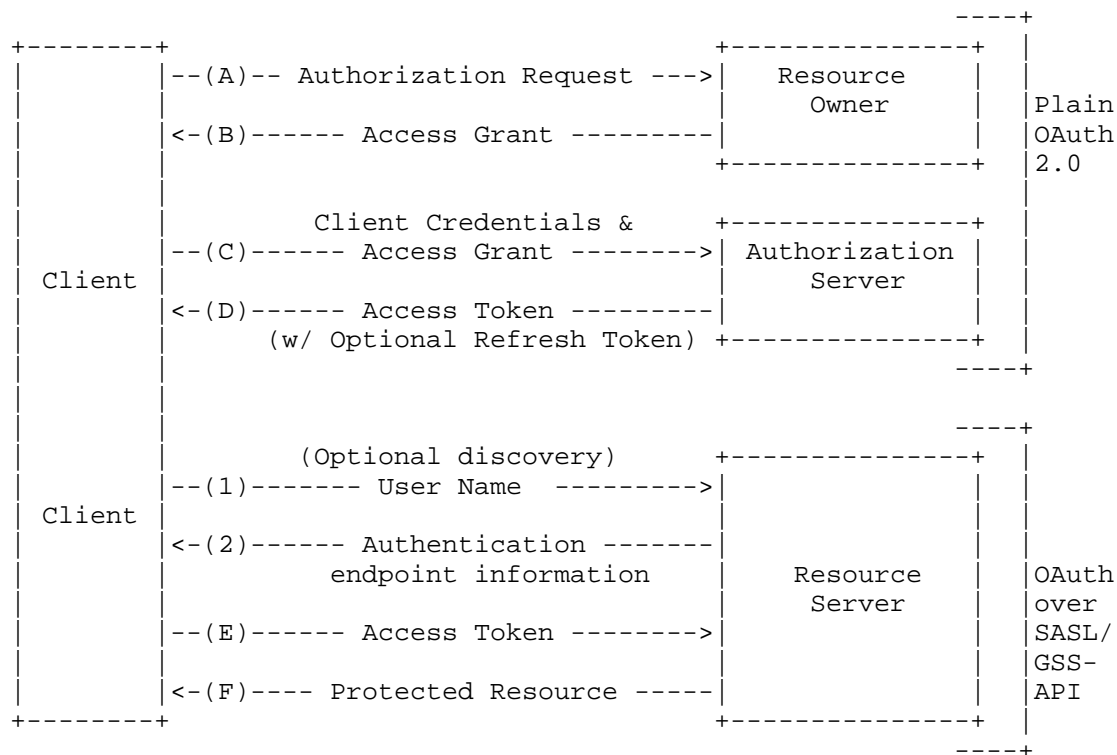


Figure 2: OAuth SASL Architecture

Note: The discovery procedure in OAuth is still work in progress. Hence, the discovery components described in this document should be considered incomplete and a tentative proposal. In general, there is a trade off between a generic, externally available defined discovery mechanisms (such as Webfinger using host-meta [I-D.hammer-hostmeta], or [I-D.jones-simple-web-discovery]) and configuration information exchanged in-band between the SASL communication endpoints.

It is worthwhile to note that this specification is also compatible with OAuth 1.0a [RFC5849].

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 [RFC2119].

The reader is assumed to be familiar with the terms used in the OAuth 2.0 specification [I-D.ietf-oauth-v2].

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. Line breaks have been inserted for readability.

Note that the IMAP SASL specification requires base64 encoding message, not this memo.

3. OAuth SASL Mechanism Specification

SASL is used as a generalized authentication method in a variety of application layer protocols. This document defines two SASL mechanisms for usage with OAuth: "OAUTH" and "OAUTH-PLUS". The "OAUTH" SASL mechanism provides bearer token alike semantic for SASL while "OAUTH-PLUS" provides a semantic similar to OAuth MAC authentication by utilizing a channel binding mechanism [RFC5056].

