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.

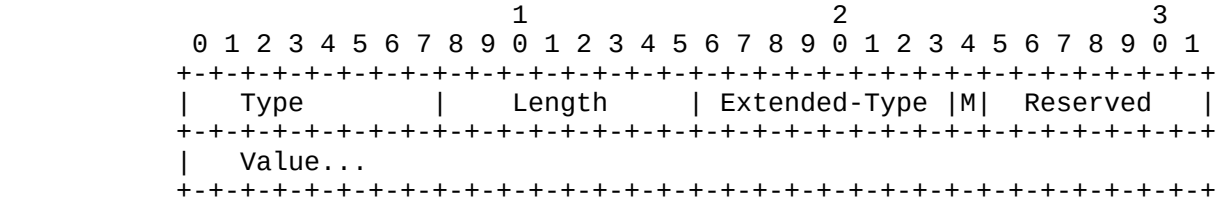


Figure 1: SAML-Assertion format

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.

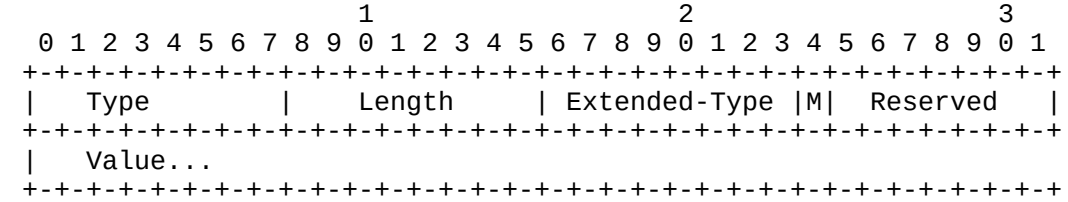


Figure 2: SAML-Protocol format

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:

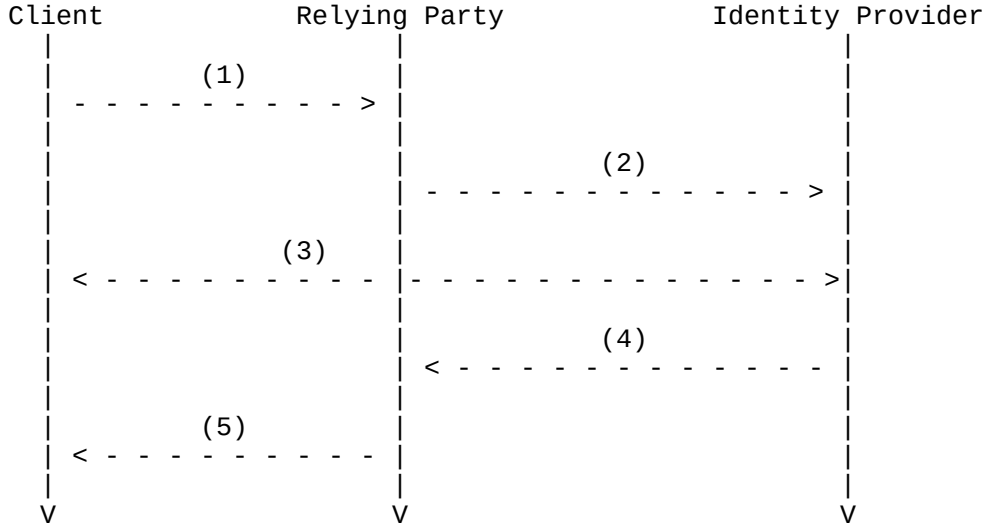
Authentication: This profile does not require the use of any particular authentication method. The ABFAB architecture does require the use of EAP [RFC3579], but this specification may be used in other non-ABFAB scenarios.

