ABFAB
Internet-Draft
Intended status: Standards Track
Expires: July 14, 2016

J. Howlett
Janet
S. Hartman
Painless Security
A. Perez-Mendez, Ed.
University of Murcia
January 11, 2016

A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML
draft-ietf-abfab-aaa-saml-14

Abstract

This document describes the use of the Security Assertion Mark-up Language (SAML) with RADIUS in the context of the ABFAB architecture. It defines two RADIUS attributes, a SAML binding, a SAML name identifier format, two SAML profiles, and two SAML confirmation methods. The RADIUS attributes permit encapsulation of SAML assertions and protocol messages within RADIUS, allowing SAML entities to communicate using the binding. The two profiles describe the application of this binding for ABFAB authentication and assertion query/request, enabling a Relying Party to request authentication of, or assertions for, users or machines (Clients). These Clients may be named using a NAI name identifier format. Finally, the subject confirmation methods allow requests and queries to be issued for a previously authenticated user or machine without needing to explicitly identify them as the subject. The use of the artifacts defined in this document is not exclusive to ABFAB. They can be applied in any AAA scenario, such as the network access control.

Status of This Memo

This Internet-Draft will expire on July 14, 2016.

Copyright Notice

Table of Contents

1. Introduction

Within the ABFAB (Application Bridging for Federated Access Beyond web) architecture [I-D.ietf-abfab-arch] it is often desirable to convey Security Assertion Mark-up Language (SAML) assertions and protocol messages.

SAML typically only considers the use of HTTP-based transports, known as bindings [OASIS.saml-bindings-2.0-os], which are primarily intended for use with the SAML V2.0 Web Browser Single Sign-On Profile [OASIS.saml-profiles-2.0-os]. However the goal of ABFAB is to extend the applicability of federated identity beyond the Web to other applications by building on the AAA framework. Consequently there exists a requirement for SAML to integrate with the AAA framework and protocols such as RADIUS [RFC2865] and Diameter [RFC6733], in addition to HTTP.

In summary this document specifies:

- Two RADIUS attributes to encapsulate SAML assertions and protocol messages respectively.
- A SAML RADIUS binding that defines how SAML assertions and protocol messages can be transported by RADIUS within a SAML exchange.
- A SAML name identifier format in the form of a Network Access Identifier.
- A profile of the SAML Authentication Request Protocol that uses the SAML RADIUS binding to effect SAML-based authentication and authorization.
- A profile of the SAML Assertion Query And Request Protocol that uses the SAML RADIUS binding to effect the query and request of SAML assertions.
- Two SAML Subject Confirmation Methods for indicating that a user or machine client is the subject of an assertion.

This document adheres to the guidelines stipulated by [OASIS.saml-bindings-2.0-os] and [OASIS.saml-profiles-2.0-os] for defining new SAML bindings and profiles respectively, and other conventions applied formally or otherwise within SAML. In particular, this document provides a 'Required Information' section for the binding and profiles that enumerate:

- A URI that uniquely identifies the protocol binding or profile.
- Postal or electronic contact information for the author.
- A reference to previously defined bindings or profiles that the new binding updates or obsoletes.
- In the case of a profile, any SAML confirmation method identifiers defined and/or utilized by the profile.

1.1. Terminology

This document uses terminology from a number of related standards, which tend to adopt different terms for similar or identical concepts. In general the document uses, when possible, the ABFAB term for the entity, as described in [I-D.ietf-abfab-arch]. For reference we include this table which maps the different terms into a single view.

Protocol	Client	Relying Party	Identity Provider
ABFAB	Client	Relying Party	Identity Provider
SAML	Subject Principal	Service Provider Requester Consumer	Identity Provider Responder Issuer
RADIUS	User	NAS RADIUS client	AS RADIUS server

Table 1. Terminology

2. Conventions

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].

3. RADIUS SAML Attributes

The RADIUS SAML binding defined in Section 4 of this document uses two attributes to convey SAML assertions and protocol messages [OASIS.saml-core-2.0-os]. Owing to the typical size of these structures, these attributes use the Long Extended Type format [RFC6929] to encapsulate their data. RADIUS entities MUST NOT include both attributes in the same RADIUS message, as they represent exclusive alternatives to convey SAML information.

3.1. SAML-Assertion attribute

This attribute is used to encode a SAML assertion. The following figure represents the format of this attribute.

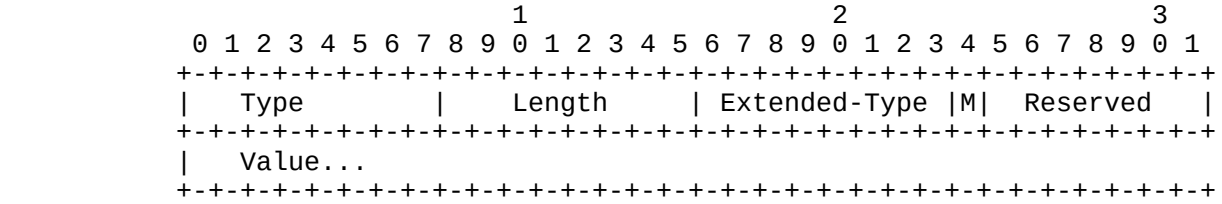


Figure 1: SAML-Assertion format

Type

245 (To be confirmed by IANA)

Length

>= 5

Extended-Type

TBD1

M (More)

As described in [RFC6929].

Reserved

As described in [RFC6929].

Value

One or more octets encoding a SAML assertion.

3.2. SAML-Protocol attribute

This attribute is used to encode a SAML protocol message. The following figure represents the format of this attribute.

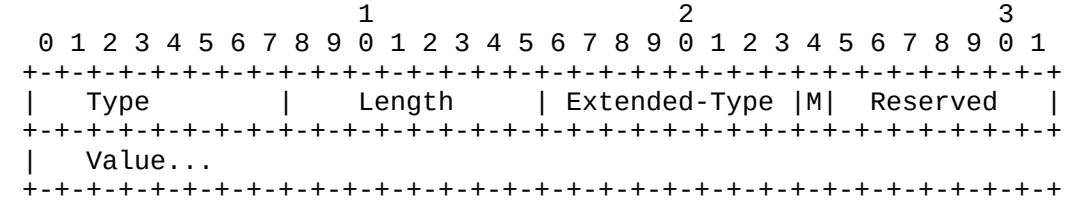


Figure 2: SAML-Protocol format

Type

245 (To be confirmed by IANA)

Length

>= 5

Extended-Type

TBD2

M (More)

As described in [RFC6929].

Reserved

As described in [RFC6929].

Value

One or more octets encoding a SAML protocol message.

4. SAML RADIUS Binding

The SAML RADIUS binding defines how RADIUS [RFC2865] can be used to enable a RADIUS client and server to exchange SAML assertions and protocol messages.

4.1. Required Information

Identification: urn:ietf:params:abfab:bindings:radius

Contact information: iesg@ietf.org

Updates: None.

4.2. Operation

In this specification, the Relying Party MUST trust any statement in the SAML messages from the IdP in the same way that it trusts information contained in RADIUS attributes. These entities MUST trust the RADIUS infrastructure to provide integrity of the SAML messages.

Hence, it is REQUIRED that the RADIUS exchange is protected using TLS encryption for RADIUS [RFC6614] to provide confidentiality and integrity protection, unless alternative methods to ensure them are used, such as IPSEC tunnels or a sufficiently secure internal network.

Implementations of this profile can take advantage of mechanisms to permit the transport of longer SAML messages over RADIUS transports, such as the Support of fragmentation of RADIUS packets [RFC7499] or Larger Packets for RADIUS over TCP [I-D.ietf-radext-bigger-packets].

There are two system models for the use of SAML over RADIUS. The first is a request-response model, using the RADIUS SAML-Protocol

attribute defined in Section 3 to encapsulate the SAML protocol messages.

1. The RADIUS client, acting as a Relying Party (RP), transmits a SAML request element within a RADIUS Access-Request message. This message MUST include a single instance of the RADIUS User-Name attribute whose value MUST conform to the Network Access Identifier [RFC7542] scheme. The Relying Party MUST NOT include more than one SAML request element.

2. The RADIUS server, acting as an Identity Provider (IdP), returns a SAML protocol message within a RADIUS Access-Accept or Access-Reject message. These messages necessarily conclude a RADIUS exchange and therefore this is the only opportunity for the Identity Provider to send a response in the context of this exchange. The Identity Provider MUST NOT include more than one SAML response. An IdP that refuses to perform a message exchange with the Relying Party can silently discard the SAML request (this could subsequently be followed by a RADIUS Access-Reject, as the same conditions that cause the IdP to discard the SAML request may also cause the RADIUS server to fail to authenticate).

The second system model permits a RADIUS server acting as an Identity Provider to use the RADIUS SAML-Assertion attribute defined in Section 3 to encapsulate an unsolicited SAML assertion. This attribute MUST be included in a RADIUS Access-Accept message. When included, the attribute MUST contain a single SAML assertion.

RADIUS servers MUST NOT include both the SAML-Protocol and the SAML-Assertion attribute in the same RADIUS message. If an IdP is producing a response to a SAML request, then the first system model is used. An IdP MAY ignore a SAML request and send an unsolicited assertion using the second system model using the RADIUS SAML-Assertion attribute.

In either system model, Identity Providers SHOULD return a RADIUS state attribute as part of the Access-Accept message so that future SAML queries or requests can be run against the same context of an authentication exchange.

This binding is intended to be composed with other uses of RADIUS, such as network access. Therefore, other arbitrary RADIUS attributes MAY be used in either the request or response.

In the case of a SAML processing error, the RADIUS server MAY include a SAML response message with an appropriate value for the <samlp:Status> element within the Access-Accept or Access-Reject

packet to notify the client. Alternatively, the RADIUS server can respond without a SAML-Protocol attribute.

4.3. Processing of names

SAML entities using profiles making use of this binding will typically possess both the SAML and AAA names of their correspondents. Frequently these entities will need to apply policies using these names; for example, when deciding to release attributes. Often these policies will be security-sensitive, and so it is important that policy is applied on these names consistently.

4.3.1. AAA names

These rules relate to the processing of AAA names by SAML entities using profiles making use of this binding.

- Identity Providers SHOULD apply policy based on the Relying Party's identity associated with the RADIUS Access-Request.

- Relying Parties SHOULD apply policy based on the NAI realm associated with the RADIUS Access-Accept.

4.3.2. SAML names

These rules relate to the processing of SAML names by SAML entities using profiles making use of this binding.

Identity Providers MAY apply policy based on the Relying Party's SAML entityId. In such cases, at least one of the following methods is required in order to establish a relation between the SAML name and the AAA name of the Relying Party:

- RADIUS client identity in trusted SAML metadata (as described in section Section 4.3.3).

- RADIUS client identity in trusted digitally signed SAML request.

A digitally signed SAML request without the RADIUS client identity is not sufficient, since a malicious RADIUS entity can observe a SAML message and include it in a different RADIUS message without the consent of the issuer of that SAML message. If an Identity Provider were to process the SAML message without confirming that it applied to the RADIUS message, inappropriate policy would be used.

Relying Parties MAY apply policy based on the SAML issuer's <entityId>. In such cases, at least one of the following methods is

required in order to establish a relationship between the SAML name and the AAA name of the Identity Provider:

- RADIUS realm in trusted SAML metadata (as described in section Section 4.3.3).
- RADIUS realm in trusted digitally signed SAML response or assertion.

A digitally signed SAML response alone is not sufficient for the same reasons described above for SAML requests.

4.3.3. Mapping of AAA names in SAML metadata

This section defines extensions to the SAML metadata schema [OASIS.saml-metadata-2.0-os] that are required in order to represent AAA names associated with a particular <EntityDescriptor> element.

In SAML metadata, a single entity may act in many different roles in the support of multiple profiles. This document defines two new roles: RADIUS IDP and RADIUS RP, requiring the declaration of two new subtypes of RoleDescriptorType: RADIUSIDPDescriptorType and RADIUSRPDescriptorType. These subtypes contain the additional elements required to represent AAA names for IDP and RP entities respectively.

4.3.3.1. RADIUSIDPDescriptorType

The RADIUSIDPDescriptorType complex type extends RoleDescriptorType with elements common to IdPs that support RADIUS. It contains the following additional elements:

<RADIUSIDPService> [Zero or More] Zero or more elements of type EndpointType that describe RADIUS endpoints that are associated with the entity.

<RADIUSRealm> [Zero or More] Zero or more elements of type string that represent the acceptable values of the RADIUS realm associated with the entity, obtained from the realm part of RADIUS User-Name attribute.

The following schema fragment defines the RADIUSIDPDescriptorType complex type:

```
        <complexType name="RADIUSIDPDescriptorType">
          <complexContent>
            <extension base="md:RoleDescriptorType">
              <sequence>
                <element ref="abfab:RADIUSIDPService" minOccurs="0" maxOccurs="u
nbounded"/>
                <element ref="abfab:RADIUSRealm" minOccurs="0" maxOccurs="unboun
ded"/>
              </sequence>
            </extension>
          </complexContent>
        </complexType>
        <element name="RADIUSIDPService" type="md:EndpointType"/>
        <element name="RADIUSRealm" type="string"/>
```

Figure 3: RADIUSIDPDescriptorType schema

4.3.3.2. RADIUSRPDescriptorType

The RADIUSRPDescriptorType complex type extends RoleDescriptorType with elements common to RPs that support RADIUS. It contains the following additional elements:

<RADIUSRPService> [Zero or More] Zero or more elements of type EndpointType that describe RADIUS endpoints that are associated with the entity.

<RADIUSNasIpAddress> [Zero or More] Zero or more elements of type string that represent the acceptable values of the RADIUS NAS-IP-Address or NAS-IPv6-Address attributes associated with the entity.

<RADIUSNasIdentifier> [Zero or More] Zero or more elements of type string that represent the acceptable values of the RADIUS NAS-Identifier attribute associated with the entity.

<RADIUSGssEapName> [Zero or More] Zero or more elements of type string that represent the acceptable values of the GSS-EAP acceptor name associated with the entity. The format for this name is described in section 3.1 of [RFC7055], while section 3.4 describes how that name is decomposed and transported using RADIUS attributes.

The following schema fragment defines the RADIUSRPDescriptorType complex type:

```
    <complexType name="RADIUSRPDescriptorType">
      <complexContent>
        <extension base="md:RoleDescriptorType">
          <sequence>
            <element ref="md:RADIUSRPService" minOccurs="0" maxOccurs="unbounded
"/>
            <element ref="md:RADIUSNasIpAddress" minOccurs="0" maxOccurs="unboun
ded"/>
            <element ref="md:RADIUSNasIdentifier" minOccurs="0" maxOccurs="unbou
nded"/>
            <element ref="md:RADIUSGssEapName" minOccurs="0" maxOccurs="unbounde
d"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
    <element name="RADIUSRPService" type="md:EndpointType"/>
    <element name="RADIUSNasIpAddress" type="string"/>
    <element name="RADIUSNasIdentifier" type="string"/>
    <element name="RADIUSGssEapName" type="string"/>
```

Figure 4: RADIUSRPDescriptorType schema

4.3.4. Example of SAML metadata including AAA names

The following figures illustrate an example of metadata including AAA names for and IDP and a RP respectively. The IDP's SAML name is "https://IdentityProvider.com/", whereas its RADIUS realm is "idp.com". The RP's SAML name is "https://RelyingParty.com/SAML", being its GSS-EAP acceptor name "nfs/fileserver.rp.com@RP.COM".

```
    <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:abfab="urn:ietf:params:xml:ns:abfab"
                      entityID="https://IdentityProvider.com/SAML">
        <RoleDescriptor xsi:type="abfab:RADIUSIDPDescriptorType"
              protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
            <RADIUSRealm>idp.com</RADIUSRealm>
        </RoleDescriptor>
    </EntityDescriptor>
```

Figure 5: Metadata for the IDP

```
    <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:abfab="urn:ietf:params:xml:ns:abfab"
                      entityID="https://RelyingParty.com/SAML">
        <RoleDescriptor xsi:type="abfab:RADIUSRPDescriptorType"
              protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
            <RADIUSGssEapName>nfs/fileserver.rp.com@RP.COM</RADIUSGssEapName>
        </RoleDescriptor>
    </EntityDescriptor>
```

Figure 6: Metadata for the RP

4.4. Use of XML Signatures

This binding calls for the use of SAML elements that support XML signatures. To promote interoperability, implementations of this binding MUST support a default configuration that does not require the use of XML signatures. Implementations MAY choose to use XML signatures.

4.5. Metadata Considerations

These binding and profiles are mostly intended to be used without metadata. In this usage, RADIUS infrastructure is used to provide integrity and naming of the SAML messages and assertions. RADIUS configuration is used to provide policy, including which attributes are accepted from a Relying Party and which attributes are sent by an Identity Provider.

Nevertheless, if metadata is used, the roles describe in section Section 4.3.3 MUST be present.

5. Network Access Identifier Name Identifier Format

URI: urn:ietf:params:abfab:nameid-format:nai

Indicates that the content of the element is in the form of a Network Access Identifier (NAI) using the syntax described by [RFC7542].

6. RADIUS State Confirmation Method Identifiers

URI: urn:ietf:params:abfab:cm:user

URI: urn:ietf:params:abfab:cm:machine

Indicates that the Subject is the system entity (either the user or machine) authenticated by a previously transmitted RADIUS Access-

Accept message, as identified by the value of that RADIUS message's State attribute.

7. ABFAB Authentication Profile

In the scenario supported by the ABFAB Authentication Profile, a Client controlling a User Agent requests access to a Relying Party. The Relying Party uses RADIUS to authenticate the Client. In particular, the Relying Party, acting as a RADIUS client, attempts to validate the Client's credentials against a RADIUS server acting as the Client's Identity Provider. If the Identity Provider successfully authenticates the Client, it produces an authentication assertion which is consumed by the Relying Party. This assertion MAY include a name identifier that can be used between the Relying Party and the Identity Provider to refer to the Client.

7.1. Required Information

Identification: urn:ietf:params:abfab:profiles:authentication

Contact information: iesg@ietf.org

SAML Confirmation Method Identifiers: The SAML V2.0 "RADIUS State" confirmation method identifiers, either urn:ietf:params:abfab:cm:user or urn:ietf:params:abfab:cm:machine, are used by this profile.

Updates: None.

7.2. Profile Overview

To implement this scenario, this profile of the SAML Authentication Request protocol MUST be used in conjunction with the SAML RADIUS binding defined in Section 4.

This profile is based on the SAML V2.0 Web Browser Single Sign-On Profile [OASIS.saml-profiles-2.0-os]. There are some important differences, specifically:

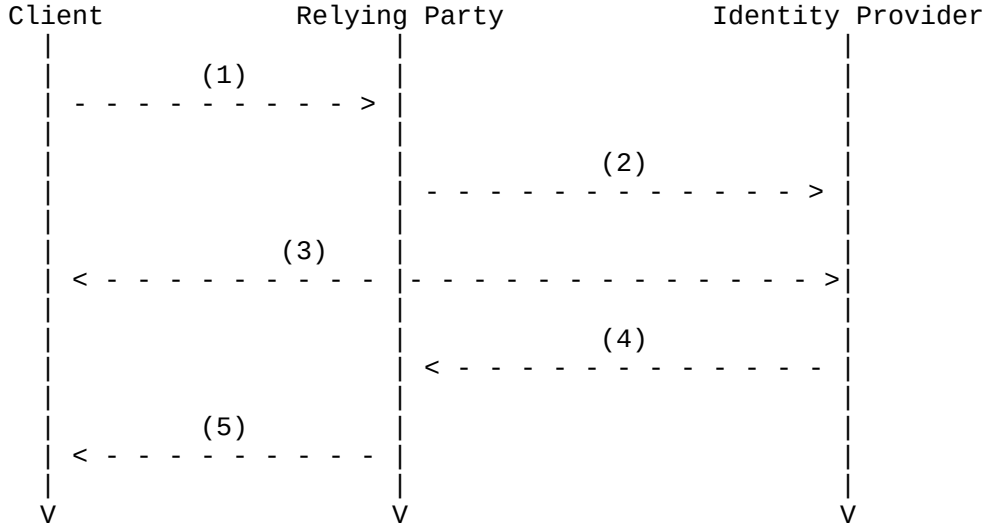
Authentication: This profile does not require the use of any particular authentication method. The ABFAB architecture does require the use of EAP [RFC3579], but this specification may be used in other non-ABFAB scenarios.

Bindings: This profile does not use HTTP-based bindings. Instead all SAML protocol messages are transported using the SAML RADIUS binding defined in Section 4. This is intended to reduce the number of bindings that implementations must support to be interoperable.

Requests: The profile does not permit the Relying Party to name the <saml:Subject> of the <samlp:AuthnRequest>. This is intended to simplify implementation and interoperability.

Responses: The profile only permits the Identity Provider to return a single SAML message or assertion that MUST contain exactly one authentication statement. Other statements may be included within this assertion at the discretion of the Identity Provider. This is intended to simplify implementation and interoperability.

Figure 7 below illustrates the flow of messages within this profile.



The following steps are described by the profile. Within an individual step, there may be one or more actual message exchanges.

Figure 7

1. Client request to Relying Party (Section 7.3.1): In step 1, the Client, via a User Agent, makes a request for a secured resource at the Relying Party. The Relying Party determines that no security context for the Client exists and initiates the authentication process.