3.1. Channel Binding

If the specification for the underlying authorization scheme requires a security layer, such as TLS [RFC5246], the server SHOULD only offer a mechanism where channel binding can be enabled.

The channel binding data is computed by the client based on it's choice of preferred channel binding type. As specified in [RFC5056], the channel binding information MUST start with the channel binding unique prefix, followed by a colon (ASCII 0x3A), followed by a base64 encoded channel binding payload. The channel binding payload is the raw data from the channel binding type if the raw channel binding data is less than 500 bytes. If the raw channel binding data is 500 bytes or larger then a SHA-1 [RFC3174] hash of the raw channel binding data is computed.

If the client is using tls-unique for a channel binding then the raw channel binding data equals the first TLS finished message. This is under the 500 byte limit, so the channel binding payload sent to the server would be the base64 encoded first TLS finished message.

In the case where the client has chosen tls-endpoint, the raw channel binding data is the certificate of the server the client connected to, which will frequently be 500 bytes or more. If it is then the channel binding payload is the base64 encoded SHA-1 hash of the server certificate.

3.2. Initial Client Response

The SASL client response is formatted as an HTTP [RFC2616] request. The HTTP request is limited in that the path MUST be "/". In the OAUTH mechanism no query string is allowed. The following header lines are defined in the client response:

User (OPTIONAL): Contains the user identifier being authenticated, and is provided to allow correct discovery information to be returned.

Host (REQUIRED): Contains the host name to which the client connected.

Authorization (REQUIRED): An HTTP Authorization header.

The user name is provided by the client to allow the discovery information to be customized for the user, a given server could allow multiple authenticators and it needs to return the correct one. For instance, a large ISP could provide mail service for several domains who manage their own user information. For instance, users at foo-example.com could be authenticated by an OAuth service at <https://oauth.foo-example.com/>, and users at bar-example.com could be authenticated by https://oauth.bar-example.com, but both could be served by a hypothetical IMAP server running at a third domain, imap.example.net.

3.2.1. Query String in OAUTH-PLUS

In the OAUTH-PLUS mechanism the channel binding information is carried in the query string. OAUTH-PLUS defines following query parameter(s):

cbdata (REQUIRED): Contains the base64 encoded channel binding data, properly escaped as an HTML query parameter value.

3.3. Server's Response

The server validates the response per the specification for the authorization scheme used. If the authorization scheme used includes signing of the request parameters the client must provide a complete HTTP style request that satisfies the data requirements for the scheme in use.

In the OAUTH-PLUS mechanism the server examines the channel binding data, extracts the channel binding unique prefix, and extracts the raw channel binding data based on the channel binding type used. It then computes it's own copy of the channel binding payload and compares that to the payload sent by the client in the query parameters of the tunneled HTTP request. Those two must be equal for channel binding to succeed.

The server responds to a successfully verified client message by

completing the SASL negotiation. The authentication scheme MUST carry the user ID to be used as the authorization identity (identity to act as). The server MUST use that ID as the user being authorized, that is the user assertion we accept and not other information such as from the URL or "User:" header.

The server responds to failed authentication by sending discovery information in an HTTP style response with the HTTP status code set to 401, and then failing the authentication.

If channel binding is in use and the channel binding fails the server responds with a minimal HTTP response without discovery information and the HTTP status code set to 412 to indicate that the channel binding precondition failed. If the authentication scheme in use does not include signing the server SHOULD revoke the presented credential and the client SHOULD discard that credential.

3.4. Mapping to SASL Identities

Some OAuth mechanisms can provide both an authorization identity and an authentication identity. An example of this is OAuth 1.0a [RFC5849] where the consumer key (oauth_consumer_key) identifies the entity using to token which equates to the SASL authentication identity, and is authenticated using the shared secret. The authorization identity in the OAuth 1.0a case is carried in the token (per the requirement above), which SHOULD validated independently. The server MAY use a consumer key or other comparable identity in the OAuth authorization scheme as the SASL authentication identity. If an appropriate authentication identity is not available the server MUST use the identity asserted in the token.