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

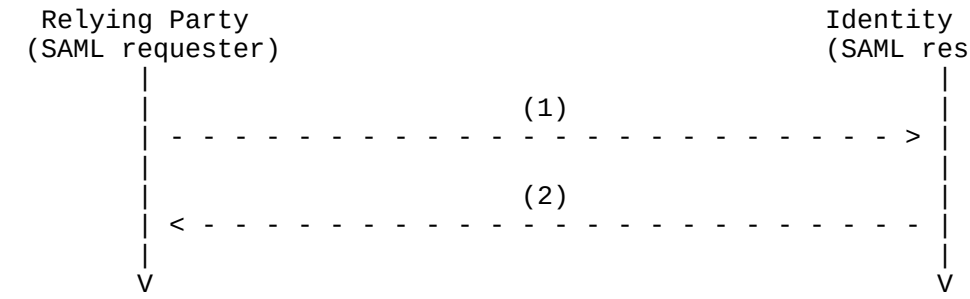
Description: Given below.

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

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).

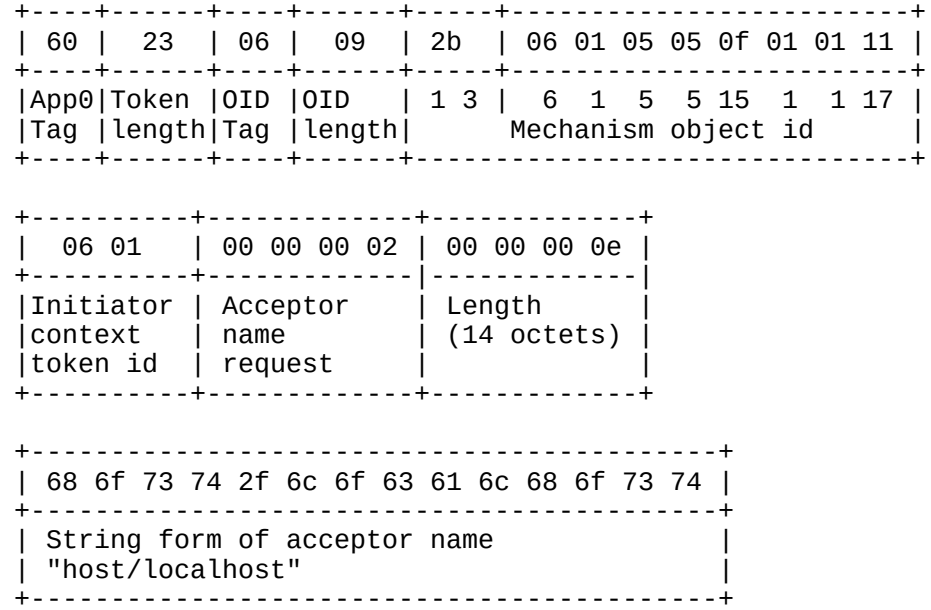
3. For each subtoken other than the MIC subtoken itself in the order the subtokens appear in the token:

   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
Updates: 4556 (if approved)
Intended status: Standards Track
Expires: August 29, 2014

M. Wasserman
S. Hartman
Painless Security
M. Wasserman
JANET (UK)

February 25, 2014

Application Bridging for Federation Beyond the Web (ABFAB) Trust Router Protocol

draft-mrw-abfab-trust-router-02.txt

Abstract

A Trust Router is an infrastucture element used to construct multihop Application Bridging for Federated Authentication Beyond the Web (ABFAB) federations. This document defines both the Trust Router Protocol and the Temporary Identity Protocol, which can be used together to enable multihop ABFAB federations without requiring a centralized Public Key Infrastructure (PKI).

Status of this Memo

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

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

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

This Internet-Draft will expire on August 29, 2014.

Table of Contents

1. Introduction

A Trust Router is an infrastucture element used to construct multihop Application Bridging for Federated Authentication Beyond the Web (ABFAB) federations. This document defines the Temporary Identity Protocol and the Trust Router Protocol, which can be used together to enable multihop ABFAB federations without requiring a centralized Public Key Infrastructure (PKI).

This document defines a Temporary Identity Protocol that can be used by a AAA Client (such as a AAA Proxy near a Relying Party) to negotiate a shared key with the AAA Server(s) in a target IdP realm, so that the AAA Client can use the AAA Server(s) to authenticate users within the realm.