2. Relying Party issues <samlp:AuthnRequest> to Identity Provider (Section 7.3.2). In step 2, the Relying Party may optionally issue a <samlp:AuthnRequest> message to be delivered to the Identity Provider using the SAML-Protocol RADIUS attribute.

3. Identity Provider identifies Client (Section 7.3.3). In step 3, the Client is authenticated and identified by the Identity Provider, while honoring any requirements imposed by the Relying Party in the <samlp:AuthnRequest> message if provided.

4. Identity Provider issues <samlp:Response> to Relying Party (Section 7.3.4). In step 4, the Identity Provider issues a <samlp:Response> message to the Relying Party using the SAML RADIUS binding. The response either indicates an error or includes a SAML Authentication Statement in exactly one SAML Assertion. If the RP did not send an <samlp:AuthnRequest>, the IdP issues an unsolicited <samlp:Assertion>, as described in Section 7.4.4.

5. Relying Party grants or denies access to Client (Section 7.3.5). In step 5, having received the response from the Identity Provider, the Relying Party can respond to the Client with its own error, or can establish its own security context for the Client and return the requested resource.

7.3. Profile Description

The ABFAB Authentication Profile is a profile of the SAML V2.0 Authentication Request Protocol [OASIS.saml-core-2.0-os]. Where both specifications conflict, the ABFAB Authentication Profile takes precedence.

7.3.1. Client Request to Relying Party

The profile is initiated by an arbitrary Client request to the Relying Party. There are no restrictions on the form of the request. The Relying Party is free to use any means it wishes to associate the subsequent interactions with the original request. The Relying Party, acting as a RADIUS client, attempts to authenticate the Client.

7.3.2. Relying Party Issues <samlp:AuthnRequest> to Identity Provider

The Relying Party uses RADIUS to communicate with the Client’s Identity Provider. The Relying Party MAY include a <samlp:AuthnRequest> within this RADIUS Access-Request message using the SAML-Protocol RADIUS attribute. The next hop destination MAY be the Identity Provider or alternatively an intermediate RADIUS proxy.

Profile-specific rules for the contents of the <samlp:AuthnRequest> element are given in Section 7.4.1.

7.3.3. Identity Provider Identifies Client

The Identity Provider MUST establish the identity of the Client using a RADIUS authentication method, or else it will return an error. If the ForceAuthn attribute on the <samlp:AuthnRequest> element (if sent by the Relying Party) is present and true, the Identity Provider MUST freshly establish this identity rather than relying on any existing session state it may have with the Client (for example, TLS state that may be used for session resumption). Otherwise, and in all other respects, the Identity Provider may use any method to authenticate the Client, subject to the constraints called out in the <samlp:AuthnRequest> message.

7.3.4. Identity Provider Issues <samlp:Response> to Relying Party

The Identity Provider MUST conclude the authentication in a manner consistent with the RADIUS authentication result. The IdP MAY issue a <samlp:Response> message to the Relying Party that is consistent with the authentication result, as described in [OASIS.saml-core-2.0-os]. This SAML response is delivered to the Relying Party using the SAML RADIUS binding described in Section 4.

Profile-specific rules regarding the contents of the <samlp:Response> element are given in Section 7.4.2.

7.3.5. Relying Party Grants or Denies Access to Client

If a <samlp:Response> message is issued by the Identity Provider, the Relying Party MUST process that message and any enclosed assertion elements as described in [OASIS.saml-core-2.0-os]. Any subsequent use of the assertion elements is at the discretion of the Relying Party, subject to any restrictions contained within the assertions themselves or from any previously established out-of-band policy that governs the interaction between the Identity Provider and the Relying Party.

7.4. Use of Authentication Request Protocol

This profile is based on the Authentication Request Protocol defined in [OASIS.saml-core-2.0-os]. In the nomenclature of actors enumerated in section 3.4 of that document, the Relying Party is the requester, the User Agent is the attesting entity and the Client is the Requested Subject.

7.4.1. <samlp:AuthnRequest> Usage

The Relying Party MUST NOT include a <saml:Subject> element in the request. The authenticated RADIUS identity identifies the Client to the Identity Provider.

A Relying Party MAY include any message content described in [OASIS.saml-core-2.0-os], section 3.4.1. All processing rules are as defined in [OASIS.saml-core-2.0-os].

If the Relying Party wishes to permit the Identity Provider to establish a new identifier for the Client if none exists, it MUST include a <saml:NameIDPolicy> element with the AllowCreate attribute set to "true". Otherwise, only a Client for whom the Identity Provider has previously established an identifier usable by the Relying Party can be authenticated successfully.

The <samlp:AuthnRequest> message MAY be signed. Authentication and integrity are also provided by the SAML RADIUS binding.

7.4.2. <samlp:Response> Message Usage

If the Identity Provider cannot or will not satisfy the request, it MUST either respond with a <samlp:Response> message containing an appropriate error status code or codes and/or respond with a RADIUS Access-Reject message.

If the Identity Provider wishes to return an error, it MUST NOT include any assertions in the <samlp:Response> message. Otherwise, if the request is successful (or if the response is not associated with a request), the <samlp:Response> element is subject to the following constraints:

- It MAY be signed.
- It MUST contain exactly one assertion. The <saml:Subject> element of this assertion MUST refer to the authenticated RADIUS user.
- The assertion MUST contain a <saml:AuthnStatement>. Besides, the assertion MUST contain a <saml:Subject> element with at least one <saml:SubjectConfirmation> element containing a Method of urn:ietf:params:abfab:cm:user or urn:ietf:params:abfab:cm:machine that reflects the authentication of the Client to the Identity Provider. Since the containing message is in response to an <samlp:AuthnRequest>, the InResponseTo attribute (both in the <saml:SubjectConfirmationData> and in the <saml:Response> elements) MUST match the request’s ID. The <saml:Subject> element

MAY use the NAI Name Identifier Format described in Section 5 to establish an identifier between the Relying Party and the IdP.

- Other conditions MAY be included as requested by the Relying Party or at the discretion of the Identity Provider. The Identity Provider is NOT obligated to honor the requested set of conditions in the <samlp:AuthnRequest>, if any.

7.4.3. <samlp:Response> Message Processing Rules

The Relying Party MUST do the following:

- Assume that the Client's identifier implied by a SAML <Subject> element, if present, takes precedence over an identifier implied by the RADIUS User-Name attribute.

- Verify that the InResponseTo attribute in the "RADIUS State" <saml:SubjectConfirmationData> equals the ID of its original <samlp:AuthnRequest> message, unless the response is unsolicited, in which case the attribute MUST NOT be present.

- If a <saml:AuthnStatement> used to establish a security context for the Client contains a SessionNotOnOrAfter attribute, the security context SHOULD be discarded once this time is reached, unless the Relying Party reestablishes the Client's identity by repeating the use of this profile.

- Verify that any assertions relied upon are valid according to processing rules in [OASIS.saml-core-2.0-os].

- Any assertion which is not valid, or whose subject confirmation requirements cannot be met MUST be discarded and MUST NOT be used to establish a security context for the Client.

7.4.4. Unsolicited Responses

An Identity Provider MAY initiate this profile by delivering an unsolicited assertion to a Relying Party. This MUST NOT contain any <saml:SubjectConfirmationData> elements containing an InResponseTo attribute.

7.4.5. Use of the SAML RADIUS Binding

It is RECOMMENDED that the RADIUS exchange is protected using TLS encryption for RADIUS [RFC6614] to provide confidentiality and integrity protection.

7.4.6. Use of XML Signatures

This profile calls for the use of SAML elements that support XML signatures. To promote interoperability implementations of this profile MUST NOT require the use of XML signatures. Implementations MAY choose to use XML signatures.

7.4.7. Metadata Considerations

There are no metadata considerations particular to this profile, aside from those applying to the use of the RADIUS binding.

8. ABFAB Assertion Query/Request Profile

This profile builds on the SAML V2.0 Assertion Query/Request Profile defined by [OASIS.saml-profiles-2.0-os]. That profile describes the use of the Assertion Query and Request Protocol defined by section 3.3 of [OASIS.saml-core-2.0-os] with synchronous bindings, such as the SOAP binding defined in [OASIS.saml-bindings-2.0-os].

While the SAML V2.0 Assertion Query/Request Profile is independent of the underlying binding, it is nonetheless useful to describe the use of the SAML RADIUS binding defined in Section 4 of this document, in the interests of promoting interoperable implementations, particularly as the SAML V2.0 Assertion Query/Request Profile is most frequently discussed and implemented in the context of the SOAP binding.

8.1. Required Information

Identification: urn:ietf:params:abfab:profiles:query

Contact information: iesg@ietf.org

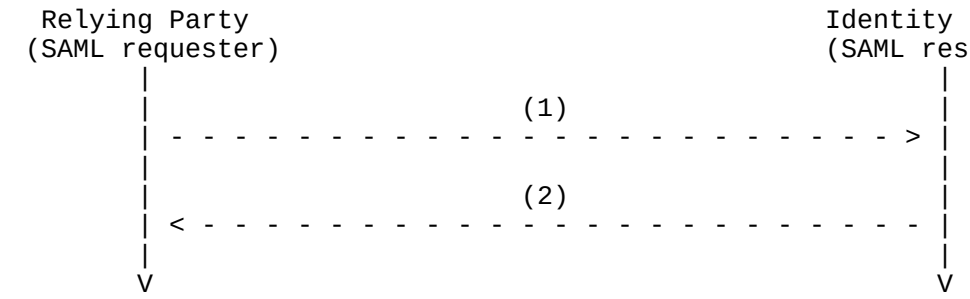
Description: Given below.

Updates: None.

8.2. Profile Overview

As with the SAML V2.0 Assertion Query/Request Profile defined by [OASIS.saml-profiles-2.0-os] the message exchange and basic processing rules that govern this profile are largely defined by Section 3.3 of [OASIS.saml-core-2.0-os] that defines the messages to be exchanged, in combination with the binding used to exchange the messages. The SAML RADIUS binding described in this document defines the binding of the message exchange to RADIUS. Unless specifically noted here, all requirements defined in those specifications apply.

Figure 8 below illustrates the basic template for the query/request profile.



The following steps are described by the profile.

Figure 8

1. Query/Request issued by Relying Party: In step 1, a Relying Party initiates the profile by sending an <AssertionIDRequest>, <SubjectQuery>, <AuthnQuery>, <AttributeQuery>, or <AuthzDecisionQuery> message to a SAML authority.

2. <Response> issued by SAML Authority: In step 2, the responding SAML authority (after processing the query or request) issues a <Response> message to the Relying Party.

8.3. Profile Description

8.3.1. Differences from the SAML V2.0 Assertion Query/Request Profile

This profile is identical to the SAML V2.0 Assertion Query/Request Profile, with the following exceptions:

- When processing the SAML request, the IdP MUST give precedence to the Client's identifier implied by RADIUS State attribute, if present, over the identifier implied by the SAML request's <Subject>, if any.

- In respect to sections 6.3.1 and 6.5 of [OASIS.saml-profiles-2.0-os], this profile does not consider the use of metadata (as in [OASIS.saml-metadata-2.0-os]). See Section 8.3.4.

- In respect to sections 6.3.2, 6.4.1, and 6.4.2 of [OASIS.saml-profiles-2.0-os], this profile additionally stipulates that implementations of this profile MUST NOT require the use of XML signatures. See Section 8.3.3.

8.3.2. Use of the SAML RADIUS Binding

The RADIUS Access-Request sent by the Relying Party:

- MUST include an instance of the RADIUS Service-Type attribute, having a value of Authorize-Only.
- SHOULD include the RADIUS State attribute, where this Query/ Request pertains to previously authenticated Client.

When processing the SAML request, the IdP MUST give precedence to the Client's identifier implied by RADIUS State attribute over the identifier implied by the SAML request's <Subject>, if any.

It is RECOMMENDED that the RADIUS exchange is protected using TLS encryption for RADIUS [RFC6614] to provide confidentiality and integrity protection.

8.3.3. Use of XML Signatures

This profile calls for the use of SAML elements that support XML signatures. To promote interoperability implementations of this profile MUST NOT require the use of XML signatures. Implementations MAY choose to use XML signatures.

8.3.4. Metadata Considerations

There are no metadata considerations particular to this profile, aside from those applying to the use of the RADIUS binding.

9. Privacy considerations

The profiles defined in this document allow a Relying Party to request specific information about the Client, and allow an IdP to disclose information about that Client. In this sense, Identity Providers MUST apply policy to decide what information is released to a particular Relying Party. Moreover, the identity of the Client is typically hidden from the Relying Party unless informed by the Identity Provider. Conversely, the Relying Party does typically know the realm of the IdP, as it is required to route the RADIUS packets to the right destination.

The kind of information that is released by the IdP can include generic attributes such as affiliation shared by many Clients. But even these generic attributes can help to identify a specific Client. Other kinds of attributes may also provide a Relying Party with the ability to link the same Client between different sessions. Finally, other kind of attributes might provide a group of Relying Parties

with the ability to link the Client between them or with personally identifiable information about the Client.

These profiles do not directly provide a Client with a mechanism to express preferences about what information is released. That information can be expressed out-of-band, for example as part of the enrollment process.

The Relying Party may disclose privacy-sensitive information about itself as part of the request, although this is unlikely in typical deployments.

If RADIUS proxies are used and encryption is not used, the attributes disclosed by the IdP are visible to the proxies. This is a significant privacy exposure in some deployments. Ongoing work is exploring mechanisms for creating TLS connections directly between the RADIUS client and the RADIUS server to reduce this exposure. If proxies are used, the impact of exposing SAML assertions to the proxies needs to be carefully considered.

The use of TLS to provide confidentiality for the RADIUS exchange is strongly encouraged. Without this, passive eavesdroppers can observe the assertions.

10. Security Considerations

In this specification, the Relying Party MUST trust any statement in the SAML messages from the IdP in the same way that it trusts information contained in RADIUS attributes. These entities MUST trust the RADIUS infrastructure to provide integrity of the SAML messages.

Furthermore, the Relying Party MUST apply policy and filter the information based on what information the IdP is permitted to assert and on what trust is reasonable to place in proxies between them.

XML signatures and encryption are provided as an OPTIONAL mechanism for end-to-end security. These mechanism can protect SAML messages from being modified by proxies in the RADIUS infrastructure. These mechanisms are not mandatory-to-implement. It is believed that ongoing work to provide direct TLS connections between a RADIUS client and RADIUS server will provide similar assurances but better deployability. XML security is appropriate for deployments where end-to-end security is required but proxies cannot be removed or where SAML messages need to be verified at a later time or by parties not involved in the authentication exchange.

11. IANA Considerations

11.1. RADIUS Attributes

The authors request that Attribute Types and Attribute Values defined in this document be registered by the Internet Assigned Numbers Authority (IANA) from the RADIUS namespaces as described in the "IANA Considerations" section of [RFC3575], in accordance with BCP 26 [RFC5226]. For RADIUS packets, attributes and registries created by this document IANA is requested to place them at http://www.iana.org/assignments/radius-types.

In particular, this document defines two new RADIUS attributes, entitled "SAML-Assertion" and "SAML-Protocol" (see Section 3), with assigned values of 245.TBD1 and 245.TBD2 from the Long Extended Space of [RFC6929]:

Type	Ext. Type	Name	Length	Meaning
245	TBD1	SAML-Assertion	>=5	Encodes a SAML assertion
245	TBD2	SAML-Protocol	>=5	Encodes a SAML protocol message

11.2. ABFAB Parameters

A new top-level registry is created titled "ABFAB Parameters".

In this top-level registry, a sub-registry titled "ABFAB URN Parameters" is created. Registration in this registry is by the IETF review or expert review procedures [RFC5226].

This paragraph gives guidance to designated experts. Registrations in this registry are generally only expected as part of protocols published as RFCs on the IETF stream; other URIs are expected to be better choices for non-IETF work. Expert review is permitted mainly to allow early registration related to specifications under development when the community believes they have reached sufficient maturity. The expert SHOULD evaluate the maturity and stability of such an IETF-stream specification. Experts SHOULD review anything not from the IETF stream for consistency and consensus with current practice. Today such requests would not typically be approved.

If a parameter named "paramname" is to be registered in this registry, then its URN will be "urn:ietf:params:abfab:paramname". The initial registrations are as follows:

Parameter	Reference
bindings:radius	Section 4
nameid-format:nai	Section 5
profiles:authentication	Section 7
profiles:query	Section 8
cm:user	Section 6
cm:machine	Section 6

ABFAB Parameters

11.3. Registration of the ABFAB URN Namespace

IANA is requested to register the "abfab" URN sub-namespace in the IETF URN sub-namespace for protocol parameters defined in [RFC3553].

Registry Name: abfab

Specification: draft-ietf-abfab-aaa-saml

Repository: ABFAB URN Parameters (Section Section 11.2)

Index Value: Sub-parameters MUST be specified in UTF-8 using standard URI encoding where necessary.

12. Acknowledgements

The authors would like to acknowledge the OASIS Security Services (SAML) Technical Committee, and Scott Cantor in particular, for their help with the SAML-related material.

The authors would also like to acknowledge the collaboration of Jim Schaad, Leif Johansson, Klaas Wierenga, Stephen Farell, Gabriel Lopez, and Rafael Marin, who have provided valuable comments on this document.

13. References

13.1. 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>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <http://www.rfc-editor.org/info/rfc2865>.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <http://www.rfc-editor.org/info/rfc3579>.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, DOI 10.17487/RFC6614, May 2012, <http://www.rfc-editor.org/info/rfc6614>.

[RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, DOI 10.17487/RFC6929, April 2013, <http://www.rfc-editor.org/info/rfc6929>.

[RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, DOI 10.17487/RFC3575, July 2003, <http://www.rfc-editor.org/info/rfc3575>.

[RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <http://www.rfc-editor.org/info/rfc7542>.

[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.

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

13.2. Informative References

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <http://www.rfc-editor.org/info/rfc3553>.

[RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <http://www.rfc-editor.org/info/rfc6733>.

[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>.

[RFC7055] Hartman, S., Ed. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", RFC 7055, DOI 10.17487/RFC7055, December 2013, <http://www.rfc-editor.org/info/rfc7055>.

[RFC7499] Perez-Mendez, A., Ed., Marin-Lopez, R., Pereniguez-Garcia, F., Lopez-Millan, G., Lopez, D., and A. DeKok, "Support of Fragmentation of RADIUS Packets", RFC 7499, DOI 10.17487/RFC7499, April 2015, <http://www.rfc-editor.org/info/rfc7499>.

[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-13 (work in progress), July 2014.

[I-D.ietf-radext-bigger-packets]
Hartman, S., "Larger Packets for RADIUS over TCP", draft-ietf-radext-bigger-packets-05 (work in progress), December 2015.

[W3C.REC-xmlschema-1]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

Appendix A. XML Schema

The following schema formally defines the "urn:ietf:params:xml:ns:abfab" namespace used in this document, in conformance with [W3C.REC-xmlschema-1] While XML validation is optional, the schema that follows is the normative definition of the constructs it defines. Where the schema differs from any prose in this specification, the schema takes precedence.

```
        <schema
          targetNamespace="urn:ietf:params:xml:ns:abfab"
          xmlns="http://www.w3.org/2001/XMLSchema"
          xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
          xmlns:abfab="urn:ietf:params:xml:ns:abfab"
          elementFormDefault="unqualified"
          attributeFormDefault="unqualified"
          blockDefault="substitution"
          version="1.0">

          <import namespace="urn:oasis:names:tc:SAML:2.0:metadata"/>

          <complexType name="RADIUSIDPDescriptorType">
            <complexContent>
              <extension base="md:RoleDescriptorType">
                <sequence>
                  <element ref="abfab:RADIUSIDPService" minOccurs="0" maxOccurs=
"unbounded"/>
                  <element ref="abfab:RADIUSRealm" minOccurs="0" maxOccurs="unbo
unded"/>
                </sequence>
              </extension>
            </complexContent>
          </complexType>
          <element name="RADIUSIDPService" type="md:EndpointType"/>
          <element name="RADIUSRealm" type="string"/>

          <complexType name="RADIUSRPDescriptorType">
            <complexContent>
              <extension base="md:RoleDescriptorType">
                <sequence>
                  <element ref="md:RADIUSRPService" minOccurs="0" maxOccurs="unb
ounded"/>
                  <element ref="md:RADIUSNasIpAddress" minOccurs="0" maxOccurs="
unbounded"/>
                  <element ref="md:RADIUSNasIdentifier" minOccurs="0" maxOccurs=
"unbounded"/>
                  <element ref="md:RADIUSGssEapName" minOccurs="0" maxOccurs="un
bounded"/>
                </sequence>
              </extension>
            </complexContent>
          </complexType>
          <element name="RADIUSRPService" type="md:EndpointType"/>
          <element name="RADIUSNasIpAddress" type="string"/>
          <element name="RADIUSNasIdentifier" type="string"/>
          <element name="RADIUSGssEapName" type="string"/>

        </schema>
```

Authors' Addresses

Josh Howlett
Janet
Lumen House, Library Avenue, Harwell
Oxford OX11 0SG
UK

Phone: +44 1235 822363
EMail: Josh.Howlett@ja.net

Sam Hartman
Painless Security

EMail: hartmans-ietf@mit.edu

Alejandro Perez-Mendez (editor)
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 46 44
EMail: alex@um.es

ABFAB
Internet-Draft
Intended status: Informational
Expires: January 22, 2015

J. Howlett
JANET(UK)
S. Hartman
Painless Security
H. Tschofenig
ARM Ltd.
E. Lear
Cisco Systems GmbH
J. Schaad
Soaring Hawk Consulting

July 21, 2014

Application Bridging for Federated Access Beyond Web (ABFAB) Architecture
draft-ietf-abfab-arch-13.txt

Abstract

Over the last decade a substantial amount of work has occurred in the space of federated access management. Most of this effort has focused on two use cases: network access and web-based access. However, the solutions to these use cases that have been proposed and deployed tend to have few building blocks in common.