3.5. Discovery Information

The server MUST send discovery information in response to a failed authentication exchange or a request with an empty Authorization header. If discovery information is returned it MUST include an authentication endpoint appropriate for the user. If the "User" header is present the discovery information MUST be for that user. Discovery information is provided by the server to the client to allow a client to discover the appropriate OAuth authentication and token endpoints. The client then uses that information to obtain the access token needed for OAuth authentication. The client SHOULD cache and re-use the user specific discovery information for service endpoints.

Discovery information makes use of both the WWW-Authenticate header as defined in HTTP Authentication: Basic and Digest Access Authentication [RFC2617] and Link headers as defined in [RFC5988].

The following elements are defined for discovery information:

WWW-Authenticate A WWW-Authenticate header for each authentication scheme supported by the server. Authentication scheme names are case insensitive. The following [RFC2617] authentication parameters are defined:

realm REQUIRED -- (as defined by RFC2617)

scope OPTIONAL -- A quoted string. This provides the client an OAuth 2 scope known to be valid for the resource.

oauth2-authenticator An [RFC5988] Link header specifying the [I-D.ietf-oauth-v2] authentication endpoint. This link has an OPTIONAL link-extension "scheme", if included this link applies ONLY to the specified scheme.

oauth2-token An [RFC5988] Link header specifying the [I-D.ietf-oauth-v2] token endpoint. This link has an OPTIONAL link-extension "scheme", if included this link applies ONLY to the specified scheme.

oauth-initiate (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] initiation endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server.

oauth-authorize (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] authentication endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server.

oauth-token (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] token endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server. This link type has one link-extension "grant-types" which is a space separated list of the OAuth 2.0 grant types that can be used at the token endpoint to obtain a token.

Usage of the URLs provided in the discovery information is defined in the relevant specifications. If the server supports multiple authenticators the discovery information returned for unknown users MUST be consistent with the discovery information for known users to prevent user enumeration. The OAuth 2.0 specification [I-D.ietf-oauth-v2] supports multiple types of authentication schemes and the server MUST specify at least one supported authentication scheme in the discovery information. The server MAY support multiple

schemes and MAY support schemes not listed in the discovery information.

If the resource server provides a scope the client SHOULD always request scoped tokens from the token endpoint. The client MAY use a scope other than the one provided by the resource server. Scopes other than those advertised by the resource server must be defined by the resource owner and provided in service documentation (which is beyond the scope of this memo).

3.6. Use of Signature Type Authorization

This mechanism supports authorization using signatures, which requires that both client and server construct the string to be signed. OAuth 2 is designed for authentication/authorization to access specific URIs. SASL is designed for user authentication, and has no facility for being more specific. In this mechanism we require an HTTP style format specifically to support signature type authentication, but this is extremely limited. The HTTP style request is limited to a path of "/". This mechanism is in the SASL model, but is designed so that no changes are needed if there is a revision of SASL which supports more specific resource authorization, e.g. IMAP access to a specific folder or FTP access limited to a specific directory.

Using the example in the MAC specification [I-D.ietf-oauth-v2-http-mac] as a starting point, on an IMAP server running on port 143 and given the MAC style authorization request (with long lines wrapped for readability) below:

```
GET / HTTP/1.1
Host: server.example.com
User: user@example.com
Authorization: MAC token="h480djs93hd8",timestamp="137131200",
              nonce="dj83hs9s",signature="YTVjyNSujYs1WsDurFnvFi4JK6o="
```

The normalized request string would be constructed per the MAC specification [I-D.ietf-oauth-v2-http-mac]. In this example the normalized request string with the new line separator character is represented by "\n" for display purposes only would be:

```
h480djs93hi8\n
137131200\n
dj83hs9s\n
\n
GET\n
server.example.com\n
143\n
/>\n
\n
```

4. GSS-API OAuth Mechanism Specification

Note: The normative references in this section are informational for SASL implementers, but they are normative for GSS-API implementers.