Temporary Identity requests are forwarded by Trust Routers across a chain of Trust Links, eventually reaching the AAA Servers within the target realm. Responses are returned along the same chain. Information about available Trust Links and the paths that can be used to reach AAA Servers within the IdP realms is propogated between Trust Routers using the Trust Router Protocol, which is also defined in this document.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document introduces the following terms:

Trust Router: This is a logical ABFAB entity that exchanges information about Trust Paths that Relying Parties can use to create transtitive chains of trust across multihop ABFAB federations.

Trust Link: A Trust Link is an assertion that a given Trust Router is capable of providing a temporary identity to communicate with another ABFAB entity (either another Trust Router, or a AAA Server within an IdP).

Trust Path: A Trust Path is a concatenation of Trust Links that can be used by an RP to contruct a transitive trust chain across a federation to a target Identity Provider.

Temporary Identity Protocol: The Temporary Identity (TID) Protocol is used to negotiate a shared key between a AAA Client and a AAA Server that can be used for subsequent AAA authentication requests.

Trust Router Protocol: The Trust Router Protocol is the mechanism used by two Trust Routers to exchange information about Trust Links and Trust Paths.

Community of Interest A Community of Interest (COI) defines a group of Services and IdPs that have agreed to cooperate to provide access to a specific set of services only to those users within a particular community. Communities of Interest can be layered on top of the base Trust Router infrastructure to allow selected access to IdPs that have joined a specific group.

Authentication Policy Community An Authentication Policy Community (APC) is a type of community in which the members have agreed to specific policies regarding user authentication.

The terms Identity Provider (IdP), Relying Party (RP), Subject, and Federation are used as defined in [I-D.lear-abfab-arch].

3. Motivation

Figure 1 shows an example federation where the Relying Party Foo, has established relationships with various Identity Providers.

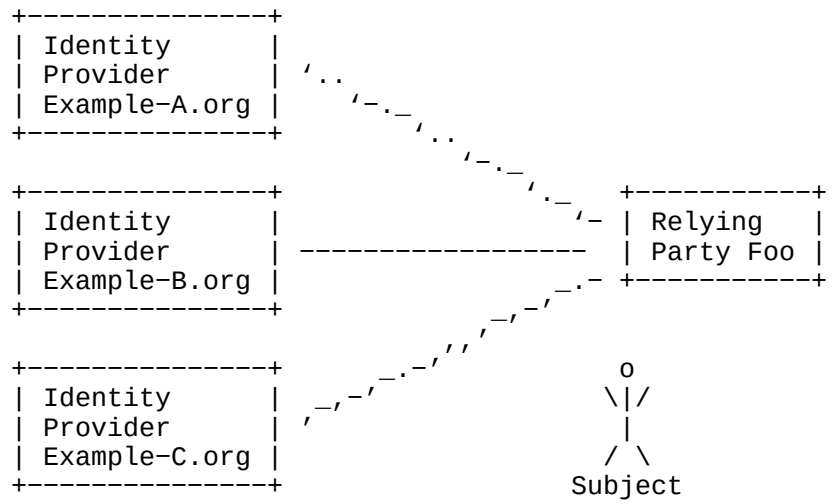


Figure 1: One-to-many Federation Example

When an RP receives a request to access a protected resource (or requires authentication for other purposes) the request includes a realm name that indicates the IdP the Subject has selected for this exchange. Offering the Subject the ability to choose among many different IdPs is necessary because a Subject may have, and want to maintain, uncorrelated identities in several different realms within a single federation (i.e. work, school, social networking, etc.). However, this also places a burden on the RPs to establish and maintain business agreements and exchange security credentials with a potentially large number of Identity Providers.

In order for a single-hop federation to function, each IdP needs to maintain business agreements and exchange credentials with every RP that its Subjects are authorized to access. Figure 2, shows the likely outcome, which is that a single-hop federation will come to resemble a dense mesh topology.