This memo describes an architecture that makes use of extensions to the commonly used security mechanisms for both federated and non-federated access management, including the Remote Authentication Dial In User Service (RADIUS) the Generic Security Service Application Program Interface (GSS-API), the Extensible Authentication Protocol (EAP) and the Security Assertion Markup Language (SAML). The architecture addresses the problem of federated access management to primarily non-web-based services, in a manner that will scale to large numbers of identity providers, relying parties, and federations.

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 22, 2015.

Table of Contents

1. Introduction

Numerous security mechanisms have been deployed on the Internet to manage access to various resources. These mechanisms have been generalized and scaled over the last decade through mechanisms such as Simple Authentication and Security Layer (SASL) with the Generic Security Server Application Program Interface (GSS-API) (known as the GS2 family) [RFC5801], Security Assertion Markup Language (SAML) [OASIS.saml-core-2.0-os], and the Authentication, Authorization, and Accounting (AAA) architecture as embodied in RADIUS [RFC2865] and Diameter [RFC6733].

A Relying Party (RP) is the entity that manages access to some resource. The entity that is requesting access to that resource is often described as the Client. Many security mechanisms are manifested as an exchange of information between these entities. The RP is therefore able to decide whether the Client is authorized, or not.

Some security mechanisms allow the RP to delegate aspects of the access management decision to an entity called the Identity Provider (IdP). This delegation requires technical signaling, trust and a common understanding of semantics between the RP and IdP. These aspects are generally managed within a relationship known as a 'federation'. This style of access management is accordingly described as 'federated access management'.

Federated access management has evolved over the last decade through specifications like SAML [OASIS.saml-core-2.0-os], OpenID [1], OAuth [RFC6749] and WS-Trust [WS-TRUST]. The benefits of federated access management include:

Single or Simplified sign-on:

An Internet service can delegate access management, and the associated responsibilities such as identity management and credentialing, to an organization that already has a long-term relationship with the Client. This is often attractive as Relying Parties frequently do not want these responsibilities. The Client also requires fewer credentials, which is also desirable.

Data Minimization and User Participation:

Often a Relying Party does not need to know the identity of a Client to reach an access management decision. It is frequently only necessary for the Relying Party to know specific attributes about the client, for example, that the client is affiliated with a particular organization or has a certain role or entitlement. Sometimes the RP only needs to know a pseudonym of the client.

Prior to the release of attributes to the RP from the IdP, the IdP will check configuration and policy to determine if the attributes are to be released. There is currently no direct client participation in this decision.

Provisioning:

Sometimes a Relying Party needs, or would like, to know more about a client than an affiliation or a pseudonym. For example, a Relying Party may want the Client’s email address or name. Some federated access management technologies provide the ability for the IdP to supply this information, either on request by the RP or unsolicited.

This memo describes the Application Bridging for Federated Access Beyond the Web (ABFAB) architecture. The architecture addresses the problem of federated access management primarily for non-web-based services. This architecture makes use of extensions to the commonly used security mechanisms for both federated and non-federated access management, including RADIUS, the Generic Security Service (GSS), the Extensible Authentication Protocol (EAP) and SAML. The architecture should be extended to use Diameter in the future. It does so in a manner that designed to scale to large numbers of identity providers, relying parties, and federations.

1.1. Terminology

This document uses identity management and privacy terminology from [RFC6973]. In particular, this document uses the terms identity provider, relying party, identifier, pseudonymity, unlinkability, and anonymity.

In this architecture the IdP consists of the following components: an EAP server, a RADIUS server, and optionally a SAML Assertion service.

This document uses the term Network Access Identifier (NAI), as defined in [I-D.ietf-radext-nai]. An NAI consists of a realm identifier, which is associated with an AAA server and thus an IdP and a username which is associated with a specific client of the IdP.

One of the problems some people have found with reading this document is that the terminology sometimes appears to be inconsistent. This is due the fact that the terms used by the different standards we are referencing are not consistent with each other. In general the document uses either the ABFAB term or the term associated with the standard under discussion as appropriate. For reference we include this table which maps the different terms into a single table.

Protocol	Client	Relying Party	Identity Provider
ABFAB	Client	Relying Party (RP)	Identity Provider (IdP)
	Initiator	Acceptor	
		Server	
SAML	Subject	Service Provider	Issuer
GSS-API	Initiator	Acceptor	
EAP	EAP peer	EAP Authenticator	EAP server
AAA		AAA Client	AAA server
RADIUS	user	NAS	RADIUS server
		RADIUS client	

Table 1. Terminology

Note that in some cases a cell has been left empty; in these cases there is no name that represents the entity.

1.1.1. Channel Binding

This document uses the term channel binding in two different contexts. The term channel binding has a different meaning in each of these contexts.

EAP channel binding is used to implement GSS-API naming semantics. EAP 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 protocol from the EAP authenticator to the EAP server. The EAP server confirms the consistency of these attributes and provides the confirmation back to the peer. In this document, channel binding without qualification refers to EAP channel binding.

GSS-API channel binding provides protection against man-in-the-middle attacks when GSS-API is used for authentication inside of some tunnel; it is similar to a facility called cryptographic binding in EAP. The binding works by each side deriving a cryptographic value from the tunnel itself and then using that cryptographic value to prove to the other side that it knows the value.

See [RFC5056] for a discussion of the differences between these two facilities. However, the difference can be summarized as GSS-API channel binding says that there is nobody between the client and the EAP authenticator while EAP channel binding allows the client to have knowledge about attributes of the EAP authenticator (such as its name).

Typically when considering both EAP and GSS-API channel binding, people think of channel binding in combination with mutual authentication. This is sufficiently common that without additional qualification channel binding should be assumed to imply mutual authentication. In GSS-API, without mutual authentication only the acceptor has authenticated the initiator. Similarly in EAP, only the EAP server has authenticated the peer. That’s sometimes useful. Consider for example a user who wishes to access a protected resource for a shared whiteboard in a conference room. The whiteboard is the acceptor; it knows that the initiator is authorized to give it a presentation and the user can validate the whitebord got the correct presentation by visual means. (The presention should not be confidential in this case.) If channel binding is used without mutual authentication, it is effectively a request to disclose the resource in the context of a particular channel. Such an authentication would be similar in concept to a holder-of-key SAML

assertion. However, also note that while it is not happening in the protocol, mutual authentication is happening in the overall system: the user is able to visually authenticate the content. This is consistent with all uses of channel binding without protocol level mutual authentication found so far.

1.2. An Overview of Federation

In the previous section we introduced the following entities:

- the Client,
- the Identity Provider, and
- the Relying Party.

The final entity that needs to be introduced is the Individual. An Individual is a human being that is using the Client. In any given situation, an Individual may or may not exist. Clients can act either as front ends for Individuals or they may be independent entities that are setup and allowed to run autonomously. An example of such an independent entity can be found in the trust routing protocol [2] where the routers use ABFAB to authenticate to each other.

These entities and their relationships are illustrated graphically in Figure 1.

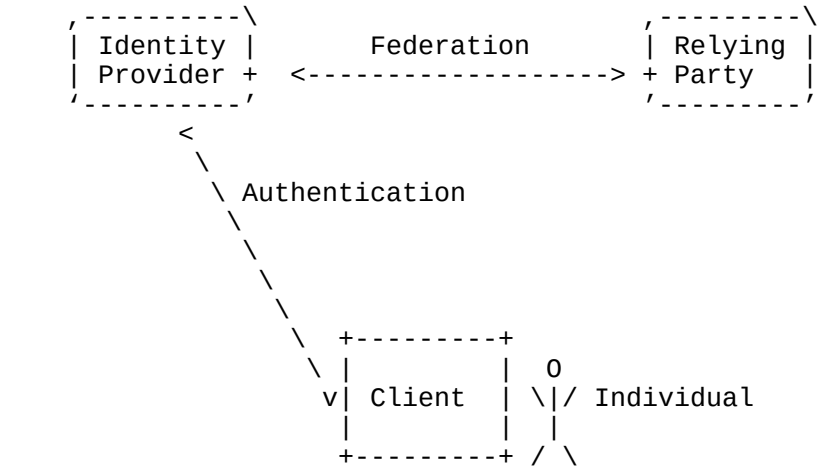


Figure 1: Entities and their Relationships

The relationships between the entities in Figure 1 are:

Federation

The Identity Provider and the Relying Parties are part of a Federation. The relationship may be direct (they have an explicit trust relationship) or transitive (the trust relationship is mediated by one or more entities). The federation relationship is governed by a federation agreement. Within a single federation, there may be multiple Identity Providers as well as multiple Relying Parties.

Authentication

There is a direct relationship between the Client and the Identity Provider. This relationship provides the means by which they trust each other and can securely authenticate each other.

A federation agreement typically encompasses operational specifications and legal rules:

Operational Specifications:

The goal of operational specifications is to provide enough definition that the system works and interoperability is possible. These include the technical specifications (e.g. protocols used to communicate between the three parties), process standards, policies, identity proofing, credential and authentication algorithm requirements, performance requirements, assessment and audit criteria, etc.

Legal Rules:

The legal rules take the legal framework into consideration and provide contractual obligations for each entity. The rules define the responsibilities of each party and provide further clarification of the operational specifications. These legal rules regulate the operational specifications, make operational specifications legally binding to the participants, define and govern the rights and responsibilities of the participants. The legal rules may, for example, describe liability for losses, termination rights, enforcement mechanisms, measures of damage, dispute resolution, warranties, etc.

The Operational Specifications can demand the usage of a specific technical infrastructure, including requirements on the message routing intermediaries, to offer the required technical functionality. In other environments, the Operational Specifications

require fewer technical components in order to meet the required technical functionality.

The Legal Rules include many non-technical aspects of federation, such as business practices and legal arrangements, which are outside the scope of the IETF. The Legal Rules can still have an impact on the architectural setup or on how to ensure the dynamic establishment of trust.

While a federation agreement is often discussed within the context of formal relationships, such as between an enterprise and an employee or a government and a citizen, a federation agreement does not have to require any particular level of formality. For an IdP and a Client, it is sufficient for a relationship to be established by something as simple as using a web form and confirmation email. For an IdP and an RP, it is sufficient for the IdP to publish contact information along with a public key and for the RP to use that data. Within the framework of ABFAB, it will generally be required that a mechanism exists for the IdP to be able to trust the identity of the RP, if this is not present then the IdP cannot provide the assurances to the client that the identity of the RP has been established.

The nature of federation dictates that there is some form of relationship between the identity provider and the relying party. This is particularly important when the relying party wants to use information obtained from the identity provider for access management decisions and when the identity provider does not want to release information to every relying party (or only under certain conditions).

While it is possible to have a bilateral agreement between every IdP and every RP; on an Internet scale this setup requires the introduction of the multi-lateral federation concept, as the management of such pair-wise relationships would otherwise prove burdensome.

The IdP will typically have a long-term relationship with the Client. This relationship typically involves the IdP positively identifying and credentialing the Client (for example, at time of employment within an organization). When dealing with individuals, this process is called identity proofing [NIST-SP.800-63]. The relationship will often be instantiated within an agreement between the IdP and the Client (for example, within an employment contract or terms of use that stipulates the appropriate use of credentials and so forth).

The nature and quality of the relationship between the Client and the IdP is an important contributor to the level of trust that an RP may assign to an assertion describing a Client made by an IdP. This is sometimes described as the Level of Assurance [NIST-SP.800-63].

Federation does not require an a priori relationship or a long-term relationship between the RP and the Client; it is this property of federation that yields many of the federation benefits. However, federation does not preclude the possibility of a pre-existing relationship between the RP and the Client, nor that they may use the introduction to create a new long-term relationship independent of the federation.

Finally, it is important to reiterate that in some scenarios there might indeed be an Individual behind the Client and in other cases the Client may be autonomous.

1.3. Challenges for Contemporary Federation

As the number of federated IdPs and RPs (services) proliferats, the role of the individual can become ambiguous in certain circumstances. For example, a school might provide online access for a student’s grades to their parents for review, and to the student’s teacher for modification. A teacher who is also a parent must clearly distinguish her role upon access.

Similarly, as the number of federations proliferates, it becomes increasingly difficult to discover which identity provider(s) a user is associated with. This is true for both the web and non-web case, but is particularly acute for the latter as many non-web authentication systems are not semantically rich enough on their own to allow for such ambiguities. For instance, in the case of an email provider, the SMTP and IMAP protocols do not have the ability for the server to request information from the client, beyond the clients NAI, that the server would then use to decide between the multiple federations it is associated with. However, the building blocks do exist to add this functionality.

1.4. An Overview of ABFAB-based Federation

The previous section described the general model of federation, and the application of access management within the federation. This section provides a brief overview of ABFAB in the context of this model.

In this example, a client is attempting to connect to a server in order to either get access to some data or perform some type of transaction. In order for the client to mutually authenticate with

the server, the following steps are taken in an ABFAB architecture (a graphical view of the steps can be found in figure Figure 2):

1. Client Configuration: The Client Application is configured with an NAI assigned by the IdP. It is also configured with any keys, certificates, passwords or other secret and public information needed to run the EAP protocols between it and the IdP.

2. Authentication mechanism selection: The Client Application is configured to use the GSS-EAP GSS-API mechanism for authentication/authorization.

3. Client provides an NAI to RP: The client application sets up a transport to the RP and begins the GSS-EAP authentication. In response, the RP sends an EAP request message (nested in the GSS-EAP protocol) asking for the Client's name. The Client sends an EAP response with an NAI name form that, at a minimum, contains the realm portion of its full NAI.

4. Discovery of federated IdP: The RP uses pre-configured information or a federation proxy to determine what IdP to use based on policy and the realm portion of the provided Client NAI. This is discussed in detail below (Section 2.1.2).

5. Request from Relying Party to IdP: Once the RP knows who the IdP is, it (or its agent) will send a RADIUS request to the IdP. The RADIUS access request encapsulates the EAP response. At this stage, the RP will likely have no idea who the client is. The RP sends its identity to the IdP in AAA attributes, and it may send a SAML Attribute Request in a AAA attribute. The AAA network checks that the identity claimed by the RP is valid.

6. IdP begins EAP with the client: The IdP sends an EAP message to the client with an EAP method to be used. The IdP should not re-request the clients name in this message, but clients need to be able to handle it. In this case the IdP must accept a realm only in order to protect the client's name from the RP. The available and appropriate methods are discussed below in this memo (Section 2.2.1).

7. The EAP protocol is run: A bunch of EAP messages are passed between the client (EAP peer) and the IdP (EAP server), until the result of the authentication protocol is determined. The number and content of those messages depends on the EAP method selected. If the IdP is unable to authenticate the client, the IdP sends an EAP failure message to the RP. As part of the EAP protocol, the client sends a channel bindings EAP message to the

IdP (Section 2.2.2). In the channel binding message the client identifies, among other things, the RP to which it is attempting to authenticate. The IdP checks the channel binding data from the client with that provided by the RP via the AAA protocol. If the bindings do not match the IdP sends an EAP failure message to the RP.

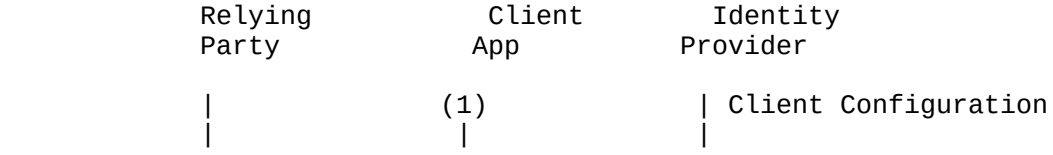
8. Successful EAP Authentication: At this point, the IdP (EAP server) and client (EAP peer) have mutually authenticated each other. As a result, the client and the IdP hold two cryptographic keys: a Master Session Key (MSK), and an Extended MSK (EMSK). At this point the client has a level of assurance about the identity of the RP based on the name checking the IdP has done using the RP naming information from the AAA framework and from the client (by the channel binding data).

9. Local IdP Policy Check: At this stage, the IdP checks local policy to determine whether the RP and client are authorized for a given transaction/service, and if so, what if any, attributes will be released to the RP. If the IdP gets a policy failure, it sends an EAP failure message to the RP and client. (The RP will have done its policy checks during the discovery process.)

10. IdP provides the RP with the MSK: The IdP sends a positive result EAP to the RP, along with an optional set of AAA attributes associated with the client (usually as one or more SAML assertions). In addition, the EAP MSK is returned to the RP.

11. RP Processes Results: When the RP receives the result from the IdP, it should have enough information to either grant or refuse a resource access request. It may have information that associates the client with specific authorization identities. If additional attributes are needed from the IdP the RP may make a new SAML Request to the IdP. It will apply these results in an application-specific way.

12. RP returns results to client: Once the RP has a response it must inform the client application of the result. If all has gone well, all are authenticated, and the application proceeds with appropriate authorization levels. The client can now complete the authentication of the RP by the use of the EAP MSK value.



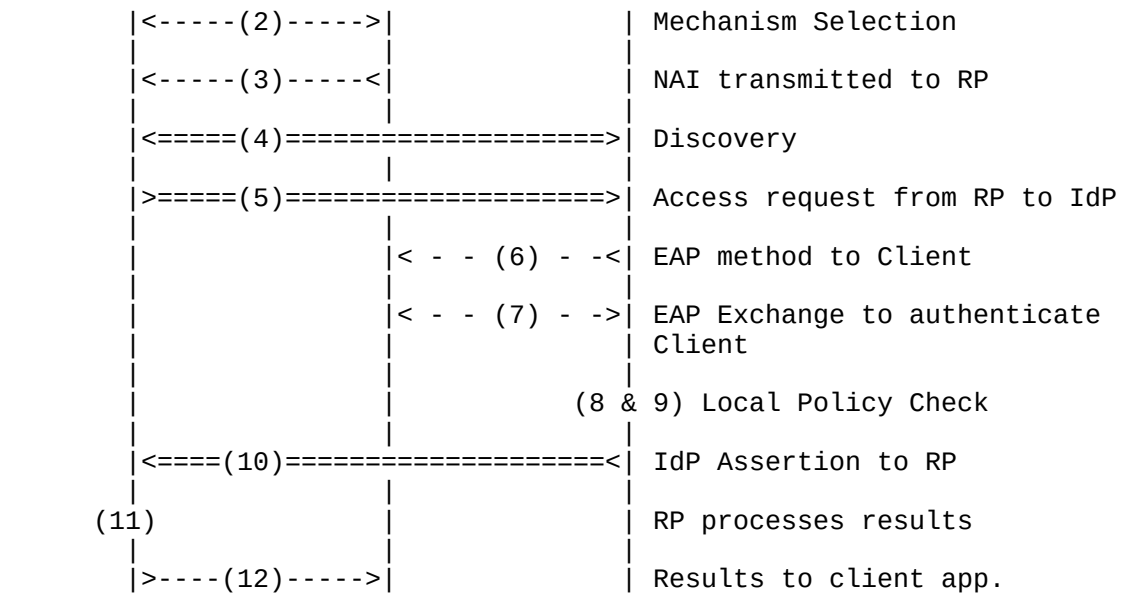


Figure 2: ABFAB Authentication Steps

1.5. Design Goals

Our key design goals are as follows:

- Each party in a transaction will be authenticated, although perhaps not identified, and the client will be authorized for access to a specific resource.
- Means of authentication is decoupled from the application protocol so as to allow for multiple authentication methods with minimal changes to the application.
- The architecture requires no sharing of long term private keys between clients and RPs.
- The system will scale to large numbers of identity providers, relying parties, and users.

- The system will be designed primarily for non-Web-based authentication.

- The system will build upon existing standards, components, and operational practices.

Designing new three party authentication and authorization protocols is hard and fraught with risk of cryptographic flaws. Achieving widespread deployment is even more difficult. A lot of attention on federated access has been devoted to the Web. This document instead focuses on a non-Web-based environment and focuses on those protocols where HTTP is not used. Despite the growing trend to layer every protocol on top of HTTP there are still a number of protocols available that do not use HTTP-based transports. Many of these protocols are lacking a native authentication and authorization framework of the style shown in Figure 1.