The SASL OAuth mechanism is also a GSS-API mechanism and the messages described in Section 3 are the same, but

1. the GS2 header on the client's first message is excluded when OAUTH is used as a GSS-API mechanism, and
2. initial context token header is prefixed to the client's first authentication message (context token), as described in Section 3.1 of RFC 2743,

The GSS-API mechanism OID for OAuth is [[TBD: IANA]].

OAuth security contexts always have the mutual_state flag (GSS_C_MUTUAL_FLAG) set to TRUE. OAuth supports credential delegation, therefore security contexts may have the deleg_state flag (GSS_C_DELEG_FLAG) set to either TRUE or 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 OAuth mechanism does not support per-message tokens or GSS_Pseudo_random.

OAuth supports a standard generic name syntax for acceptors, such as GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4.1). These service names MUST be associated with the "entityID" claimed by the RP. OAuth supports only a single name type for initiators: GSS_C_NT_USER_NAME. GSS_C_NT_USER_NAME is the default name type. The query, display, and exported name syntaxes for OAuth principal names are all the same. There is no OAuth-specific name syntax; 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 OAuth name normalization rules.

5. Examples

These examples illustrate exchanges between an IMAP client and an IMAP server.

5.1. Successful Bearer Token Exchange

This example shows a successful OAuth 2.0 bearer token exchange with an initial client response. Note that line breaks are inserted for readability.

```
S: * IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8gSFRUUC8xLjENCKhvc3Q6IGltYXAuZXhhbXBs
  ZS5jb20NCKFldGhvcml6YXRpb246IEJFQVJFUjAiAidkY5ZGZ0NHFtVGMyTnZiMlJ
  sY2tCaGJIUmhkbWx6ZEdFdVkyOXRDZz09Ig0KDQo=
S: +
S: t1 OK SASL authentication succeeded
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response is:

```
GET / HTTP/1.1
Host: imap.example.com
Authorization: BEARER "vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdGEuY29tCg=="
```

The line containing just a "+" and a space is an empty response from the server. This response contains discovery information, and in the success case no discovery information is necessary so the response is empty. Like other messages, and in accordance with the IMAP SASL binding, the empty response is base64-encoded.

5.2. MAC Authentication with Channel Binding

This example shows a channel binding failure. The example sends the same request as above, but in the context of an OAUTH-PLUS exchange the channel binding information is missing. Note that line breaks are inserted for readability.

```

S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE MAC R0VUIC8/Y2JkYXRhPSJTRzkzSudKcFp5QnBjeUJoSUZSTVV5Q
mlhVzVoYkNCdFpYTnpZV2RsUHdvPSIgSFRUUC8xLjENCkhvc3Q6IHNlcnZlci5leGFtcG
xlLmNvbQ0KVXNlcjogdXNlckBleGFtcGxlLmNvbQ0KQXV0aG9yaXphdGlvbGogTUFDIHR
va2VuPSJoNDgwZGpzOTNoZDgiLHRpbWVzdGFtcD0iMTM3MTMxMjAwIixub25jZT0iZGo4
M2hzOXMiLHNpZ25hdHVyZT0iVlc5MUIHMTFjMlFnWW1VZ1ltOXlaVlFlSUvPSINCg0K
S: +
S: t1 OK SASL authentication succeeded

```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response is:

```

GET /?cbdata="SG93IGJpZyBpcyBhIFRmUyBmaW5hbCBtZXNzYWdlPwo=" HTTP/1.1
Host: server.example.com
User: user@example.com
Authorization: MAC token="h480djs93hd8",timestamp="137131200",
               nonce="dj83hs9s",signature="WW91IGl1c3QgYmUgYm9yZWQuIAo="

```

The line containing just a "+" and a space is an empty response from the server. This response contains discovery information, and in the success case no discovery information is necessary so the response is empty. Like other messages, and in accordance with the IMAP SASL binding, the empty response is base64-encoded.

5.3. Failed Exchange

This example shows a failed exchange because of the empty Authorization header, which is how a client can query for discovery information. Note that line breaks are inserted for readability.