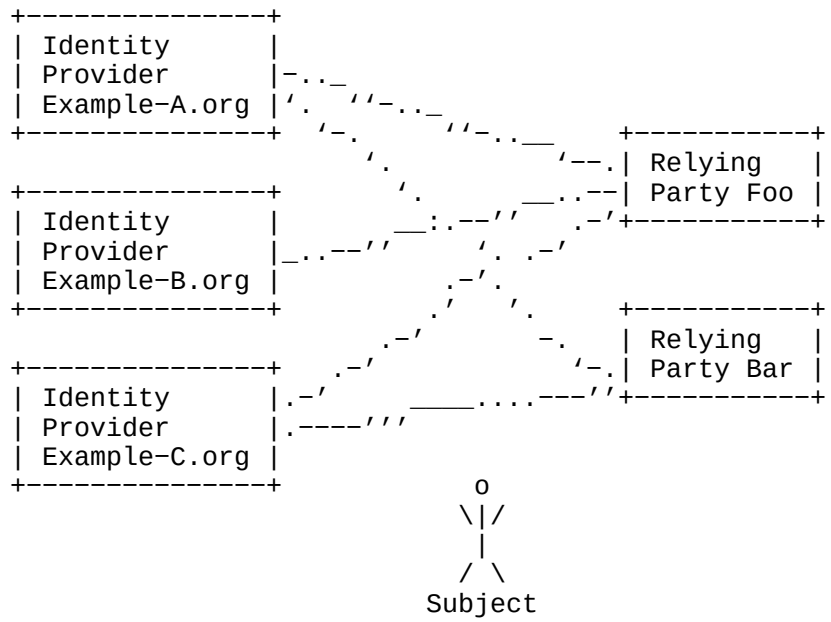


Figure 2: Mesh Federation Example

As discussed in section 2.1.1 of [I-D.lear-abfab-arch], as the number of organizations involved in a ABFAB federation increase, static configuration may not scale sufficiently. Also, using a Trust Broker to establish keys between entities near the RP and entities near the IDP with improve the security and privacy of an ABFAB federation. Figure 3 shows the structure of a federation where each IdP and RP has a single connection to the Trust Router infrastructure.

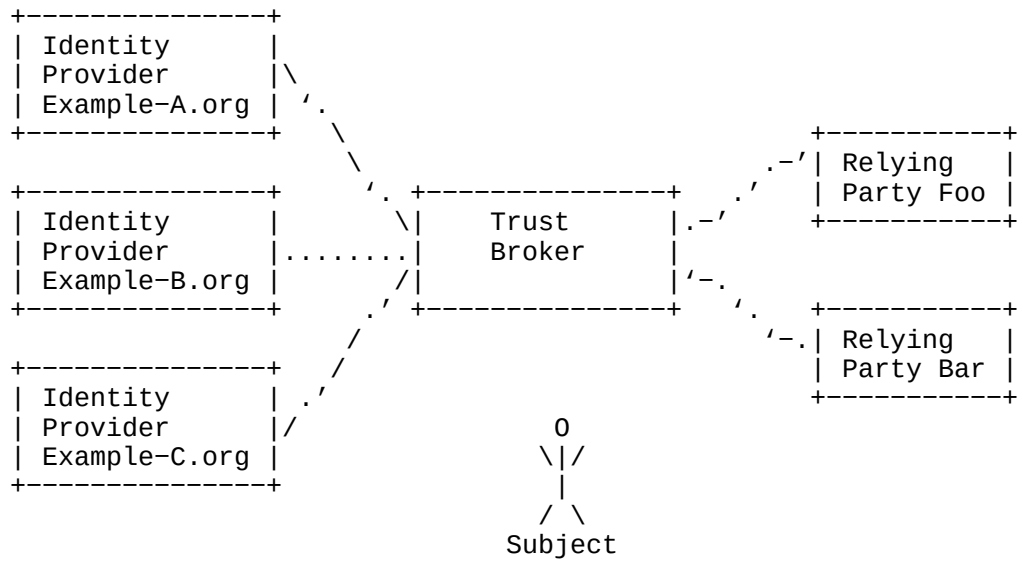


Figure 3: Federation Broker

To improve the operational scalability and security of large ABFAB federations, this document proposes a Trust Broker solution consisting of of a set of Trust Routers, as described in this document, running the Trust Router Protocol and forwarding Temporary Identity Requests between RPs and IdPs.