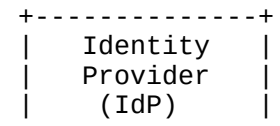
2. Architecture

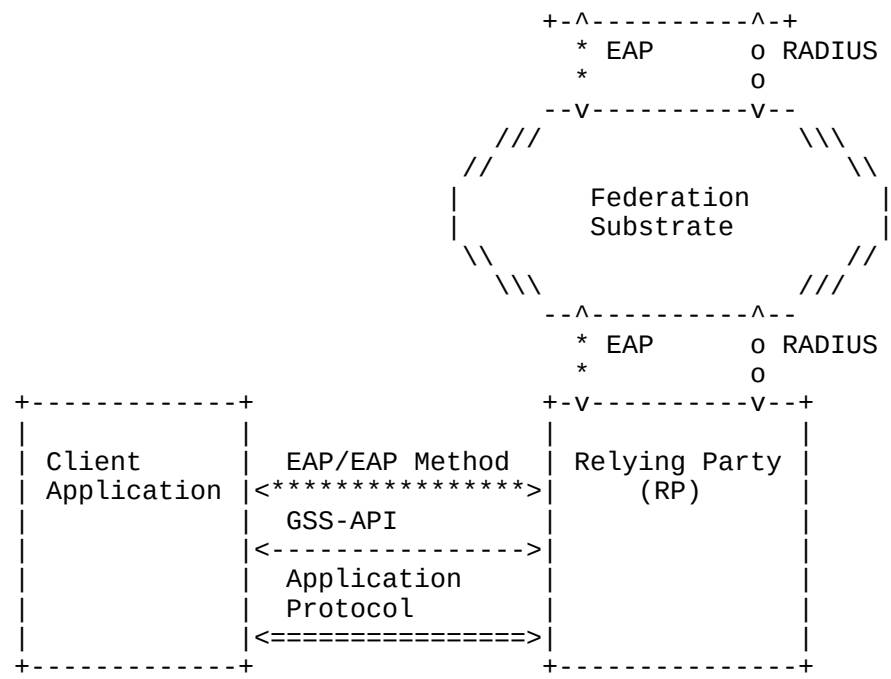
We have already introduced the federated access architecture, with the illustration of the different actors that need to interact, but did not expand on the specifics of providing support for non-Web based applications. This section details this aspect and motivates design decisions. The main theme of the work described in this document is focused on re-using existing building blocks that have been deployed already and to re-arrange them in a novel way.

Although this architecture assumes updates to the relying party, the client application, and the IdP, those changes are kept at a minimum. A mechanism that can demonstrate deployment benefits (based on ease of update of existing software, low implementation effort, etc.) is preferred and there may be a need to specify multiple mechanisms to support the range of different deployment scenarios.

There are a number of ways to encapsulate EAP into an application protocol. For ease of integration with a wide range of non-Web based application protocols, GSS-API was chosen. The technical specification of GSS-EAP can be found in [RFC7055].

The architecture consists of several building blocks, which is shown graphically in Figure 3. In the following sections, we discuss the data flow between each of the entities, the protocols used for that data flow and some of the trade-offs made in choosing the protocols.





Legend:

<****>: Client-to-IdP Exchange
<---->: Client-to-RP Exchange
<oooo>: RP-to-IdP Exchange
<====>: Protocol through which GSS-API/GS2 exchanges are tunneled

Figure 3: ABFAB Protocol Instantiation

2.1. Relying Party to Identity Provider

Communications between the Relying Party and the Identity Provider is done by the federation substrate. This communication channel is responsible for:

- Establishing the trust relationship between the RP and the IdP.
- Determining the rules governing the relationship.
- Conveying authentication packets from the client to the IdP and back.
- Providing the means of establishing a trust relationship between the RP and the client.

- Providing a means for the RP to obtain attributes about the client from the IdP.

The ABFAB working group has chosen the AAA framework for the messages transported between the RP and IdP. The AAA framework supports the requirements stated above as follows:

- The AAA backbone supplies the trust relationship between the RP and the IdP.
- The agreements governing a specific AAA backbone contains the rules governing the relationships within the AAA federation.
- A method exists for carrying EAP packets within RADIUS [RFC3579] and Diameter [RFC4072].
- The use of EAP channel binding [RFC6677] along with the core ABFAB protocol provide the pieces necessary to establish the identities of the RP and the client, while EAP provides the cryptographic methods for the RP and the client to validate they are talking to each other.
- A method exists for carrying SAML packets within RADIUS [I-D.ietf-abfab-aaa-saml] which allows the RP to query attributes about the client from the IdP.

Protocols that support the same framework, but do different routing are expected to be defined and used the future. One such effort call the Trust Router is to setup a framework that creates a trusted point-to-point channel on the fly [3].

2.1.1. AAA, RADIUS and Diameter

The usage of the AAA framework with RADIUS [RFC2865] and Diameter [RFC6733] for network access authentication has been successful from a deployment point of view. To map to the terminology used in Figure 1 to the AAA framework the IdP corresponds to the AAA server, the RP corresponds to the AAA client, and the technical building blocks of a federation are AAA proxies, relays and redirect agents (particularly if they are operated by third parties, such as AAA brokers and clearing houses). The front-end, i.e. the end host to AAA client communication, is in case of network access authentication offered by link layer protocols that forward authentication protocol exchanges back-and-forth. An example of a large scale RADIUS-based federation is EDUROAM [4].

By using the AAA framework, ABFAB can be built on the federation agreements already exist, the agreements can then merely be expanded

to cover the ABFAB. The AAA framework has already addressed some of the problems outlined above. For example,

- It already has a method for routing requests based on a domain.
- It already has an extensible architecture allowing for new attributes to be defined and transported.
- Pre-existing relationships can be re-used.

The astute reader will notice that RADIUS and Diameter have substantially similar characteristics. Why not pick one? RADIUS and Diameter are deployed in different environments. RADIUS can often be found in enterprise and university networks, and is also in use by fixed network operators. Diameter, on the other hand, is deployed by mobile operators. Another key difference is that today RADIUS is largely transported upon UDP. We leave as a deployment decision, which protocol will be appropriate. The protocol defines all the necessary new AAA attributes as RADIUS attributes. A future document could define the same AAA attributes for a Diameter environment. We also note that there exist proxies which convert from RADIUS to Diameter and back. This makes it possible for both to be deployed in a single federation substrate.

Through the integrity protection mechanisms in the AAA framework, the identity provider can establish technical trust that messages are being sent by the appropriate relying party. Any given interaction will be associated with one federation at the policy level. The legal or business relationship defines what statements the identity provider is trusted to make and how these statements are interpreted by the relying party. The AAA framework also permits the relying party or elements between the relying party and identity provider to make statements about the relying party.

The AAA framework provides transport for attributes. Statements made about the client by the identity provider, statements made about the relying party and other information are transported as attributes.

One demand that the AAA substrate makes of the upper layers is that they must properly identify the end points of the communication. It must be possible for the AAA client at the RP to determine where to send each RADIUS or Diameter message. Without this requirement, it would be the RP's responsibility to determine the identity of the client on its own, without the assistance of an IdP. This architecture makes use of the Network Access Identifier (NAI), where the IdP is indicated by the realm component [I-D.ietf-radext-nai]. The NAI is represented and consumed by the GSS-API layer as GSS_C_NT_USER_NAME as specified in [RFC2743]. The GSS-API EAP mechanism includes the NAI in the EAP Response/Identity message.

As of the time this document was published, no profiles for the use of Diameter have been created.

2.1.2. Discovery and Rules Determination

While we are using the AAA protocols to communicate with the IdP, the RP may have multiple federation substrates to select from. The RP has a number of criteria that it will use in selecting which of the different federations to use:

- The federation selected must be able to communicate with the IdP.
- The federation selected must match the business rules and technical policies required for the RP security requirements.

The RP needs to discover which federation will be used to contact the IdP. The first selection criterion used during discovery is going to be the name of the IdP to be contacted. The second selection criteria used during discovery is going to be the set of business rules and technical policies governing the relationship; this is called rules determination. The RP also needs to establish technical trust in the communications with the IdP.

Rules determination covers a broad range of decisions about the exchange. One of these is whether the given RP is permitted to talk to the IdP using a given federation at all, so rules determination encompasses the basic authorization decision. Other factors are included, such as what policies govern release of information about the client to the RP and what policies govern the RP's use of this information. While rules determination is ultimately a business function, it has significant impact on the technical exchanges. The protocols need to communicate the result of authorization. When multiple sets of rules are possible, the protocol must disambiguate which set of rules are in play. Some rules have technical enforcement mechanisms; for example in some federations intermediaries validate information that is being communicated within the federation.

At the time of writing no protocol mechanism has been specified to allow a AAA client to determine whether a AAA proxy will indeed be able to route AAA requests to a specific IdP. The AAA routing is impacted by business rules and technical policies that may be quite complex and at the present time, the route selection is based on manual configuration.

2.1.3. Routing and Technical Trust

Several approaches to having messages routed through the federation substrate are possible. These routing methods can most easily be classified based on the mechanism for technical trust that is used. The choice of technical trust mechanism constrains how rules determination is implemented. Regardless of what deployment strategy is chosen, it is important that the technical trust mechanism be able to validate the identities of both parties to the exchange. The trust mechanism must ensure that the entity acting as IdP for a given NAI is permitted to be the IdP for that realm, and that any service name claimed by the RP is permitted to be claimed by that entity. Here are the categories of technical trust determination:

AAA Proxy:
: The simplest model is that an RP is an AAA client and can send the request directly to an AAA proxy. The hop-by-hop integrity protection of the AAA fabric provides technical trust. An RP can submit a request directly to the correct federation. Alternatively, a federation disambiguation fabric can be used. Such a fabric takes information about what federations the RP is part of and what federations the IdP is part of and routes a message to the appropriate federation. The routing of messages across the fabric plus attributes added to requests and responses provides rules determination. For example, when a disambiguation fabric routes a message to a given federation, that federation's

rules are chosen. Name validation is enforced as messages travel across the fabric. The entities near the RP confirm its identity and validate names it claims. The fabric routes the message towards the appropriate IdP, validating the name of the IdP in the process. The routing can be statically configured. Alternatively a routing protocol could be developed to exchange reachability information about a given IdP and to apply policy across the AAA fabric. Such a routing protocol could flood naming constraints to the appropriate points in the fabric.

Trust Broker:
Instead of routing messages through AAA proxies, some trust broker could establish keys between entities near the RP and entities near the IdP. The advantage of this approach is efficiency of message handling. Fewer entities are needed to be involved for each message. Security may be improved by sending individual messages over fewer hops. Rules determination involves decisions made by trust brokers about what keys to grant. Also, associated with each credential is context about rules and about other aspects of technical trust including names that may be claimed. A routing protocol similar to the one for AAA proxies is likely to be useful to trust brokers in flooding rules and naming constraints.

Global Credential:
A global credential such as a public key and certificate in a public key infrastructure can be used to establish technical trust. A directory or distributed database such as the Domain Name System is used by the RP to discover the endpoint to contact for a given NAI. Either the database or certificates can provide a place to store information about rules determination and naming constraints. Provided that no intermediates are required (or appear to be required) and that the RP and IdP are sufficient to enforce and determine rules, rules determination is reasonably simple. However applying certain rules is likely to be quite complex. For example if multiple sets of rules are possible between an IdP and RP, confirming the correct set is used may be difficult. This is particularly true if intermediates are involved in making the decision. Also, to the extent that directory information needs to be trusted, rules determination may be more complex.

Real-world deployments are likely to be mixtures of these basic approaches. For example, it will be quite common for an RP to route traffic to a AAA proxy within an organization. That proxy could then use any of the three methods to get closer to the IdP. It is also likely that rather than being directly reachable, the IdP may have a proxy on the edge of its organization. Federations will likely

provide a traditional AAA proxy interface even if they also provide another mechanism for increased efficiency or security.

2.1.4. AAA Security

For the AAA framework there are two different places where security needs to be examined. The first is the security that is in place for the links in the AAA backbone being used. The second are the nodes that form the AAA backbone.

The default link security for RADIUS is showing its age as it uses MD5 and a shared secret to both obfuscate passwords and to provide integrity on the RADIUS messages. While some EAP methods include the ability to protect the client authentication credentials, the MSK returned from the IdP to the RP is protected only by the RADIUS security. In many environments this is considered to be insufficient, especially as not all attributes are obfuscated and can thus leak information to a passive eavesdropper. The use of RADIUS with TLS [RFC6614] and/or DTLS [I-D.ietf-radext-dtls] addresses these attacks. The same level of security is included in the base Diameter specifications.

2.1.5. SAML Assertions

For the traditional use of AAA frameworks, network access, an affirmative response from the IdP can be sufficient to grant access. In the ABFAB world, the RP may need to get significantly more additional information about the client before granting access. ABFAB therefore has a requirement that it can transport an arbitrary set of attributes about the client from the IdP to the RP.

Security Assertions Markup Language (SAML) [OASIS.saml-core-2.0-os] was designed in order to carry an extensible set of attributes about a subject. Since SAML is extensible in the attribute space, ABFAB has no immediate needs to update the core SAML specifications for our work. It will be necessary to update IdPs that need to return SAML assertions to RPs and for both the IdP and the RP to implement a new SAML profile designed to carry SAML assertions in AAA. The new profile can be found in RFCXXXX [I-D.ietf-abfab-aaa-saml]. As SAML statements will frequently be large, RADIUS servers and clients that deal with SAML statements will need to implement RFC XXXX [I-D.ietf-radext-radius-fragmentation]

There are several issues that need to be highlighted:

- The security of SAML assertions.
- Namespaces and mapping of SAML attributes.

- Subject naming of entities.
- Making multiple queries about the subject(s).
- Level of Assurance for authentication.

SAML assertions have an optional signature that can be used to protect and provide origination of the assertion. These signatures are normally based on asymmetric key operations and require that the verifier be able to check not only the cryptographic operation, but also the binding of the originators name and the public key. In a federated environment it will not always be possible for the RP to validate the binding, for this reason the technical trust established in the federation is used as an alternate method of validating the origination and integrity of the SAML Assertion.

Attributes in a SAML assertion are identified by a name string. The name string is either assigned by the SAML issuer context or is scoped by a namespace (for example a URI or object identifier (OID)). This means that the same attribute can have different name strings used to identify it. In many, but not all, cases the federation agreements will determine what attributes and names can be used in a SAML statement. This means that the RP needs to map from the SAML issuer or federation name, type and semantic into the name, type and semantics that the policies of the RP are written in. In other cases the federation substrate, in the form of proxies, will modify the SAML assertions in transit to do the necessary name, type and value mappings as the assertion crosses boundaries in the federation. If the proxies are modifying the SAML Assertion, then they will remove any signatures on the SAML as changing the content of the SAML statement would invalidate the signature. In this case the technical trust is the required mechanism for validating the integrity of the assertion. (The proxy could re-sign the SAML assertion, but the same issues of establishing trust in the proxy would still exist.) Finally, the attributes may still be in the namespace of the originating IdP. When this occurs the RP will need to get the required mapping operations from the federation agreements and do the appropriate mappings itself.

The RADIUS SAML RFC [I-D.ietf-abfab-aaa-saml] has defined a new SAML name format that corresponds to the NAI name form defined by RFC XXXX [I-D.ietf-radext-nai]. This allows for easy name matching in many cases as the name form in the SAML statement and the name form used in RADIUS or Diameter will be the same. In addition to the NAI name form, the document also defines a pair of implicit name forms corresponding to the Client and the Client’s machine. These implicit name forms are based on the Identity-Type enumeration defined in TEAP [I-D.ietf-emu-eap-tunnel-method]. If the name form returned in a

SAML statement is not based on the NAI, then it is a requirement on the EAP server that it validate that the subject of the SAML assertion, if any, is equivalent to the subject identified by the NAI used in the RADIUS or Diameter session.

RADIUS has the ability to deal with multiple SAML queries for those EAP Servers which follow RFC 5080 [RFC5080]. In this case a State attribute will always be returned with the Access-Accept. The EAP client can then send a new Access-Request with the State attribute and the new SAML Request Multiple SAML queries can then be done by making a new Access-Request using the State attribute returned in the last Access-Accept to link together the different RADIUS sessions.

Some RPs need to ensure that specific criteria are met during the authentication process. This need is met by using Levels of Assurance. The way a Level of Assurance is communicated to the RP from the EAP server is by the use of a SAML Authentication Request using the Authentication Profile from RFC XXX [I-D.ietf-abfab-aaa-saml] When crossing boundaries between different federations, either the policy specified will need to be shared between the two federations, the policy will need to be mapped by the proxy server on the boundary or the proxy server on the boundary will need to supply information the EAP server so that it can do the required mapping. If this mapping is not done, then the EAP server will not be able to enforce the desired Level of Assurance as it will not understand the policy requirements.

2.2. Client To Identity Provider

Looking at the communications between the client and the IdP, the following items need to be dealt with:

- The client and the IdP need to mutually authenticate each other.
- The client and the IdP need to mutually agree on the identity of the RP.

ABFAB selected EAP for the purposes of mutual authentication and assisted in creating some new EAP channel binding documents for dealing with determining the identity of the RP. A framework for the channel binding mechanism has been defined in RFC 6677 [RFC6677] that allows the IdP to check the identity of the RP provided by the AAA framework with that provided by the client.

2.2.1. Extensible Authentication Protocol (EAP)

Traditional web federation does not describe how a client interacts with an identity provider for authentication. As a result, this

communication is not standardized. There are several disadvantages to this approach. Since the communication is not standardized, it is difficult for machines to recognize which entity is going to do the authentication and thus which credentials to use and where in the authentication form that the credentials are to be entered. Humans have a much easier time to correctly deal with these problems. The use of browsers for authentication restricts the deployment of more secure forms of authentication beyond plaintext username and password known by the server. In a number of cases the authentication interface may be presented before the client has adequately validated they are talking to the intended server. By giving control of the authentication interface to a potential attacker, the security of the system may be reduced and phishing opportunities introduced.

As a result, it is desirable to choose some standardized approach for communication between the client's end-host and the identity provider. There are a number of requirements this approach must meet.

Experience has taught us one key security and scalability requirement: it is important that the relying party not get possession of the long-term secret of the client. Aside from a valuable secret being exposed, a synchronization problem can develop when the client changes keys with the IdP.

Since there is no single authentication mechanism that will be used everywhere there is another associated requirement: The authentication framework must allow for the flexible integration of authentication mechanisms. For instance, some IdPs require hardware tokens while others use passwords. A service provider wants to provide support for both authentication methods, and other methods from IdPs not yet seen.

These requirements can be met by utilizing standardized and successfully deployed technology, namely by the Extensible Authentication Protocol (EAP) framework [RFC3748]. Figure 3 illustrates the integration graphically.

EAP is an end-to-end framework; it provides for two-way communication between a peer (i.e. client or individual) through the EAP authenticator (i.e., relying party) to the back-end (i.e., identity provider). Conveniently, this is precisely the communication path that is needed for federated identity. Although EAP support is already integrated in AAA systems (see [RFC3579] and [RFC4072]) several challenges remain:

- The first is how to carry EAP payloads from the end host to the relying party.

- Another is to verify statements the relying party has made to the client, confirm these statements are consistent with statements made to the identity provider and confirm all of the above are consistent with the federation and any federation-specific policy or configuration.

- Another challenge is choosing which identity provider to use for which service.

The EAP method used for ABFAB needs to meet the following requirements:

- It needs to provide mutual authentication of the client and IdP.

- It needs to support channel binding.

As of this writing, the only EAP method that meets these criteria is TEAP [I-D.ietf-emu-eap-tunnel-method] either alone (if client certificates are used) or with an inner EAP method that does mutual authentication.

2.2.2. EAP Channel Binding

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.

The client knows, in theory, the name of the RP that it attempted to connect to, however in the event that an attacker has intercepted the protocol, the client and the IdP need to be able to detect this situation. A general overview of the problem along with a recommended way to deal with the channel binding issues can be found in RFC 6677 [RFC6677].

Since that document was published, a number of possible attacks were found and methods to address these attacks have been outlined in [RFC7029].

2.3. Client to Relying Party

The final set of interactions between the parties to consider are those between the client and the RP. In some ways this is the most complex set since at least part of it is outside the scope of the ABFAB work. The interactions between these parties include:

- Running the protocol that implements the service that is provided by the RP and desired by the client.
- Authenticating the client to the RP and the RP to the client.
- Providing the necessary security services to the service protocol that it needs beyond authentication.
- Deal with client re-authentication where desired.

2.3.1. GSS-API

One of the remaining layers is responsible for integration of federated authentication into the application. There are a number of approaches that applications have adopted for security. So, there may need to be multiple strategies for integration of federated authentication into applications. However, we have started with a strategy that provides integration to a large number of application protocols.

Many applications such as SSH [RFC4462], NFS [RFC2203], DNS [RFC3645] and several non-IETF applications support the Generic Security Services Application Programming Interface [RFC2743]. Many applications such as IMAP, SMTP, XMPP and LDAP support the Simple Authentication and Security Layer (SASL) [RFC4422] framework. These two approaches work together nicely: by creating a GSS-API mechanism, SASL integration is also addressed. In effect, using a GSS-API mechanism with SASL simply requires placing some headers on the front of the mechanism and constraining certain GSS-API options.

GSS-API is specified in terms of an abstract set of operations which can be mapped into a programming language to form an API. When people are first introduced to GSS-API, they focus on it as an API. However, from the prospective of authentication for non-web applications, GSS-API should be thought of as a protocol as well as an API. When looked at as a protocol, it consists of abstract operations such as the initial context exchange, which includes two sub-operations (gss_init_sec_context and gss_accept_sec_context). An application defines which abstract operations it is going to use and where messages produced by these operations fit into the application architecture. A GSS-API mechanism will define what actual protocol messages result from that abstract message for a given abstract operation. So, since this work is focusing on a particular GSS-API mechanism, we generally focus on protocol elements rather than the API view of GSS-API.