```

S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8gSFRUUC8xLjENC1VzZXI6IHNjb290ZXJAYW
x0YXZpc3RhLmNvbQ0KSG9zdDogaW1hcC55YWhvby5jb20NCkFldGhlbnRpy2F0ZT
ogDQoNCg==
S: + SFRUUC8xLjEgNDExIFVudGVzYXV0aG9yaXplZA0KVldXLUFldGhlbnRpy2F0ZTogQk
VBukVSIHJlYWxtPSJleGFtcGxlLmNvbSINCkxpbms6IDxodHRwc3ovL2xvZ2luLn
lhaG9vLmNvbS9vYXV0aD4gcmVsPSJvYXV0aDItYXV0aGVudGljYXRvciIgIA0KTG
luazogPGh0dHBzOi8vbG9naW4ueWFob28uY29tL29hdXR0PiByZWw9Im91YXR0Mi
10b2tlbiINCg0K
S: t1 NO SASL authentication failed

```

The decoded initial client response is:

```
GET / HTTP/1.1
User: alice@example.com
Host: imap.example.com
Authorization:
```

The decoded server discovery response is:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: BEARER realm="example.com"
Link: <https://login.example.com/oauth> rel="oauth2-authenticator"
Link: <https://login.example.com/oauth> rel="oauth2-token"
```

5.4. Failed Channel Binding

This example shows a channel binding failure in a discovery request. The channel binding information is empty. Note that line breaks are inserted for readability.

```
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8/Y2JkYXRhPSIiIEhUVFAvMS4xDQpVc2VyOi
  BhbGljZUBleGFtcGx1LmNvbQ0KSG9zdDogaWlhcC5leGFtcGx1LmNvbQ0KQXV0aG
  9yaXphdGlvbjoNCg0K
S: + SFRUUC8xLjEgNDEyIFByZWVbmRpdGlvb2IiBGYwlsZWQNCg0KDQo=
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

```
GET /?cbdata="" HTTP/1.1
User: alice@example.com
Host: imap.example.com
Authorization:
```

The decoded server response is:

```
HTTP/1.1 412 Precondition Failed
```

6. Security Considerations

This mechanism does not provide a security layer, but does provide a provision for channel binding. The OAuth 2 specification [I-D.ietf-oauth-v2] allows for a variety of usages, and the security properties of these profiles vary. The usage of bearer tokens, for example, provide security features similar to cookies. Applications using this mechanism SHOULD exercise the same level of care using this mechanism as they would in using the SASL PLAIN mechanism. In particular, TLS 1.2 or an equivalent secure channel MUST be implemented and its usage is RECOMMENDED.

Channel binding in this mechanism has different properties based on the authentication scheme used. Channel binding to TLS with a bearer token provides only a binding to the TLS layer. Authentication schemes like MAC tokens have a signature over the channel binding information. These provide additional protection against a man in the middle attacks, and the MAC authorization header is bound to the channel and only valid in that context.

It is possible that SASL will be authenticating a connection and the life of that connection may outlast the life of the token used to authenticate it. This is a common problem in application protocols where connections are long-lived, and not a problem with this mechanism per se. Servers MAY unilaterally disconnect clients in accordance with the application protocol.

An OAuth credential is not equivalent to the password or primary account credential. There are protocols like XMPP that allow actions like change password. The server SHOULD ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

It is possible for an application server running on Evil.example.com to tell a client to request a token from Good.example.org. A client following these instructions will pass a token from Good to Evil. This is by design, since it is possible that Good and Evil are merely names, not descriptive, and that this is an innocuous activity between cooperating two servers in different domains. For instance, a site might operate their authentication service in-house, but outsource their mail systems to an external entity.

Tokens have a lifetime associated with them. Reducing both the lifetime of a token provides security benefits in case that tokens leak. In addition a previously obtained token MAY be revoked or rendered invalid at any time. The client MAY request a new access token for each connection to a resource server, but it SHOULD cache and re-use access credentials that appear to be valid.

7. IANA Considerations

7.1. SASL Registration

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTH

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 requested to register the following SASL profile:

SASL mechanism profile: OAUTH-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

7.2. GSS-API Registration

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.

7.3. Link Type Registration

Pursuant to [RFC5988] The following link type registrations [[will be]] registered by mail to `link-relations@ietf.org`.

7.3.1. OAuth 2 Authentication Endpoint

- o Relation Name: oauth2-authenticator
- o Description: An OAuth 2.0 authentication endpoint.
- o Reference:
- o Notes: This link type indicates an OAuth 2.0 authentication endpoint that can be used for user authentication/authorization for the endpoint providing the link.
- o Application Data: [optional]

7.3.2. OAuth 2 Token Endpoint

- o Relation Name: oauth2-token
- o Description: The OAuth token endpoint used to get tokens for access.
- o Reference:
- o Notes: The OAuth 2.0 token endpoint to be used for obtaining tokens to access the endpoint providing the link.
- o Application Data: This link type has one link-extension "grant-types", which is the OAuth 2.0 grant types that can be used at the token endpoint to obtain a token. This is not an exclusive list, it provides a hint to the application of what SHOULD be valid. A token endpoint MAY support additional grant types not advertised by a resource endpoint.

7.3.3. OAuth 1.0a Request Initiation Endpoint

- o Relation Name: oauth-initiate
- o Description: The OAuth 1.0a request initiation endpoint used to get tokens for access.
- o Reference:
- o Notes: The OAuth 1.0a endpoint used to initiate the sequence, this temporary request is what the user approves to grant access to the resource.
- o Application Data:

7.3.4. OAuth 1.0a Authorization Endpoint

- o Relation Name: oauth-authorize
- o Description: The OAuth 1.0a authorization endpoint used to approve an access request.
- o Reference:
- o Notes:
- o Application Data:

7.3.5. OAuth 1.0a Token Endpoint

- o Relation Name: oauth-token
- o Description: The OAuth 1.0a token endpoint used to get tokens for access.
- o Reference:
- o Notes:
- o Application Data:

8. Appendix A -- Document History

[[to be removed by RFC editor before publication as an RFC]]

-04

- o Editorial clean-up and text in introduction improved.
- o Added GSS-API support

-03

- o Fixing channel binding, not tls-unique specific. Also defining how the CB data is properly generated.
- o Various small editorial changes and embarrassing spelling fixes.

-02

- o Filling out Channel Binding
- o Added text clarifying how to bind to the 2 kinds of SASL identities.

-01

- o Bringing this into line with draft 12 of the core spec, the bearer token spec, and references the MAC token spec
- o Changing discovery over to using the Link header construct from RFC5988.
- o Added the seeds of channel binding.

-00

- o Initial revision

9. References

9.1. Normative References

- [I-D.ietf-oauth-v2]
Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Protocol", draft-ietf-oauth-v2-22 (work in progress), September 2011.
- [I-D.ietf-oauth-v2-bearer]
Jones, M., Hardt, D., and D. Recordon, "The OAuth 2.0 Authorization Protocol: Bearer Tokens", draft-ietf-oauth-v2-bearer-13 (work in progress), October 2011.