The API view of GSS-API does have significant value as well, since the abstract operations are well defined, the set of information that

a mechanism gets from the application is well defined. Also, the set of assumptions the application is permitted to make is generally well defined. As a result, an application protocol that supports GSS-API or SASL is very likely to be usable with a new approach to authentication including this one with no required modifications. In some cases, support for a new authentication mechanism has been added using plugin interfaces to applications without the application being modified at all. Even when modifications are required, they can often be limited to supporting a new naming and authorization model. For example, this work focuses on privacy; an application that assumes it will always obtain an identifier for the client will need to be modified to support anonymity, unlinkability or pseudonymity.

So, we use GSS-API and SASL because a number of the application protocols we wish to federate support these strategies for security integration. What does this mean from a protocol standpoint and how does this relate to other layers? This means we need to design a concrete GSS-API mechanism. We have chosen to use a GSS-API mechanism that encapsulates EAP authentication. So, GSS-API (and SASL) encapsulates EAP between the end-host and the service. The AAA framework encapsulates EAP between the relying party and the identity provider. The GSS-API mechanism includes rules about how initiators and services are named as well as per-message security and other facilities required by the applications we wish to support.

2.3.2. Protocol Transport

The transport of data between the client and the relying party is not provided by GSS-API. GSS-API creates and consumes messages, but it does not provide the transport itself, instead the protocol using GSS-API needs to provide the transport. In many cases HTTP or HTTPS is used for this transport, but other transports are perfectly acceptable. The core GSS-API document [RFC2743] provides some details on what requirements exist.

In addition we highlight the following:

- The transport does not need to provide either confidentiality or integrity. After GSS-EAP has finished negotiation, GSS-API can be used to provide both services. If the negotiation process itself needs protection from eavesdroppers then the transport would need to provide the necessary services.

- The transport needs to provide reliable transport of the messages.

- The transport needs to ensure that tokens are delivered in order during the negotiation process.

- o GSS-API messages need to be delivered atomically. If the transport breaks up a message it must also reassemble the message before delivery.

2.3.3. Reauthentication

There are circumstances where the RP will want to have the client reauthenticate itself. These include very long sessions, where the original authentication is time limited or cases where in order to complete an operation a different authentication is required. GSS-EAP does not have any mechanism for the server to initiate a reauthentication as all authentication operations start from the client. If a protocol using GSS-EAP needs to support reauthentication that is initiated by the server, then a request from the server to the client for the reauthentiction to start needs to be placed in the protocol.

Clients can re-use the existing secure connection established by GSS-API to run the new authentication in by calling GSS_Init_sec_context. At this point a full reauthentication will be done.

3. Application Security Services

One of the key goals is to integrate federated authentication into existing application protocols and where possible, existing implementations of these protocols. Another goal is to perform this integration while meeting the best security practices of the technologies used to perform the integration. This section describes security services and properties required by the EAP GSS-API mechanism in order to meet these goals. This information could be viewed as specific to that mechanism. However, other future application integration strategies are very likely to need similar services. So, it is likely that these services will be expanded across application integration strategies if new application integration strategies are adopted.

3.1. Authentication

GSS-API provides an optional security service called mutual authentication. This service means that in addition to the initiator providing (potentially anonymous or pseudonymous) identity to the acceptor, the acceptor confirms its identity to the initiator. Especially for the ABFAB context, this service is confusingly named. We still say that mutual authentication is provided when the identity of an acceptor is strongly authenticated to an anonymous initiator.

RFC 2743, unfortunately, does not explicitly talk about what mutual authentication means. Within this document we therefore define mutual authentication as:

- If a target name is configured for the initiator, then the initiator trusts that the supplied target name describes the acceptor. This implies both that appropriate cryptographic exchanges took place for the initiator to make such a trust decision, and that after evaluating the results of these exchanges, the initiator's policy trusts that the target name is accurate.

- If no target name is configured for the initiator, then the initiator trusts that the acceptor name, supplied by the acceptor, correctly names the entity it is communicating with.

- Both the initiator and acceptor have the same key material for per-message keys and both parties have confirmed they actually have the key material. In EAP terms, there is a protected indication of success.

Mutual authentication is an important defense against certain aspects of phishing. Intuitively, clients would like to assume that if some party asks for their credentials as part of authentication, successfully gaining access to the resource means that they are talking to the expected party. Without mutual authentication, the server could "grant access" regardless of what credentials are supplied. Mutual authentication better matches this user intuition.

It is important, therefore, that the GSS-EAP mechanism implement mutual authentication. That is, an initiator needs to be able to request mutual authentication. When mutual authentication is requested, only EAP methods capable of providing the necessary service can be used, and appropriate steps need to be taken to provide mutual authentication. While a broader set of EAP methods could be supported by not requiring mutual authentication, it was decided that the client needs to always have the ability to request it. In some cases the IdP and the RP will not support mutual authentication, however the client will always be able to detect this and make an appropriate security decision.

The AAA infrastructure may hide the initiator's identity from the GSS-API acceptor, providing anonymity between the initiator and the acceptor. At this time, whether the identity is disclosed is determined by EAP server policy rather than by an indication from the initiator. Also, initiators are unlikely to be able to determine whether anonymous communication will be provided. For this reason, initiators are unlikely to set the anonymous return flag from GSS_Init_Sec_context (Section 4.2.1 in [RFC4178].

3.2. GSS-API Channel Binding

[RFC5056] defines a concept of channel binding which is used prevent man-in-the-middle attacks. The channel binding works by taking a cryptographic value from the transport security and checks that both sides of the GSS-API conversation know this value. Transport Layer Security (TLS) [RFC5246] is the most common transport security layer used for this purpose.

It needs to be stressed that RFC 5056 channel binding (also called GSS-API channel binding when GSS-API is involved) is not the same thing as EAP channel binding. GSS-API channel binding is used for detecting Man-In-The-Middle attacks. EAP channel binding is used for mutual authentication and acceptor naming checks. Details are discussed in the mechanisms specification [RFC7055]. A fuller description of the differences between the facilities can be found in RFC 5056 [RFC5056].

The use of TLS can provide both encryption and integrity on the channel. It is common to provide SASL and GSS-API with these other security services.

One of the benefits that the use of TLS provides, is that client has the ability to validate the name of the server. However this validation is predicated on a couple of things. The TLS sessions needs to be using certificates and not be an anonymous session. The client and the TLS server need to share a common trust point for the certificate used in validating the server. TLS provides its own server authentication. However there are a variety of situations where this authentication is not checked for policy or usability reasons. When the TLS authentication is checked, if the trust infrastructure behind the TLS authentication is different from the trust infrastructure behind the GSS-API mutual authentication then confirming the end-points using both trust infrastructures is likely to enhance security. If the endpoints of the GSS-API authentication are different than the endpoints of the lower layer, this is a strong indication of a problem such as a man-in-the-middle attack. Channel binding provides a facility to determine whether these endpoints are the same.

The GSS-EAP mechanism needs to support channel binding. When an application provides channel binding data, the mechanism needs to confirm this is the same on both sides consistent with the GSS-API specification.

3.3. Host-Based Service Names

IETF security mechanisms typically take a host name and perhaps a service, entered by a user, and make some trust decision about whether the remote party in the interaction is the intended party. This decision can be made by the use of certificates, pre-configured key information or a previous leap of trust. GSS-API has defined a relatively flexible name convention, however most of the IETF applications that use GSS-API (including SSH, NFS, IMAP, LDAP and XMPP) have chosen to use a more restricted naming convention based on the host name. The GSS-EAP mechanism needs to support host-based service names in order to work with existing IETF protocols.

The use of host-based service names leads to a challenging trust delegation problem. Who is allowed to decide whether a particular host name maps to a specific entity? Possible solutions to this problem have been looked at.

- The public-key infrastructure (PKI) used by the web has chosen to have a number of trust anchors (root certificate authorities) each of which can map any host name to a public key.

- A number of GSS-API mechanisms, such as Kerberos [RFC1964], have split the problem into two parts. A new concept called a realm is introduced, the realm is responsible for host mapping within that realm. The mechanism then decides what realm is responsible for a given name. This is the approach adopted by ABFAB.

GSS-EAP defines a host naming convention that takes into account the host name, the realm, the service and the service parameters. An example of GSS-API service name is "xmpp/foo@example.com". This identifies the XMPP service on the host foo in the realm example.com. Any of the components, except for the service name may be omitted from a name. When omitted, then a local default would be used for that component of the name.

While there is no requirement that realm names map to Fully Qualified Domain Names (FQDN) within DNS, in practice this is normally true. Doing so allows for the realm portion of service names and the portion of NAIs to be the same. It also allows for the use of DNS in locating the host of a service while establishing the transport channel between the client and the relying party.

It is the responsibility of the application to determine the server that it is going to communicate with; GSS-API has the ability to help confirm that the server is the desired server but not to determine the name of the server to use. It is also the responsibility of the application to determine how much of the information identifying the service needs to be validated by the ABFAB system. The information that needs to be validated is used to build up the service name passed into the GSS-EAP mechanism. What information is to be validated will depend on both what information was provided by the client, and what information is considered significant. If the client only cares about getting a specific service, then the host and realm that provides the service does not need to be validated.

Applications may retrieve information about providers of services from DNS. Service Records (SRV) [RFC2782] and Naming Authority Pointer (NAPTR) [RFC3401] records are used to help find a host that provides a service; however the necessity of having DNSSEC on the queries depends on how the information is going to be used. If the host name returned is not going to be validated by EAP channel binding, because only the service is being validated, then DNSSEC [RFC4033] is not required. However, if the host name is going to be validated by EAP channel binding then DNSSEC needs to be use to ensure that the correct host name is validated. In general, if the information that is returned from the DNS query is to be validated, then it needs to be obtained in a secure manner.

Another issue that needs to be addressed for host-based service names is that they do not work ideally when different instances of a service are running on different ports. If the services are equivalent, then it does not matter. However if there are substantial differences in the quality of the service that information needs to be part of the validation process. If one has just a host name and not a port in the information being validated, then this is not going to be a successful strategy.

3.4. Additional GSS-API Services

GSS-API provides per-message security services that can provide confidentiality and/or integrity. Some IETF protocols such as NFS and SSH take advantage of these services. As a result GSS-EAP needs to support these services. As with mutual authentication, per-message security services will limit the set of EAP methods that can be used to those that generate a Master Session Key (MSK). Any EAP method that produces an MSK is able to support per-message security services described in [RFC2743].

GSS-API provides a pseudo-random function. This function generates a pseudo-random sequence using the shared session key as the seed for

the bytes generated. This provides an algorithm that both the initiator and acceptor can run in order to arrive at the same key value. The use of this feature allows for an application to generate keys or other shared secrets for use in other places in the protocol. In this regards, it is similar in concept to the TLS extractor (RFC 5705 [RFC5705].). While no current IETF protocols require this, non-IETF protocols are expected to take advantage of this in the near future. Additionally, a number of protocols have found the TLS extractor to be useful in this regards so it is highly probable that IETF protocols may also start using this feature.

4. Privacy Considerations

ABFAB, as an architecture designed to enable federated authentication and allow for the secure transmission of identity information between entities, obviously requires careful consideration around privacy and the potential for privacy violations.

This section examines the privacy related information presented in this document, summarizing the entities that are involved in ABFAB communications and what exposure they have to identity information. In discussing these privacy considerations in this section, we use terminology and ideas from [RFC6973].

Note that the ABFAB architecture uses at its core several existing technologies and protocols; detailed privacy discussion around these is not examined. This section instead focuses on privacy considerations specifically related to overall architecture and usage of ABFAB.

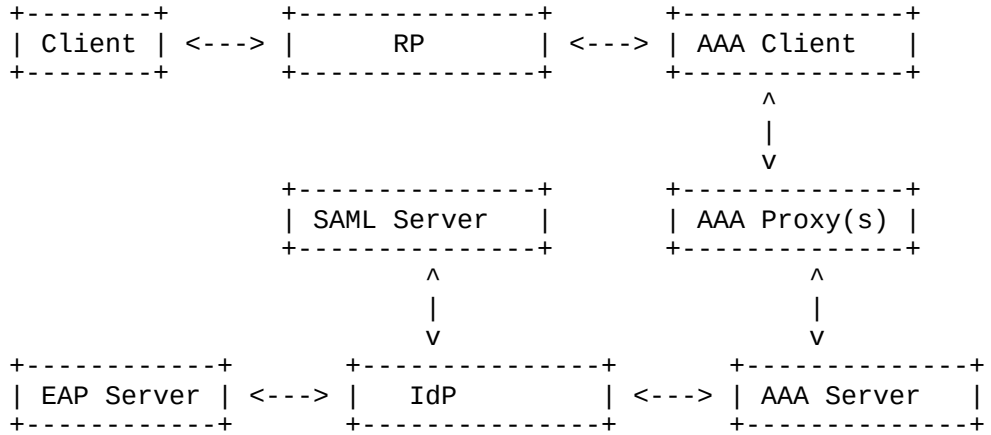


Figure 4: Entities and Data Flow

4.1. Entities and their roles

Categorizing the ABFAB entities shown in the Figure 4 according to the taxonomy of terms from [RFC6973] the entities shown in Figure 4 is somewhat complicated as during the various phases of ABFAB communications the roles of each entity changes. The three main phases of relevance are the Client to RP communication phase, the Client to IdP (via the Federation Substrate) phase, and the IdP to RP (via the Federation Substrate) phase.

In the Client to RP communication phase, we have:

Initiator: Client.

Observers: Client, RP.

Recipient: RP.

In the Client to IdP (via the Federation Substrate) communication phase, we have:

Initiator: Client.

Observers: Client, RP, AAA Client, AAA Proxy(s), AAA Server, IdP.

Recipient: IdP

In the IdP to Relying party (via the Federation Substrate) communication phase, we have:

Initiator: RP.

Observers: IdP, AAA Server, AAA Proxy(s), AAA Client, RP.

Recipient: IdP

Eavesdroppers and Attackers can reside on any or all communication links between entities in Figure 4.

The various entities in the system might also collude or be coerced into colluding. Some of the significant collusions to look at are:

- If two RPs are colluding, they have the information available to both nodes. This can be analyzed as if a single RP was offering multiple services.

- If an RP and a AAA proxy are colluding, then the trust of the system is broken as the RP would be able to lie about its own

identity to the IdP. There is no known way to deal with this situation.

- If multiple AAA proxies are colluding, it can be treated as a single node for analysis.

The Federation Substrate consists of all of the AAA entities. In some cases the AAA Proxies entities may not exist as the AAA Client can talk directly to the AAA Server. Specifications such as the Trust Router Protocol [5] and RADIUS dynamic discovery [I-D.ietf-radext-dynamic-discovery] can be used to shorten the path between the AAA client and the AAA server (and thus stop these AAA Proxies from being Observers); however even in these circumstances there may be AAA Proxies in the path.

In Figure 4 the IdP has been divided into multiple logical pieces, in actual implementations these pieces will frequently be tightly coupled. The links between these pieces provide the greatest opportunity for attackers and eavesdroppers to acquire information, however, as they are all under the control of a single entity they are also the easiest to have tightly secured.

4.2. Privacy Aspects of ABFAB Communication Flows

In the ABFAB architecture, there are a few different types of data and identifiers in use. The best way to understand them, and the potential privacy impacts of them, is to look at each phase of communication in ABFAB.

4.2.1. Client to RP

The flow of data between the client and the RP is divided into two parts. The first part consists of all of the data exchanged as part of the ABFAB authentication process. The second part consists of all of the data exchanged after the authentication process has been finished.

During the initial communications phase, the client sends an NAI (see [I-D.ietf-radext-nai]) to the RP. Many EAP methods (but not all) allow for the client to disclose an NAI to RP the in a form that includes only a realm component during this communications phase. This is the minimum amount of identity information necessary for ABFAB to work - it indicates an IdP that the principal has a relationship with. EAP methods that do not allow this will necessarily also reveal an identifier for the principal in the IdP realm (e.g. a username).

The data shared during the initial communication phase may be protected by a channel protocol such as TLS. This will prevent the leak of information to passive eavesdroppers, however an active attacker may still be able to setup as a man-in-the-middle. The client may not be able to validate the certificates (if any) provided by the service, deferring the check of the identity of the RP until the completion of the ABFAB authentication protocol (i.e., using EAP channel binding).

The data exchanged after the authentication process can have privacy and authentication using the GSS-API services. If the overall application protocol allows for the process of re-authentication, then the same privacy implications as discussed in previous paragraphs apply.

4.2.2. Client to IdP (via Federation Substrate)

This phase sees a secure TLS tunnel initiated between the Client and the IdP via the RP and federation substrate. The process is initiated by the RP using the realm information given to it by the client. Once set up, the tunnel is used to send credentials to IdP to authenticate.

Various operational information is transported between RP and IdP, over the AAA infrastructure, for example using RADIUS headers. As no end-to-end security is provided by AAA, all AAA entities on the path between the RP and IdP have the ability to eavesdrop on this information unless additional security measures are taken (such as the use of TLS for RADIUS [I-D.ietf-radext-dtls]). Some of this information may form identifiers or explicit identity information:

- The Relying Party knows the IP address of the Client. It is possible that the Relying Party could choose to expose this IP address by including it in a RADIUS header such as Calling Station ID. This is a privacy consideration to take into account of the application protocol.

- The EAP MSK is transported between the IdP and the RP over the AAA infrastructure, for example through RADIUS headers. This is a particularly important privacy consideration, as any AAA Proxy that has access to the EAP MSK is able to decrypt and eavesdrop on any traffic encrypted using that EAP MSK (i.e., all communications between the Client and RP). This problem can be mitigted by the application protocol setting up a secure tunnel between the Client and the RP and performing a cryptographic binding between the tunnel and EAP MSK.

- o Related to the above, the AAA server has access to the material necessary to derive the session key, thus the AAA server can observe any traffic encrypted between the Client and RP. This "feature" was chosen as a simplification and to make performance faster; if it was decided that this trade-off was not desirable for privacy and security reasons, then extensions to ABFAB that make use of techniques such as Diffie-Helman key exchange would mitigate against this.

The choice of EAP method used has other potential privacy implications. For example, if the EAP method in use does not support trust anchors to enable mutual authentication, then there are no guarantees that the IdP is who it claims to be, and thus the full NAI including a username and a realm might be sent to any entity masquerading as a particular IdP.

Note that ABFAB has not specified any AAA accounting requirements. Implementations that use the accounting portion of AAA should consider privacy appropriately when designing this aspect.

4.2.3. IdP to RP (via Federation Substrate)

In this phase, the IdP communicates with the RP informing it as to the success or failure of authentication of the user, and optionally, the sending of identity information about the principal.

As in the previous flow (Client to IdP), various operation information is transported between IdP and RP over the AAA infrastructure, and the same privacy considerations apply. However, in this flow, explicit identity information about the authenticated principal can be sent from the IdP to the RP. This information can be sent through RADIUS headers, or using SAML [I-D.ietf-abfab-aaa-saml]. This can include protocol specific identifiers, such as SAML NameIDs, as well as arbitrary attribute information about the principal. What information will be released is controlled by policy on the Identity Provider. As before, when sending this through RADIUS headers, all AAA entities on the path between the RP and IdP have the ability to eavesdrop unless additional security measures are taken (such as the use of TLS for RADIUS [I-D.ietf-radext-dtls]). When sending this using SAML, as specified in [I-D.ietf-abfab-aaa-saml], confidentiality of the information should however be guaranteed as [I-D.ietf-abfab-aaa-saml] requires the use of TLS for RADIUS.

4.3. Relationship between User and Entities

- o Between User and IdP - the IdP is an entity the user will have a direct relationship with, created when the organization that

operates the entity provisioned and exchanged the user's credentials. Privacy and data protection guarantees may form a part of this relationship.

- Between User and RP - the RP is an entity the user may or may not have a direct relationship with, depending on the service in question. Some services may only be offered to those users where such a direct relationship exists (for particularly sensitive services, for example), while some may not require this and would instead be satisfied with basic federation trust guarantees between themselves and the IdP). This may well include the option that the user stays anonymous with respect to the RP (though obviously never to the IdP). If attempting to preserve privacy through the mitigation of data minimization, then the only attribute information about individuals exposed to the RP should be that which is strictly necessary for the operation of the service.

- Between User and Federation substrate - the user is highly likely to have no knowledge of, or relationship with, any entities involved with the federation substrate (not that the IdP and/or RP may, however). Knowledge of attribute information about individuals for these entities is not necessary, and thus such information should be protected in such a way as to prevent access to this information from being possible.

4.4. Accounting Information

Alongside the core authentication and authorization that occurs in AAA communications, accounting information about resource consumption may be delivered as part of the accounting exchange during the lifetime of the granted application session.

4.5. Collection and retention of data and identifiers

In cases where Relying Parties are not required to identify a particular individual when an individual wishes to make use of their service, the ABFAB architecture enables anonymous or pseudonymous access. Thus data and identifiers other than pseudonyms and unlinkable attribute information need not be stored and retained.