- [I-D.ietf-oauth-v2-http-mac]
Hammer-Lahav, E., Barth, A., and B. Adida, "HTTP Authentication: MAC Access Authentication", draft-ietf-oauth-v2-http-mac-00 (work in progress), May 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [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.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

- [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.
- [RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [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.

9.2. Informative References

- [I-D.hammer-hostmeta]
Hammer-Lahav, E. and B. Cook, "Web Host Metadata",
draft-hammer-hostmeta-17 (work in progress),
September 2011.
- [I-D.jones-simple-web-discovery]
Jones, M. and Y. Goland, "Simple Web Discovery (SWD)",
draft-jones-simple-web-discovery-01 (work in progress),
July 2011.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

Authors' Addresses

William Mills
Yahoo! Inc.

Phone:
Email: wmills@yahoo-inc.com

Tim Showalter
Yahoo! Inc.

Phone:
Email: timshow@yahoo-inc.com

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2011

T. Yu
MIT Kerberos Consortium
July 12, 2010

Desired changes to the GSS-API
draft-yu-kitten-api-wishlist-01

Abstract

Feedback from GSS-API implementors and application developers suggests that the API as it currently exists would benefit from improvements. This memo collects some specific suggestions of KITTEN WG participants.

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 January 13, 2011.

Copyright Notice

Copyright (c) 2010 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	3
2. Asynchronous calls	4
3. Error message reporting	5
4. Security strength reporting	6
5. Programmer friendliness	7
6. Security Considerations	8
Author's Address	9

1. Introduction

Experiences of GSS-API implementors and GSS-API application developers, particularly with the C bindings, suggest that the GSS-API would benefit from certain improvements. Some of these suggestions collected from the KITTEN working group include:

1. initialization/new credentials
2. listing/iterating credentials
3. exporting/importing credentials
4. error message reporting
5. asynchronous calls
6. security strength reporting
7. programmer friendliness

This summary is not complete; it is meant as a starting point.

2. Asynchronous calls

The desire for supporting asynchronous calls is a specific case of a generally accepted goal of increasing concurrency. Proponents of this goal typically note that new computers appear to be gaining more processor cores faster than they are gaining computing speed per core. Asynchronous calls (or event-based solutions) are an alternative to the traditional multi-threading model for increasing concurrency.

The existing C bindings say nothing about thread safety for the GSS-API. Implementors have considered various interpretations of thread safety, including using internal mutex locks within a GSS-API implementation to provide thread safety for callers. While the existing C bindings allow for such an approach, the traditional threaded programming model has its drawbacks.

In addition, some GSS-API mechanisms are nearly impossible to implement in a way that prevents `gss_init_sec_context` and such from blocking on I/O operations, particularly network I/O. An application that attempts to achieve high concurrency must dedicate a thread for each context establishment operation, for example. This can be problematic on platforms where threads are expensive.

Forcing callers to call into an event loop provided by GSS-API is not desirable.

3. Error message reporting

Existing GSS-API facilities for obtaining error information are limited to 32-bit major and minor status codes. This prevents callers from obtaining detailed (perhaps textual) information that may assist in troubleshooting. In addition, the existing GSS-API specifications do not have provisions for gracefully dealing with potentially conflicting minor status codes in multi-mechanism implementations, particularly ones that allow for runtime loading of GSS-API mechanisms.

Concrete approaches for improving GSS-API error reporting appear to be somewhat lacking, apart from the "PGSSAPI" proposal by Nico Williams, which adds semantics to the actual pointer value passed as the `minor_context` argument. Several working group participants find the "PGSSAPI" approach distasteful.

4. Security strength reporting

There is some interest in adding an interface to report the security strength of the established context, for use with implementations of protocols such as SASL. Some debate has taken place about whether a numeric report of security strength is an appropriate means of communicating this information to an application.

5. Programmer friendliness

In the GSS-API C bindings, the `gss_accept_sec_context` function takes 11 parameters, and the `gss_init_sec_context` function takes 13. Many of these parameters accept a default value, and in fact application developers sometimes unnecessarily provide non-default values, which often unintentionally results in reduced functionality.

Some programmers find that needing to explicitly loop over `gss_init_sec_context` and `gss_accept_sec_context` (as is currently required by a conforming GSS-API application during context establishment) is cumbersome. It may be beneficial to define a simpler interface for programmers who do not require the additional control afforded by explicitly calling the context establishment functions in a loop.

6. Security Considerations

Addition of an interface to report security strength of a GSS-API context enables applications to make better-informed decisions about security policy.

Author's Address

Tom Yu
MIT Kerberos Consortium
77 Massachusetts Ave
Building W92 Room 145
Cambridge, MA 02139
US

Phone: +1 617 253 1753
Email: tlyu@mit.edu