However, in cases where Relying Parties require the ability to identify a particular individual (e.g. so they can link this identity information to a particular account in their service, or where identity information is required for audit purposes), the service will need to collect and store such information, and to retain it for as long as they require. Deprovisioning of such accounts and information is out of scope for ABFAB, but obviously for privacy

protection any identifiers collected should be deleted when they are no longer needed.

4.6. User Participation

In the ABFAB architecture, by its very nature users are active participants in the sharing of their identifiers as they initiate the communications exchange every time they wish to access a server. They are, however, not involved in control of the set of information related to them that transmitted from the IdP to RP for authorization purposes; rather, this is under the control of policy on the IdP. Due to the nature of the AAA communication flows, with the current ABFAB architecture there is no place for a process of gaining user consent for the information to be released from IdP to RP.

5. Security Considerations

This document describes the architecture for Application Bridging for Federated Access Beyond Web (ABFAB) and security is therefore the main focus. Many of the items that are security considerations have already been discussed in the Privacy Considerations section. Readers should be sure to read that section as well.

There are many places in this document where TLS is used. While in some places (i.e. client to RP) anonymous connections can be used, it is very important that TLS connections within the AAA infrastructure and between the client and the IdP be fully authenticated and, if using certificates, that revocation be checked as well. When using anonymous connections between the client and the RP, all messages and data exchanged between those two entities will be visible to an active attacker. In situations where the client is not yet on the net, the status_request extension [RFC6066] can be used to obtain revocation checking data inside of the TLS protocol. Clients also need to get the Trust Anchor for the IdP configured correctly in order to prevent attacks, this is a hard problem in general and is going to be even harder for kiosk environments.

Selection of the EAP methods to be permitted by clients and IdPs is important. The use of a tunneling method such as TEAP [I-D.ietf-emu-eap-tunnel-method] allows for other EAP methods to be used while hiding the contents of those EAP exchanges from the RP and the AAA framework. When considering inner EAP methods the considerations outlined in [RFC7029] about binding the inner and outer EAP methods needs to be considered. Finally, one wants to have the ability to support channel binding in those cases where the client needs to validate that it is talking to the correct RP.

In those places where SAML statements are used, RPs will generally be unable to validate signatures on the SAML statement, either because it is stripped off by the IdP or because it is unable to validate the binding between the signer, the key used to sign and the realm represented by the IdP. For these reasons it is required that IdPs do the necessary trust checking on the SAML statements and RPs can trust the AAA infrastructure to keep the SAML statement valid.

When a pseudonym is generated as a unique long term identifier for a client by an IdP, care must be taken in the algorithm that it cannot easily be reverse engineered by the service provider. If it can be reversed then the service provider can consult an oracle to determine if a given unique long term identifier is associated with a different known identifier.

6. IANA Considerations

This document does not require actions by IANA.

7. Acknowledgments

We would like to thank Mayutan Arumaithurai, Klaas Wierenga and Rhys Smith for their feedback. Additionally, we would like to thank Eve Maler, Nicolas Williams, Bob Morgan, Scott Cantor, Jim Fenton, Paul Leach, and Luke Howard for their feedback on the federation terminology question.

Furthermore, we would like to thank Klaas Wierenga for his review of the pre-00 draft version.

8. References

8.1. Normative References

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

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

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

[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.

[RFC7055] Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", RFC 7055, December 2013.

[I-D.ietf-abfab-aaa-saml] Howlett, J. and S. Hartman, "A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML", draft-ietf-abfab-aaa-saml-09 (work in progress), February 2014.

[I-D.ietf-radext-nai] DeKok, A., "The Network Access Identifier", draft-ietf-radext-nai-06 (work in progress), June 2014.

[RFC6677] Hartman, S., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, July 2012.

8.2. Informative References

[RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, July 2013.

[I-D.ietf-radext-radius-fragmentation] Perez-Mendez, A., Lopez, R., Pereniguez-Garcia, F., Lopez-Millan, G., Lopez, D., and A. DeKok, "Support of fragmentation of RADIUS packets", draft-ietf-radext-radius-fragmentation-06 (work in progress), April 2014.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, September 1997.

[RFC3645] Kwan, S., Garg, P., Gilroy, J., Esibov, L., Westhead, J., and R. Hall, "Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)", RFC 3645, October 2003.

[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.

[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.

[RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.

[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.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

[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.

[RFC7029] Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, October 2013.

[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-10 (work in progress), January 2014.

[I-D.ietf-radext-dtls]

DeKok, A., "DTLS as a Transport Layer for RADIUS", draft-ietf-radext-dtls-13 (work in progress), July 2014.

[I-D.ietf-radext-dynamic-discovery]
Winter, S. and M. McCauley, "NAI-based Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS", draft-ietf-radext-dynamic-discovery-11 (work in progress), March 2014.

[WS-TRUST]
Lawrence, K., Kaler, C., Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.4", OASIS Standard ws-trust-200902, February 2009, <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>.

[NIST-SP.800-63]
Burr, W., Dodson, D., and W. Polk, "Electronic Authentication Guideline", NIST Special Publication 800-63, April 2006.

[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.

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

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC3401] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

Authors' Addresses

Josh Howlett
JANET(UK)
Lumen House, Library Avenue, Harwell
Oxford OX11 0SG
UK

Phone: +44 1235 822363
Email: Josh.Howlett@ja.net

Sam Hartman
Painless Security

Email: hartmans-ietf@mit.edu

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

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

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

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com

ABFAB Working Group
Internet-Draft
Updates: 3748 (if approved)
Intended status: Standards Track
Expires: February 20, 2014

S. Winter
RESTENA
J. Salowey
Cisco

August 19, 2013

Update to the EAP Applicability Statement for ABFAB
draft-ietf-abfab-eapapplicability-06

Abstract

This document updates the Extensible Authentication Protocol (EAP) applicability statement from RFC3748 to reflect recent usage of the EAP protocol in the Application Bridging for Federated Access Beyond web (ABFAB) architecture.

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 20, 2014.

Copyright Notice

Table of Contents

1. Introduction

The EAP applicability statement in [RFC3748] defines the scope of the Extensible Authentication Protocol to be "for use in network access authentication, where IP layer connectivity may not be available.", and states that "Use of EAP for other purposes, such as bulk data transport, is NOT RECOMMENDED.".

While some of the recommendation against usage of EAP for bulk data transport is still valid, some of the other provisions in the applicability statement have turned out to be too narrow. Section 2 describes the example where EAP is used to authenticate application layer access. Section 3 provides new text to update the paragraph 1.3. "Applicability" in [RFC3748].

1.1. Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

2. Uses of EAP for Application-Layer Access

Ongoing work in the IETF specifies the use of EAP over GSSAPI for generic application layer access [I-D.ietf-abfab-gss-eap]. In the past, using EAP in this context has met resistance due to the lack of channel bindings [RFC6677]. Without channel bindings, a peer cannot verify if an authenticator is authorized to provide an advertised service.

However as additional services use EAP for authentication, the distinction of which service is being contacted becomes more important. Application services might have different properties. Consider an environment with multiple printers some of which provide a confidential service to output documents to a controlled location. If a peer sent a document to the wrong service then potentially sensitive information might be printed in an uncontrolled location and be disclosed. In addition, it might be more likely that a low-value service is compromised than some high value service. If the high-value service could be impersonated by a low-value service then the security of the overall system would be limited by the security of the lower value service.

This distinction is present in any environment where peers' security depends on which service they reach. However it is particularly acute in a federated environment where multiple organizations are involved. It is very likely that these organizations will have different security policies and practices. It is very likely that the goals of these organizations will not entirely be aligned. In many situations one organization could gain value by being able to impersonate another. In this environment, authenticating the EAP server is insufficient: the peer must also validate that the contacted host is authorized to provide the requested service.

In environments where EAP is used for purposes other than network access authentication:

- All EAP servers and all application access EAP peers MUST support channel bindings. All network access EAP peers SHOULD support channel bindings.

- Channel binding MUST be used for all application authentication. The EAP server MUST either require that the correct EAP lower-layer attribute or another attribute indicating the purpose of the authentication be present in the channel binding data for application authentication.

- Channel binding SHOULD be used for all network access authentication, and when channel binding data is present, the EAP server MUST require that it contain the correct EAP lower-layer attribute to explicitly identify the reason for authentication.

- Any new usage of EAP MUST use channel bindings including the EAP lower-layer attribute to prevent confusion with network access usage.

Operators need to carefully consider the security implications before relaxing these requirements. One potentially serious attack exists

when channel binding is not required and EAP authentication is introduced into an existing service other than network access. A device can be created that impersonates a Network Access Service to peers, but actually proxies the authentication to the new application service that accepts EAP authentications. This may decrease the security of this service even for users who previously used non-EAP means of authentication to the service.

It is REQUIRED for the application layer to prove that both the EAP Peer and EAP Authenticator possess the EAP Master Session Key (MSK). Failing to validate the possession of the EAP MSK can allow an attacker to insert himself into the conversation and impersonate the peer or authenticator. In addition, the application should define channel binding attributes that are sufficient to validate that the application service is being correctly represented to the peer.

2.1. Retransmission

In EAP, the authenticator is responsible for retransmission. By default EAP assumes that the lower layer (the application in this context) is unreliable. The authenticator can send a packet whenever its retransmission timer triggers. In this mode, applications need to be able to receive and process EAP messages at any time during the authentication conversation.

Alternatively, EAP permits a lower layer to set the retransmission timer to infinite. When this happens, the lower layer becomes responsible for reliable delivery of EAP messages. Applications that use a lock-step or client-driven authentication protocol might benefit from this approach.

In addition to retransmission behavior applications need to deal with discarded EAP messages. For example, whenever some EAP methods receive erroneous input, these methods discard the input rather than generating an error response. If the erroneous input was generated by an attacker, legitimate input can sometimes be received after the erroneous input. Applications MUST handle discarded EAP messages, although the specific way in which discarded messages will be handled depends on the characteristics of the application. Options include failing the authentication at the application level, requesting an EAP retransmit and waiting for additional EAP input.

Applications designers that incorporate EAP into their application need to determine how retransmission and message discards are handled.

2.2. Re-Authentication

EAP lower layers MAY provide a mechanism for re-authentication to happen within an existing session [RFC3748]. Re-authentication permits security associations to be updated without establishing a new session. For network access, this can be important because interrupting network access can disrupt connections and media.

Some applications might not need re-authentication support. For example if sessions are relatively short-lived or if sessions can be replaced without significant disruption, re-authentication might not provide value. Protocols like Hypertext Transfer Protocol (HTTP) [RFC2616] and Simple Mail Transport Protocol (SMTP) [RFC5321] are examples of protocols where establishing a new connection to update security associations is likely to be sufficient.

Re-authentication is likely to be valuable if sessions or connections are long-lived or if there is a significant cost to disrupting them.

Another factor may make re-authentication important. Some protocols only permit one party in a protocol (for example the client) to establish a new connection. If another party in the protocol needs the security association refreshed then re-authentication can provide a mechanism to do so.

Application designers need to determine whether re-authentication support is needed and which parties can initiate it.

3. Revised EAP applicability statement

The following text is added to the EAP applicability statement in [RFC3748].

In cases where EAP is used for application authentication, support for EAP Channel Bindings is REQUIRED on the EAP Peer and EAP Server to validate that the host is authorized to provide the services requested. In addition, the application MUST define channel binding attributes that are sufficient to validate that the application service is being correctly represented to the peer. The protocol carrying EAP MUST prove possession of the EAP MSK between the EAP Peer and EAP Authenticator. In the context of EAP for application access the application is providing the EAP Lower Layer. Applications protocols vary so their specific behavior and transport characteristics needs to be considered when determining their retransmission and re-authentication behavior. Circumstances might require that applications need to perform conversion of identities from an application specific character set to UTF-8 or another character set required by a particular EAP method.

4. Security Considerations

In addition to the requirements discussed in the main sections of the document applications should take into account how server authentication is achieved. Some deployments may allow for weak server authentication that is then validated with an additional existing exchange that provides mutual authentication. In order to fully mitigate the risk of NAS impersonation when these mechanisms are used, it is RECOMMENDED that mutual channel bindings be used to bind the authentications together as described in [I-D.ietf-emu-crypto-bind]. When doing channel binding it is REQUIRED that the authenticator is not able to modify the channel binding data passed between the peer to the authenticator as part of the authentication process.

5. IANA Considerations

This document has no actions for IANA.

6. Acknowledgements

Large amounts of helpful text and insightful thoughts were contributed by Sam Hartman, Painless Security. David Black contributed to the text clarifying channel bindings usage.

7. References

7.1. Normative References

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

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

[RFC6677] Hartman, S., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, July 2012.

7.2. Informational References

[I-D.ietf-emu-crypto-bind] Hartman, S., Wasserman, M., and D. Zhang, "EAP Mutual Cryptographic Binding", draft-ietf-emu-crypto-bind-04 (work in progress), July 2013.

[I-D.ietf-abfab-gss-eap]

Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", draft-ietf-abfab-gss-eap-09 (work in progress), August 2012.

[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.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

Authors' Addresses

Stefan Winter
Fondation RESTENA
6, rue Richard Coudenhove-Kalergi
Luxembourg 1359
LUXEMBOURG

Phone: +352 424409 1
Fax: +352 422473
EMail: stefan.winter@restena.lu
URI: http://www.restena.lu.

Joseph Salowey
Cisco Systems
2901 3rd Ave
Seattle, Washington 98121
USA

EMail: jsalowey@cisco.com

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

Table of Contents

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:

- Phase 0: Discovery
- Phase 1: Authentication
- 1a: EAP authentication
- 1b: AAA Key Transport (optional)
- Phase 2: Secure Association Protocol
- 2a: Unicast Secure Association
- 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 recieved, 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 reciever 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 hosntame.

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:

- 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.

- 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 get_mic

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).

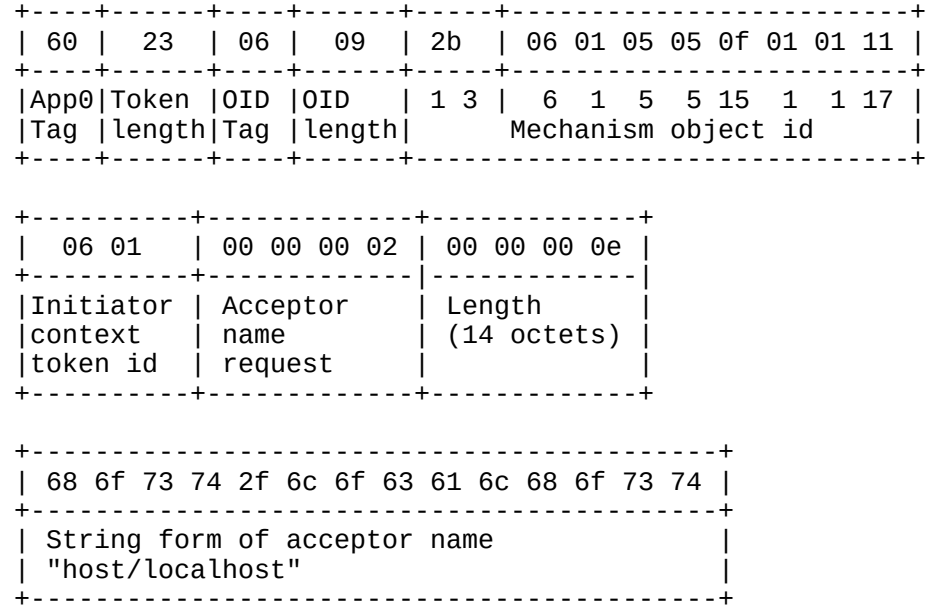
3. For each subtoken other than the MIC subtoken itself in the order the subtokens appear in the token:

   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



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 GMSK (GSS-API MSK). The GMSK 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 GMSK. 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 GMSK and random-to-key is the RFC 3961 random-to-key operation for the enctype of the mechanism. The truncate function takes the initial l 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 GMSK using the following procedure

```
Tn = pseudo-random(GMSK, 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-sha1-96 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 varient 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2013

S. Hartman
Painless Security
J. Howlett
JANET(UK)

November 14, 2012

Name Attributes for the GSS-API EAP mechanism
draft-ietf-abfab-gss-eap-naming-07

Abstract

The naming extensions to the Generic Security Services Application Programming interface provide a mechanism for applications to discover authorization and personalization information associated with GSS-API names. The Extensible Authentication Protocol GSS-API mechanism allows an Authentication/Authorization/Accounting peer to provide authorization attributes along side an authentication response. It also provides mechanisms to process Security Assertion Markup Language (SAML) messages provided in the AAA response. This document describes the necessary information to use the naming extensions API to access that information.

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 18, 2013.

Table of Contents

1. Introduction

The naming extensions [I-D.ietf-kitten-gssapi-naming-exts] to the Generic Security Services Application Programming interface (GSS-API) [RFC2743] provide a mechanism for applications to discover authorization and personalization information associated with GSS-API names. The Extensible Authentication Protocol GSS-API mechanism [I-D.ietf-abfab-gss-eap] allows an Authentication/Authorization/Accounting (AAA) peer to provide authorization attributes along side an authentication response. It also provides mechanisms to process Security Assertion Markup Language (SAML) messages provided in the AAA response. Other mechanisms such as SAML EC [I-D.ietf-kitten-sasl-saml-ec] also support SAML assertions and attributes carried in the GSS-API. This document describes the necessary information to use the naming extensions API to access SAML assertions in the federated context and AAA attributes.

The semantics of setting attributes defined in this specification are undefined and left to future work.

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. Naming Extensions and SAML

SAML assertions can carry attributes describing properties of the subject of the assertion. For example, an assertion might carry an attribute describing the organizational affiliation or e-mail address of a subject. 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 Universal Resource Identifier (URI) describing the format of the name. The second part, whose form depends on the format URI, is the actual name. GSS-API name attributes may take a form starting with a URI describing the form of the name; the rest of the name is specified by that URI.

SAML attributes carried in GSS-API names are named with three parts. The first is a Universal Resource Name (URN) indicating that the name is a SAML attribute and describing the context (Section 4). This URN is followed by a space, the URI indicating the format of the SAML name, a space and the SAML attribute name. The URI indicating the format of the SAML attribute name is not optional and MUST be present.

SAML attribute names may not be globally unique. Many names that are named by URNs or URIs are likely to have semantics independent of the issuer. However other name formats, including unspecified name formats, make it easy for two issuers to choose the same name for attributes with different semantics. Attributes using the federated context Section 4 are issued by the same party performing the authentication. So, based on who is the subject of the name, the semantics of the attribute can be determined.

4. Federated Context

GSS-API naming extensions have the concept of an authenticated name attribute. The mechanism guarantees that the contents of an authenticated name attribute are an authenticated statement from the trusted source of the peer credential. The fact that an attribute is authenticated does not imply that the trusted source of the peer credential is authorized to assert the attribute.

In the federated context, the trusted source of the peer credential is typically some identity provider. In the GSS EAP mechanism, information is combined from AAA and SAML sources. The SAML IDP and home AAA server are assumed to be in the same trust domain. However, this trust domain is not typically the same as the trust domain of the service. With other SAML mechanisms using this specification, the SAML assertion also comes from the party performing authentication. Typically, the IDP is run by another organization in the same federation. The IDP is trusted to make some statements, particularly related to the context of a federation. For example, an academic federation's participants would typically trust an IDP's assertions about whether someone was a student or a professor. However that same IDP would not typically be trusted to make assertions about local entitlements such as group membership. Thus, a service MUST make a policy decision about whether the IDP is permitted to assert a particular attribute and about whether the asserted value is acceptable. This policy can be implemented as local configuration on the service, as rules in AAA proxies, or through other deployment-specific mechanisms.

In contrast, attributes in an enterprise context are often verified by a central authentication infrastructure that is trusted to assert most or all attributes. For example, in a Kerberos infrastructure, the KDC typically indicates group membership information for clients to a server using KDC-authenticated authorization data.

The context of an attribute is an important property of that attribute; trust context is an important part of this overall context. In order for applications to distinguish the context of attributes, attributes with different context need different names. This specification defines attribute names for SAML and AAA attributes in the federated context.

These names MUST NOT be used for attributes issued by a party other than one closely associated with the source of credentials unless the source of credentials is re-asserting the attributes. For example, a source of credentials can consult whatever sources of attributes it chooses, but acceptors can assume attributes in the federated context are from the source of credentials. This requirement is typically

enforced in mechanism specifications. For example [I-D.ietf-abfab-aaa-saml] provides enough information thatwe know the attributes it carries today are in the federated context. Similarly, we know that the requirements of this paragraph are met by SAML mechanisms where the assertion is the means of authentication.

5. Name Attributes for GSS-EAP

This section describes how RADIUS attributes received in an access-accept message by the GSS-EAP [I-D.ietf-abfab-gss-eap] mechanism are named. The use of attributes defined in this section for other RADIUS messages or prior to the access-accept message is undefined at this time. Future specifations can explore these areas giving adequate weight to backward compatibility. In particular, this specification defines the meaning of these attributes for the src_name output of GSS_Accept_sec_context after that function returns GSS_S_COMPLETE. Attributes MAY be absent or values MAY change in other circumstances; future specifications MAY define this behavior.

The first portion of the name is urn:ietf:params:gss:radius-attribute (a URN indicating that this is a GSS-EAP RADIUS AVP). This is followed by a space and a numeric RADIUS name as described by section 2.6 of [I-D.ietf-radext-radius-extensions]. For example the name of the User-Name attribute is "urn:ietf:params:gss:radius-attribute 1". The name of extended type 1 within type 241 would be "urn:ietf:params:gss:radius-attribute 241.1".

Consider a case where the RADIUS access-accept response includes the RADIUS username attribute. An application wishing to retrieve the value of this attribute would first wait until GSS-_Accept_sec_Context returned GSS_S_COMPLETE. Then the application would take the src_name output from GSS_Accept_sec_context and call GSS_Get_name_attribute passing this name and an attribute of "urn:ietf:params:gss:radius-attribute 1" as inputs. After confirming that the authenticated boolean output is true, the application can find the username in the values output.

The value of RADIUS attributes is the raw octets of the packet. Integers are in network byte order. The display value SHOULD be a human readable string; an implementation can only produce this string if it knows the type of a given RADIUS attribute. If multiple attributes are present with a given name in the RADIUS message, then a multi-valued GSS-API attribute SHOULD be returned. As an exception, implementations SHOULD concatenate RADIUS attributes such as EAP-Message or large attributes defined in [I-D.ietf-radext-radius-extensions] that use multiple attributes to carry more than 253 octets of information.

6. Names of SAML Attributes in the Federated Context

6.1. Assertions

An assertion generated by the credential source is named by "urn:ietf:params:gss:federated-saml-assertion". The value of this attribute is the assertion carried in the AAA protocol or used for authentication in a SAML mechanism. This attribute is absent from a given acceptor name if no such assertion is present or if the assertion fails local policy checks.

When GSS_Get_name_attribute is called, This attribute will be returned with the authenticated output set to true only if the mechanism can successfully authenticate the SAML statement. For the GSS-EAP mechanism this is true if the AAA exchange has successfully authenticated. However, uses of the GSS-API MUST confirm that the attribute is marked authenticated as other mechanisms MAY permit an initiator to provide an unauthenticated SAML statement.

Mechanisms MAY perform additional local policy checks and MAY remove the attribute corresponding to assertions that fail these checks.

6.2. SAML Attributes

Each attribute carried in the assertion SHOULD also be a GSS name attribute. The name of this attribute has three parts, all separated by an ASCII space character. The first part is urn:ietf:params:gss:federated-saml-attribute. The second part is the URI for the <saml:Attribute> element's NameFormat XML attribute. The final part is the <saml:Attribute> element's Name XML attribute. The SAML attribute name may itself contain spaces. As required by the URI specification, spaces within a URI are encoded as "%20". Spaces within a URI, including either the first or second part of the name, encoded as "%20" do not separate parts of the GSS-API attribute name; they are simply part of the URI.

As an example, if the eduPersonEntitlement attribute is present in an assertion, then an attribute with the name "urn:ietf:params:gss:federated-saml-attribute urn:oasis:names:tc:SAML:2.0:attrname-format:uri urn:oid:1.3.6.1.4.1.5923.1.1.1.7" could be returned from GSS_Inquire_Name. If an application calls GSS_Get_name_attribute with this attribute in the attr parameter then the values output would include one or more URIs of entitlements that were associated with the authenticated user.

If the content of each <saml:AttributeValue> element is a simple text node (or nodes), then the raw and "display" values of the GSS name

attribute MUST be the text content of the element(s). The raw value MUST be encoded as UTF-8.

If the value is not simple or is empty, then the raw value(s) of the GSS name attribute MUST be a namespace well-formed serialization [XMLNS]of the <saml:AttributeValue> element(s) encoded as UTF-8. The "display" values are implementation-defined.

These attributes SHOULD be marked authenticated if they are contained in SAML assertions that have been successfully validated back to the trusted source of the peer credential. In the GSS-EAP mechanism, a SAML assertion carried in an integrity-protected and authenticated AAA protocol SHALL be successfully validated; attributes from that assertion SHALL be returned from GSS_Get_name_attribute with the authenticated output set to true. An implementation MAY apply local policy checks to each attribute in this assertion and discard the attribute if it is unacceptable according to these checks.

6.3. SAML Name Identifiers

The <saml:NameID> carried in the subject of the assertion SHOULD also be a GSS name attribute. The name of this attribute has two parts, separated by an ASCII space character. The first part is urn:ietf:params:gss:federated-saml-nameid. The second part is the URI for the <saml:NameID> element’s Format XML attribute.

The raw value of the GSS name attribute MUST be the well-formed serialization of the <saml:NameID> element encoded as UTF-8. The "display" value is implementation-defined. For formats defined by section 8.3 of [OASIS.saml-core-2.0-os], missing values of the NameQualifier or SPNameQualifier XML attributes MUST be populated in accordance with the definition of the format prior to serialization. In other words, the defaulting rules specified for the "persistent" and "transient" formats MUST be applied prior to serialization.

This attribute SHOULD be marked authenticated if the name identifier is contained in a SAML assertion that has been successfully validated back to the trusted source of the peer credential. In the GSS-EAP mechanism, a SAML assertion carried in an integrity-protected and authenticated AAA protocol SHALL be sufficiently validated. An implementation MAY apply local policy checks to this assertion and discard it if it is unacceptable according to these checks.

7. Security Considerations

This document describes how to access RADIUS attributes, SAML attributes and SAML assertions from some GSS-API mechanisms. These attributes are typically used for one of two purposes. The least sensitive is personalization: a central service MAY provide information about an authenticated user so they need not enter it with each acceptor they access. A more sensitive use is authorization.

The mechanism is responsible for authentication and integrity protection of the attributes. However, the acceptor application is responsible for making a decision about whether the credential source is trusted to assert the attribute and validating the asserted value.

Mechanisms are permitted to perform local policy checks on SAML assertions, attributes and name identifiers exposed through name attributes defined in this document. If there is another way to get access to the SAML assertion, for example the mechanism described in [I-D.ietf-abfab-aaa-saml], then an application MAY get different results depending on how the SAML is accessed. This is intended behavior; applications who choose to bypass local policy checks SHOULD perform their own evaluation before relying on information.

8. IANA Considerations

A new top-level registry is created titled "Generic Security Service Application Program Interface Parameters".

In this top-level registry, a sub-registry titled "GSS-API URN Parameters" is created. Registration in this registry is by the IETF review or expert review procedures [RFC5226].

This paragraph gives guidance to designated experts. Registrations in this registry are generally only expected as part of protocols published as RFCs on the IETF stream; other URIs are expected to be better choices for non-IETf work. Expert review is permitted mainly to permit early registration related to specifications under development when the community believes they have reach sufficient maturity. The expert SHOULD evaluate the maturity and stability of such an IETF-stream specification. Experts SHOULD review anything not from the IETF stream for consistency and consensus with current practice. Today such requests would not typically be approved.

If the "paramname" parameter is registered in this registry then its URN will be "urn:ietf:params:gss:paramname". The initial registrations are as follows:

Parameter	Reference
radius-attribute	Section 5
federated-saml-assertion	Section 6.1
federated-saml-attribute	Section 6.2
federated-saml-nameid	Section 6.3

8.1. Registration of the GSS URN Namespace

IANA is requested to register the "gss" URN sub-namespace in the IETF URN sub-namespace for protocol parameters defined in [RFC3553].

Registry Name: gss

Specification: draft-ietf-abfab-gss-eap-naming

Repository: GSS-API URN Parameters (Section 8)

Index Value: Sub-parameters MUST be specified in UTF-8 using standard

URI encoding where necessary.

9. Acknowledgements

Scott Cantor contributed significant text and multiple reviews of this document.

The authors would like to thank Stephen Farrell, Luke Howard, and Jim Schaad

Sam hartman’s work on this specification has been funded by Janet.

10. References

10.1. Normative References

[I-D.ietf-abfab-gss-eap]
Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", draft-ietf-abfab-gss-eap-09 (work in progress), August 2012.

[I-D.ietf-kitten-gssapi-naming-exts]
Williams, N., Johansson, L., Hartman, S., and S. Josefsson, "GSS-API Naming Extensions", draft-ietf-kitten-gssapi-naming-exts-15 (work in progress), May 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.

[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.

[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.

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, June 2003.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[XMLNS] W3C, "XML Namespaces Conformance", 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208/#Conformance>.

10.2. Informative References

[I-D.ietf-abfab-aaa-saml]
Howlett, J. and S. Hartman, "A RADIUS Attribute, Binding and Profiles for SAML", draft-ietf-abfab-aaa-saml-04 (work in progress), October 2012.

[I-D.ietf-kitten-sasl-saml-ec]
Cantor, S. and S. Josefsson, "SAML Enhanced Client SASL and GSS-API Mechanisms", draft-ietf-kitten-sasl-saml-ec-04 (work in progress), October 2012.

Authors' Addresses

Sam Hartman
Painless Security

Email: hartmans-ietf@mit.edu

Josh Howlett
JANET(UK)

Email: josh.howlett@ja.net

ABFAB
Internet-Draft
Intended status: Informational
Expires: March 29, 2013

R. Smith, Ed.
Cardiff University

September 25, 2012

Application Bridging for Federated Access Beyond web (ABFAB) Use Cases draft-ietf-abfab-usecases-05

Abstract

Federated identity is typically associated with Web-based services at present, but there is growing interest in its application in non Web-based contexts. The goal of this document is to document a selection of the wide variety of these contexts whose user experience could be improved through the use of technologies based on the ABFAB architecture and specifications.

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 March 29, 2013.

Copyright Notice

Table of Contents

1. Introduction

Federated identity facilitates the controlled sharing of information about people (a.k.a. 'principals'), commonly across organisational boundaries. This avoids redundant registration of principals who operate in and across multiple domains; both reducing the administrative overhead for the organizations involved and improving the usability of systems for the principal. Simultaneously, it can also help address privacy-related concerns, along with the regulatory and statutory requirements of some jurisdictions.

The information that is passed between organizations may include authentication state and identity information that can be used for many purposes, including making access management decisions. A number of mechanisms support the transmission of this information for Web-based scenarios in particular (e.g. SAML [OASIS.saml-profiles-2.0-os]), but there is significant interest in the more general application of federated identity to include non-Web use cases. This document enumerates some of these use cases, describing how technologies based on the the ABFAB architecture [I-D.lear-abfab-arch] and specifications could be used.

2. Context of Use Cases

The use cases described in this document are a result of work led by Janet, the operator of the United Kingdom's education and research network, responding to requirements from its community, and augmented by various inputs from the IETF community.

The ABFAB architecture and specifications enables authentication and authorization to occur across organizational boundaries. For many applications, principals need not have pre-instantiated accounts that their federated identity maps to before their first visit to that application; the application can perform this process on the fly. In cases where such accounts are required for particular applications, the pre-provisioning process is out of scope of ABFAB technologies, which assumes any such requirements have already been fulfilled. Standards-based work of note that would assist with this pre-provisioning of accounts includes the standards and specifications produced by the IETF SCIM working group.

3. Use Cases

This section describes some of the variety of potential use cases where technologies based on the ABFAB architecture and specifications could help improve the user experience; each includes a brief description of how current technologies attempt to solve the use cases and how this could improved upon by ABFAB implementations.

3.1. Cloud Services

Cloud computing is emerging as a common way of provisioning infrastructure services in an on-demand manner. These services are typically offered as one of three models:

- General infrastructure services such as computing power, network, storage, and utility ("Infrastructure as a Service", or IaaS);
- Software stacks or platforms such as database servers, web servers, application runtime environments, etc. ("Platform as a Service", or PaaS);
- Common application software such as email, shared storage, business applications such as Customer Relationship Management (CRM) or scientific applications ("Software as a Service", or SaaS).

In many cases the provisioned cloud infrastructures and applications need to be integrated with existing infrastructure of the organisation, and it is of course desirable if this could be achieved in a way that allows business or scientific workflows to act across infrastructure both across the cloud and in the local infrastructure in as seamless a manner as possible.

There are two main areas where federated access fits in cloud computing: using federation to help mediate access to cloud based application services (e.g. cloud provided email or CRM systems); and using federation to help mediate access to the management of cloud based infrastructure services.

3.1.1. Cloud-based Application Services

Many organizations are seeking to deliver services to their users through the use of providers based in the ’cloud’. This is typically motivated by a desire to avoid management and operation of commodity services which, through economies of scale and so-forth, can often be delivered more efficiently by such providers.

Many providers already provide web-based access using conventional federated authentication mechanisms; for example, outsourced email provision where federated access is enabled using ’webmail’ applications where access is mediated through the use of SAML [OASIS.saml-profiles-2.0-os]. This use of federated authentication enables organizations that consume cloud services to more efficiently orchestrate the delivery of these services to their users, and enables Single Sign On to the services for these users.

Frequently, however, users will prefer to use desktop applications that do not use web (i.e. HTTP [RFC2616] based) protocols. For example, a desktop email client may use a variety of non-web protocols including SMTP [RFC5321], IMAP [RFC3501] and POP [RFC1939]. Some cloud providers support access to their services using non-web protocols, however, the authentication mechanisms used by these protocols will typically require that the provider has access to the user's credentials - i.e. non federated. Consequently, the provider will require that users' credentials are regularly synchronised from the user organisation to the provider, with the obvious overhead this imparts on the organisation along with the obvious implications for security and privacy; or else be provisioned directly by the provider to the user.

The latter approach of directly provisioning accounts may be acceptable in the case where an organisation has relationships with only a small number of providers, but may become untenable if an organisation obtains services from many providers. Consequently any organisation with a requirement to use non-web protocols would prefer to make use of the credentials that they have already provisioned their users with, and to utilise federated authentication with non-web protocols to obtain access to cloud-based providers.

ABFAB could help in this context as its specifications would enable federated authentication for a variety of non-web protocols, thus gaining the benefits of federated authentication without any of the drawbacks that are currently experienced.

3.1.2. Cloud-based Infrastructure Services

Typical IaaS or PaaS cloud use cases deal with provisioning on-demand cloud based infrastructure services that may include infrastructure components such as computing and storage resources, network infrastructure, and other utilities. Cloud based virtualised applications should ideally operate in the same way as regular non-virtualised applications whilst allowing management of the virtual computing resources (scaling, migration, reconfiguration) without changing the management applications.

In many cases, moving applications or platforms to the Cloud may require their re-designing/re-factoring to support dynamic deployment and configuration, including their security and authentication and authorisation services. These will typically today be extensively based on manual setup and configuration of such components and features as trusted certificates and trust anchors, authorities and trusted services (both their location and certificates), attribute namespaces, policies, etc.

ABFAB could help in this context as a way of moving from the model of manually configured authentication and authorisation towards a more easily managed system involved federated trust and identity, and will be applicable for a wide range of existing features (e.g. connecting to a newly provisioned Virtual Machine through ABFAB enabled secure shell (SSH) [RFC4251] instead of having to manually manage an administrative login to that machine).

3.2. High Performance Computing

High-performance computing (HPC) is a discipline that uses supercomputers and computer clusters to solve complex computation problems; it most commonly associated with scientific research or computational science.

Access to HPC resources, often mediated through technologies such as secure shell, is typically managed through the use of user digital certificates [RFC5280] or through manually provisioned credentials and accounts. This requires HPC operators to issue certificates or accounts to users using a registration process that often duplicates identity management processes that already exist within most user organizations. The HPC community would like to utilise federated identity to perform both the user registration and authentication functions required to use HPC resources, and so reduce costs by avoiding this duplication of effort.

The HPC community also have following additional requirements:

- Improved Business Continuity: In the event of operational issues at an HPC system at one organisation (for example, a power failure), users and jobs could be transparently moved to other HPC systems without the overhead of having to manage user credentials for multiple organizations;

- Establish HPC-as-a-service: Many organizations who have invested in HPC systems want to make their systems easily available to external customers. Federated authentication facilitates this by enabling these customers to use their existing identity management, user credentialing and support processes;

- Improve the user experience: Authentication to HPC systems is normally performed using user digital certificates, which some users find difficult to use. Federated authentication can provide a better user experience by allowing the use of other types of credentials, without requiring technical modifications to the HPC system to support these.

ABFAB could help in this context as it could enable federated

authentication for the many of the protocols and technologies currently in use by HPC providers, such as secure shell.

3.3. Grid Infrastructure

Grids are large-scale distributed infrastructures, consisting of many loosely coupled, independently managed, and geographically distributed resources managed by organisationally independent providers. Users of grids utilise these resources using grid middleware that allows them to submit and control computing jobs, manipulate datasets, communicate with other users, etc. These users are organised into Virtual Organisations (VOs); each VO represents a group of people working collaboratively on a common project. VOs facilitate both the management of its users and the meditation of agreements between its users and resource providers.

Authentication and authorisation within most grids is performed using a Public Key Infrastructure, requiring each user to have an X.509 public-key certificate [RFC5280]. Authentication is performed through ownership of a particular certificate, while authorisation decisions are made based on the user's identity (derived from their X.509 certificate), membership of a particular VO, or additional information assigned to a user by a VO. While efficient and scalable, this approach has been found wanting in terms of usability - many users find certificates difficult to manage, for various reasons.

One approach to ameliorating this issue, adopted to some extent by some grid communities already, is to abstract away direct access to certificates from users, instead using alternative authentication mechanisms and then converting the credential provided by these into standard grid certificates. Some implementations of this idea use existing federated authentication techniques. However, current implementations of this approach suffer from a number of problems, not the least of which is the inability to use the federated credentials used to authenticate to a credential-conversion portal to also directly authenticate to non-web resources such as secure shell daemons.

The ability to use federated authentication directly through ABFAB, without the use of a credential conversion service, would allow users to authenticate to a grid and its associated services, allowing them to directly launch and control computing jobs, all without having to manage, or even see, an X.509 public-key certificate at any point in the process. Authorisation within the grid would still be performed using VO membership asserted issued by the user's identity provider through the federated transport.

3.4. Databases and Directories

Databases (e.g. MySQL, PostgreSQL, Oracle, etc.) and directory technologies (e.g. OpenLDAP, Microsoft Active Directory, Novell eDirectory, etc.) are very commonly used within many organsiations for a variety of purposes. This can include core administrative functions, such as hosting identity information for its users, as well as business functions (e.g. student records systems at educational organizations).

Access to such database and directory systems is usually provided for internal users only, however, users external to the organizations sometimes require access to these systems directly: for example, external examiners in educational organizations requiring access to student records systems, members of cross-organisational project teams who store information in a particular organisation’s systems, external auditors, etc.

Credentials for users both internal or external to the organisation that allow access these databases and directories are usually provisioned manually within an organisation, either using Identity Management technologies or through more manual processes. For the internal users, this situation is fine - this is one of the mainstays of Identity Management. However, for external users who require access, this represents more of a problem for organisational processes. The organisation either has to add these external users to its internal Identity Management systems, or else provision these credentials directly within the database/directory systems and continue to manage them, including appropriate access controls associated with each credential, for the lifetime of that credential.

Federated authentication to databases or directories, via ABFAB technologies, would improve upon this situation as it would remove the need to provision and de-provision credentials to access these systems. Organisations may still wish to manually manage access control of federated identities; however, even this could be provided through federated means, if the trust relationship between organizations was strong enough for the organisation providing the service to rely upon it for this purpose.

3.5. Media Streaming

Media streaming services (audio or audio/video) are often provided publicly to anonymous users, but authentication is important for a protected subset of streams where rights management and access control must be applied.

Streams can be delivered via protocols such as RTSP [RFC3226] / RTP

[RFC3550] which already include authentication, or can be published in an encrypted form with keys only being distributed to trusted users. Federated authentication is applicable to both of these cases.

Alternative mechanisms to managing access exist; for example, an approach where a unique stream URI is minted for each user. However, this relies on preserving the secrecy of the stream URI, and also requires a communication channel between the web page used for authentication and the streaming service itself. Federated authentication would be a better fit for this kind of access control. Thus, AFAB technologies that allow federated authentication directly within (inherently non-web) media streaming protocols would represent an enhancement to this area.

3.6. Printing

A visitor from one organisation to the premises of another often requires the use of print services. Their home organisation may of course offer printing, but the output could be a long way away so the home service is not useful. The user will typically want to print from within a desktop or mobile application.

Where this service is currently offered it would usually be achieved through the use of ’open’ printers (i.e. printers that allow anonymous print requests), where printer availability is advertised through the use of Bonjour or other similar protocols. If the organisation requires authenticated print requests (usually for accounting purposes), the the visitor would usually have to be given credentials that allow this, often supplemented with pay-as-you-go style payment systems.

Adding federated authentication to IPP [RFC2911] (and other relevant protocols) would enable this kind of remote printing service without the administrative overhead of credentialing these visitors (who, of course, may well one time visitors to the organisation). This would be immediately applicable to higher education, where this use case is increasingly important thanks to the success of federated network authentication systems such as eduroam but could also be used in other contexts such as commercial print kiosks, or in large, heterogeneous organizations.

3.7. Accessing Applications from Devices on a Telecoms Infrastructure

Telecom operators typically have the following properties:

- A large collection of registered users, many of whom may have identities registered to a fairly high level of assurance (often

for payment purposes). However, not all users will have this property - for example, non-contract customers on mobile telecoms infrastructures in countries with low levels of identity registration requirements.

- o An existing network infrastructure capable of authenticating a device (e.g. a cellphone or an ADSL router), and by inference, its owner.

- o A large collection of applications (both web-based and non web-based) that its users wish to access using their device. These applications could be hosted by the telecoms operator directly, or could be any application or system on the internet - for example, network messaging services, VoIP, email, etc.

At present, authentication to these applications will be typically configured manually by the user on the device (or on a different device connected to that device) but inputting their (usually pre-provisioned out-of-band) credentials for that application - one per application.

The use of ABFAB technologies in this case, via a mechanism dubbed "federated cross-layer access" (see [I-D.wei-abfab-fcla]) would enhance the user experience of using these applications through devices greatly. Federated cross-layer access would make use of the initial mutual authentication between device and network to enable subsequent authentication and authorisation to happen in a seamless manner for the user of that device authenticating to applications.

3.8. Enhanced Security Services for S/MIME

There are many situations where organizations want to protect information with robust access control, either for implementation of intellectual property right protections, enforcement of contractual confidentiality agreements or because of legal regulations. The Enhanced Security Services (ESS) for S/MIME defines an access control mechanism which is enforced by the recipient’s client after decryption of the message (see [I-D.freeman-plasma-requirements]). The data model used makes use of Policy decision points (PDP) which make the policy decisions, policy enforcement points (PEP) which make decision requests to the PDP, and policy information points (PIP) which issue attributes about subjects. The decisions themselves are based on the policies and on the subject attributes.

The use of ABFAB technologies in this case would enable both the front or back end attribute exchange required to provide subject attributes. When the PEP contacts the PDP, it would initiate an ABFAB authentication in order to authenticate to the PDP and allow it

to obtain these required subject attributes. Once authenticated, the PDP would return a token to the subject PEP which can be used for subsequent authentications to the PDP.

3.9. Smart Objects

Many smart device deployments involve multiple organizations that do not directly share security infrastructure. For example, in smart power deployments, devices including appliances and infrastructure such as electric car chargers will wish to connect to an energy management system. The energy management system is provided by a utility company in some deployments. The utility company may wish to grant access only to authorized devices; for example, a consortium of utility companies and device manufacturers may certify devices to connect to power networks.

In another example, consumer devices may be used to access cloud services. For example, a camera could be bound to a photo processing site. Authentication and authorization for uploading pictures or ordering prints is required. Sensors could be used to provide data to services run by organizations other than the sensor manufacturer. Authorization and authentication can become very tricky when sensors have no user interface. Cellular devices may want to access services provided by a third party regardless of whether the cellular network or wi-fi is used. This becomes difficult when authorization and billing is coordinated by the cellular provider.

The use of ABFAB technologies in this case would provide authentication between one entity, such as a smart device, and its identity provider. Only two parties are involved in this exchange; this means that the smart device need not participate in any complicated public-key infrastructure even if it is authenticating against many cloud services. Instead, the device can delegate the process of authenticating the service and even deciding whether the device should be permitted to access the service to the identity provider. This has several advantages. A wide variety of revenue sharing models are enabled. Because device authentication is only with a single identity provider, phishing of device credentials can be avoided. Authorization and decisions about what personal information to release are made by the identity provider. The device owner can use a rich interface such as a website to configure authorization and privacy policy even if the device has no user interface. This model works well with pre-provisioning of device credentials.

4. Contributors

The following individuals made important contributions to the text of this document: Tim Bannister (Manchester University), Simon Cooper (Janet), Josh Howlett (Janet), and Mark Tysom (Janet).

5. Acknowledgements

These use-cases have been developed and documented using significant input from Jens Jensen (STFC Rutherford Appleton Laboratory), Daniel Kouril (CESNET), Michal Prochazka (CESNET), Ian Stewart (University of Edinburgh), Stephen Booth (Edinburgh Parallel Computing Centre), Eefje van der Harst (SURFnet), Joost van Dijk (SURFnet), Robin Breathe (Oxford Brookes University), Yinxing Wei (ZTE Corporation), Trevor Freeman (Microsoft Corp.), Sam Hartman (Painless Security, LLC), and Yuri Demchenko (University of Amsterdam).

6. Security Considerations

This document contains only use cases and defines no protocol operations for ABFAB. Security considerations for the ABFAB architecture are documented in the ABFAB architecture specification, and security considerations for ABFAB technologies and protocols that are discussed in these use cases are documented in the corresponding protocol specifications.

7. IANA Considerations

This document does not require actions by IANA.

8. References

8.1. Normative References

[I-D.lear-abfab-arch] Howlett, J., Hartman, S., Tschofenig, H., and E. Lear, "Application Bridging for Federated Access Beyond Web (ABFAB) Architecture", draft-lear-abfab-arch-02 (work in progress), March 2011.

8.2. Informative References

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

[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.

[RFC2911] Hastings, T., Herriot, R., deBry, R., Isaacson, S., and P. Powell, "Internet Printing Protocol/1.1: Model and Semantics", RFC 2911, September 2000.

[RFC3226] Gudmundsson, O., "DNSSEC and IPv6 A6 aware server/resolver message size requirements", RFC 3226, December 2001.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[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.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[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.

[I-D.wei-abfab-fcla] Wei, Y., "Federated Cross-Layer Access", draft-wei-abfab-fcla-02 (work in progress), March 2012.

[I-D.freeman-plasma-requirements] Freeman, T., Schaad, J., and P. Patterson, "Requirements for Message Access Control", draft-freeman-plasma-requirements-03 (work in progress), August 2012.

Author's Address

Dr. Rhys Smith (editor)
Cardiff University
39-41 Park Place
Cardiff CF10 3BB
United Kingdom

Phone: +44 29 2087 0126
EMail: smith@cardiff.ac.uk

ABFAB
Internet-Draft
Intended status: Informational
Expires: July 12, 2013

R. Smith
Cardiff University

January 8, 2013

Application Bridging for Federated Access Beyond web (ABFAB) Usability and User Interface Considerations
draft-smith-abfab-usability-ui-considerations-03

Abstract

The use of ABFAB-based technologies requires that each user's device is configured with the user's identities that they wish to use in ABFAB transactions. This will require something on that device, either built into the operating system or a standalone utility, that will manage the user's identities and identity to service mappings. Anyone designing that "something" will face the same set of challenges. This document aims to document these challenges with the aim of producing well-thought out UIs with some degree of consistency.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 12, 2013.

Copyright Notice

Table of Contents

1. Introduction

The use of ABFAB-based technologies requires that a user's device is configured with their identities that they wish to use in ABFAB transactions. Achieving this will require something on that device, either built into the operating system or a standalone utility, that will manage the user's identities and identity to service mappings. Anyone designing that "something" will face the same set of challenges. This document aims to document these challenges with the aim of producing well-thought out UIs with some degree of consistency.

2. Conventions

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. Terminology

Various items of terminology used in the document are heavily overloaded and thus could mean a variety of different things to different people. In an attempt to minimise this problem, this section gives a brief description of the main items of terminology used in order to aid with a consistent understanding of this document.

- Identity: In this context, an identity is a credential given to a user by a particular organisation with which they have an organisation. A user MAY have multiple identities. The identity will consist of an NAI, alongside other information that supports authentication.

- Identity Selector: The mechanism by which the GSS-API acquires the identity to use with a particular service, and typically would allow the user to configure a set of identities and service to identity mappings.

- NAI: Network Access Identifier - a standard way of identifying a user. See [RFC4282].

- Service: The thing that the user is attempting to authenticate to via ABFAB technology. See [TODO: Link to ABFAB-Use-Cases] for example use cases of what these services could be.

- Trust anchor: An authoritative source of verification of a particular service, used to allow authentication of a server using X.509 [TODO: link]. Typically a commercial CA to allow

authentication via chain of trust, or a preconfigured non-commercial certificate.

4. Context

When using the ABFAB architecture to perform federated authentication to some service, when a user attempts to authenticate to an ABFAB secured application they will need to provide identity information that they wish to authenticate to that particular service with. This will happen through a process of the application calling the GSS-API, which will in turn gather the users credentials through whatever mechanism it has been configured to do so. We will call this mechanism the "identity selector" in this document, though note that this is not a recommendation on terminology for the mechanism!

The simplest way to achieve the desired effect would be a mechanism that simply takes the credentials from the currently logged in user (e.g. the Windows Domain Credentials) and uses those for all services that request authenticate through ABFAB. This approach gives ultimate simplicity in terms of UI - i.e. it wouldn't have one - but the least flexibility. If there is ever to be a requirement for a user to user a different set of credentials for a service, then something more complex will be needed.

Where there is a requirement for multiple credentials to be supported, there are of course two methods that could be employed to configure identities and associated information:

- They could be configured manually by a user in an application specific configuration file that could be edited by hand or some such simple mechanism. While this could work very well functionally, in practice only a small subset of users would be happy with - and able to - configure their identities in such a manner.

- They could be configured through some interactive mechanism. For ease of use this should have a simple UI, although a headless mode may need to be supported for those not using a GUI.

When designing an identity selector with a UI (or indeed, with a headless mode), any implementor will share a common set of usability considerations inherent to the context. This document aims to explore these considerations, and provide advice and guidance on addressing them where possible.

5. Considerations around Terminology

Anyone designing an identity selector will have to grapple with choosing terminology that the average user has some chance of understanding. This terminology can split into a few main functional areas, as discussed next.

5.1. Identity

The first area where terminology is needed is around the identity/ identities of the user. Users are typically used to seeing a variety of terms for aspects of their identity in the federated sense, and an even larger variety in the wider internet sense. For example, in the federated sense some of these terms include "username", "login", "network account", "institutional account", "home organisation account", "credentials", and a myriad of other such terms. However, NAI - the technically correct name for their identity in an ABFAB sense - is highly unlikely to be one of these terms that users are used to seeing.

Implementors of an identity selector will need to carefully consider their indended audience for both their level of technical capability and the existing terminology that they may have been exposed to.

Beyond terminology, careful thought needs to be given to the paradigm to use when presenting identity to users, as identities and services are abstract concepts that some users may not find is easily understandable. Implementors may wish to keep such abstract concepts, or may wish to examine attempts to map to real world paradigms, e.g. the idea of using "Identity Cards" that are held in the user’s "Wallet", as used by Microsoft Cardspace.

5.2. Services

Terminology around services is likely to be less of a problem than identity, but it will actually depend on what the service is. For example, each service could be simply described as "server", "system", etc. But for simplicity just the word "service" will probably suffice.

5.3. Identity to Service Mapping

Depending on your perspective either each identity may be mapped to multiple services, or each service has multiple identities mapped to it. Thus any UI could present either perspective, or both.

6. Considerations around Management of Identities

One of the core features of an identity selector is the management of a user's identities. This section first looks at what information associated with an identity will need to managed, and then looks in detail at various usability considerations of this area.

6.1. Information associated with each Identity

There is firstly a minimal set of information that MUST be stored about each identity to allow ABFAB authentication to take place:

- Issuing organisation: Shows the organisation that issued this particular credential. TODO: This should be... what, a realm? (e.g. "sandford.edu")? What about a friendly name for that realm? For example "Sandford University"?

- NAI: The user's Network Access Identifier (see [RFC4282]) for this particular credentials. For example, "joe@example.com".

- Password: The password associated with this particular NAI.

- Trust anchor: For the identity selector to be able to verify that the server it is going to talk to and attempt to authenticate against is the server that it is expecting, and that it is not being spoofed in some way. This is likely to be an X.509 certificate [TODO X509 ref].

Next up is a small set of information that SHOULD be stored about each identity to allow the user to effectively select a particular identity:

- Friendly name for identity: To allow the user to differentiate between the set of identities represented in the Identity Selector. This should be editable by the user. The only restriction on this name is that it MUST be unique within that particular user's set of identities. For example: "My University", "Google Account", "Work Login", etc.

- Friendly icon for identity: To allow the user to differentiate between the set of identities they have they should be able to set an icon for that particular identity.

Finally, there is a set of optional information that MAY be stored about each identity that represent useful information for the user to have. Note that this list is not intended to be exhaused; any implementor is free to add any more items to their identity selector that make sense in their implementation.

- Password changing URL: The URL the user should visit should they need to change their password for this particular identity. For example, "http://www.example.com/passwordreset".

- Helpdesk URL: The URL the user should visit to get contact details for the helpdesk of the organisation that issued this particular identity for when the user encounters issues and needs help. For example, https://www.exmaple.com/helpdesk.

6.2. Adding/Association of an Identity

Users will have one or more identities given to them by organisations that they have a relationship with. One of the core tasks of an identity selector will be to learn about these identities in order to use them when it comes to authenticating to services on behalf of the user. Adding these identities could be done in one of three ways: manual addition, automated addition that is manually triggered, or automated addition that is automatically triggered. Each of these are discussed in more detail next.

Note that the term "association" or "addition" of an identity is used rather than "provisioning" of an identity, because while we actually are provisioning identities into the UI, provisioning is an overloaded term in the identity and access management space and could easily be confused with identity provisioning in the sense of the creation of the identity by the home organisation’s identity management procedures.

6.2.1. Manual Addition

Allowing users to manually add an identity is technically the easiest method to , but it is a method that has the greatest usability drawbacks. Most of the information required is relatively technical and finding some way of explaining what each field is to an untechnical audience is challenging (to say the least). This especially is the case for trust anchor information. Thus this method should be considered as a power-user option only, or as a fall-back should the other methods not be applicable.

When this method is used, careful consideration should be given to the UI presented to the user. The UI will have to ask for all of the information detailed in Section 6.1.

There are two points at which a user could manually add an identity:

- Asynchronously: the user could be allowed to, at any time, trigger a workflow of manually adding an identity. This represents the most flexible way of adding an identity since a user can perform

this at any time. It does, however, have inherent issues when it comes to verifying the newly added identity - see Section 6.4.

- Just In Time: when connecting to a service which has no mapping to an existing identity, the user could be given an option to add a new one, as well as associating with an existing one. This presents a better user experience when it comes to verifying the newly added identity (see Section 6.4), however, represents a less direct method of adding an identity. Users who have not yet added the appropriate identity to their identity selector may find it difficult to understand that they must try to access a particular service in order to add an identity.

Of course, implementors could support both styles of identity addition to gain the benefits of both and give flexibility to the user.

TODO: Something about choosing an appropriate trust anchor and verifying your IdP...

6.2.2. Manually Triggered Automated Addition

One way to bypass the need for manual addition of a user's identity - and all of the usability issues inherent with that approach - is to provide some sort of manually triggered, but automated, provisioning process.

One approach to accomplishing this, for example, could be for an organisation to have a section on their website where their users could visit, enter the user part of their NAI, and be given piece of provisioning data that contains much or all of the relevant identity information for importing into the identity selector.

It is reasonable to assume that any such provisioning service is likely to be organisation specific, so that the Issuing Organisation and realm part of the NAI will be constant, as would be the trust anchor information. The user part of their NAI will have been input on the web service. The password could be provided as a part of the provisioning file or the identity selector could prompt the user to enter it.

Additionally, the user SHOULD be given the opportunity to:

- Supply or change the default friendly name for that identity - to allow the user to customise the identifier they use for that identity;

- Indicate whether or not the identity selector should always ask before using services with this identity - to customise the way in which the identity selector interacts with the user with this particular identity;

- Reject the addition of the identity completely - to allow the user to back out of the association process in an intuitive way.

In this case, trust anchors could be directly provided through the provisioning mechanism to help establish the trust relationship in a secure manner.

6.2.3. Fully Automated Addition

Many organisations manage the machines of their users using enterprise management tools. Such organisations may wish to be able to automatically add a particular user's identity to the identity selector on their machine/network account so that the user has to do nothing.

This represents the best usability for the user - who wouldn't actually have to do anything. However, it can only work on machines centrally managed by the organisation.

Additionally, having an identity automatically provided, including its password, does have some particular usability issues. Users are used to having to provide their username and password to access services. When attempting to access services, authenticating to them completely transparently to the user could represent a source of confusion. User training within an organisation to explain that automated provisioning of their identity has been enabled is the only way to counter this.

6.3. Modifying Identity Information

This process is conceptually fairly similar to adding an identity, and thus shares many of the usability issues with that process. Some particular things are discussed here.

6.3.1. Manual Modification

An identity selector may allow a user to manually modify some or all of the information associated with each identity. The obvious item that MUST be allowed to be changed by the user is the password associated with the identity.

6.3.2. Automated Modification

To ease usability, organisations may wish to automatically provide updates to identity information. For example, if the user's password changes, it could automatically update the password for the identity in the user's identity selector.

6.4. Verifying an identity

An inherent by-product of the ABFAB architecture is that an identity cannot be verified during the addition process; it can only be verified while it is in use with a real service. This represents a definite usability issue no matter which method of identity addition is used (see Section 6.2):

- If the user has manually added the identity (see Section 6.2) they may have gone through the whole manual process with no errors and so believe the identity has been set up correctly. However, when they attempt to access a service, they may be given an error message, thus causing some amount of confusion.

- If the user has had the identity provisioned into their identity selector, then there is a much greater chance of the identity information being correct. However, if any of the information is not correct, then there is the potential for confusion as the user did not add the information in the first place.

Also, if the identity information is incorrect the user may not know where the error lies, and the error messages provided by the mechanism may not be helpful enough to indicate the error and how to fix it (see Section 8).

6.5. Removing an Identity

This is fairly similar to adding or modifying an identity, and thus shares many of the usability issues with those processes. Some particular things are discussed here.

6.5.1. Manual Removal

Allowing the user to manually delete an identity is probably the best way to achieve the goal. Any UI should allow for this option.

6.5.2. Automated Removal

While automated removal of an identity is a way of achieving the goal without having to interact with the user, the consequence is that things may disappear from the user's identity selector without them

realising.

7. Considerations around Management of Service to Identity Mappings

A service to identity mapping tell the identity selector which identity should be used for a particular service. There is potentially a many-to-many association between identities and services since a user may wish to use one of their identities for many services, or more than one identity for a single service (e.g. if they have multiple roles on that service).

This potentially complex many-to-many association between is not easily comprehended by the user, and allowing the user to both manipulate it and control can be challenging. These obstacles are especially common when errors occur after an association has been made. In this scenario it is important to make it easy for the user to disassociate the Identity from the service.

7.1. Listing Services and Identities

A service listing should be considered in the identity selector which is both searchable and editable by the user.

7.2. Showing the Identity currently in use

It would be beneficial if, when using a service, the identity currently in use could be made visible to the user while he/she is using a specific service. This allows the user to identify which the identity is used with a particular service at a particular time (the user may have more than one identity that they could use with a particular service) - so that they can then disassociate the pairing.

7.3. Associating a Service with an Identity

There needs to be a way for the user to create the service to identity association. however this should only occur once the identity has authenticated with the service without any error.

There are a few ways this association could happen.

7.3.1. User-driven Manual Association

The user could manually associate a particular service with a particular identity. In order to do so, however, the user would need to know all of the technical details of that service before hand, such as its realm and all other required information.

7.3.2. Automated Rules-based Association

It would be benefical from a usability perspective to minimise - or avoid entirely - situations where the user has to pick an identity for a particular service. This could be accomplished by having rules to describe services and their mapping to identities. Such a rule could match, for example, a particular identity for all IMAP servers, or a particular identity for all services in a given service realm. These rules could be configured as a part of the automated identity addition process described in Section 6.2.2 or Section 6.2.3

7.4. Disassociating a Service with an Identity

A user MUST be able to disassociate an identity with a service - that is, to be able to remove the mapping without having to remove the identity.

8. Handling of Errors

All GSS-API calls need to be instantiated from the application. For this reason when an error occurs the user needs to be sent back to the application to re-initiate the GSS-API call. This can get tedious and cause the user to opt out of what they are trying to accomplish. In addition to this the error messages themselves may not be useful enough for the user to decipher what has gone wrong.

It is important to try and avoid error cases all together while using GSS-API as error messages and error handling can really effect usability. Another solution would be to alter the application to handle the errors as it is instantiating the GSS-API communication.

TODO: Lots more to discuss here...

8.1. Identity Association/Verification Errors

TODO: e.g. wrong password, bad trust anchors, etc. TODO.

8.2. Service Errors

TODO: e.g. identity is correct but no authorisation. TODO.

8.3. Other Errors.

TODO: e.g. network errors. TODO.

9. Handling of Successes

It is of course hoped that the identity selector will have to occasionally handle successes as well as errors. This section has some brief discussion about some areas you might want to think about.

9.1. Reporting Authentication Success on First Use of Identity

The first time an identity is used with a service, it may or may not be a good idea (depending on the service) to visually indiciate in some way that the process has been successful, in order that the user understands what is happening and is then prepared for future authentication attempts.

9.2. Reporting Authentication Success

On an on-going basis you may or may not wish to indiciate visually to the user a successful authentication to a service. This relates to Section 7.2.

10. Contributors

The following individuals made important contributions to the text of this document: Sam Hartman (Painless Security LLC), and Maria Turk (Codethink Ltd).

11. Acknowledgements

TODO

12. Security Considerations

TODO

13. IANA Considerations

This document does not require actions by IANA.

14. Normative References

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

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

Appendix A. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Draft -02 to draft -03

1. Bumping version to keep it alive.

Draft -01 to draft -02

1. Completed the major consideration sections, lots of rewording throughout.

Draft -00 to draft -01

1. None, republishing to refresh the document. Other than adding this comment...

Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

Author's Address

Dr. Rhys Smith
Cardiff University
39-41 Park Place
Cardiff CF10 3BB
United Kingdom

Phone: +44 29 2087 0126
EMail: smith@cardiff.ac.uk