4. Multihop Federation Example

The diagram below shows an example of a successful exchange in a multihop federation using the Trust Routers to forward Temporary Identity Requests:

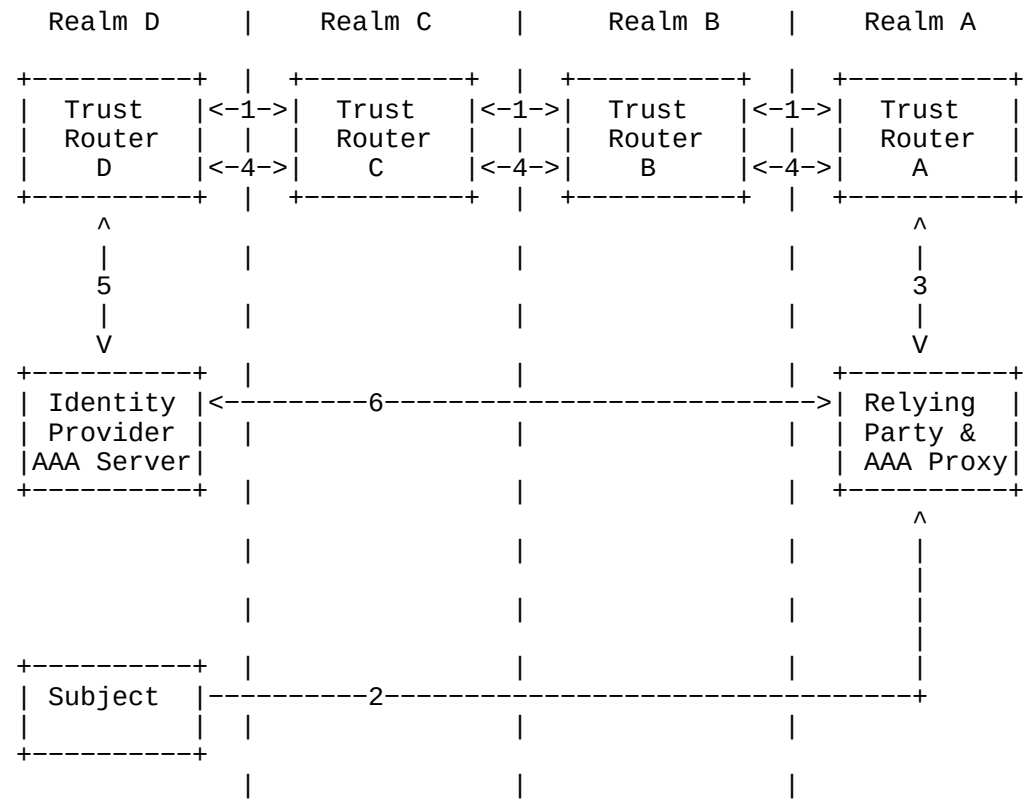


Figure 4: Example Message Exchange

A multihop federation exchange matching the above diagram can be summarized as follows:

1. We start with a single federation including four realms, each containing a single Trust Router. The Trust Routers are peered, such that their interconnections form a multihop federation.

2. A Subject (with an identity in Realm D) attempts to access a service provided by a Relying Party in Realm A.

3. The Relying Party does not have direct access to a AAA Server in Realm D that it can use to authenticate the Subject, so it asks its local Trust Router to forward a Temporary Identity Request to Realm D.

4. Trust Router A forwards the request along the Trust Path from A to B to C to D.

5. The AAA Server in Realm D receives the Temporary Identity request from the local Trust Router and configures a Temporary Identity for the AAA Proxy in Realm A.

6. The AAA Proxy in Realm A can now reach the AAA Server in Realm D to perform the authentication.

5. Temporary Identity Protocol

The Temporary Identity protocol is a simple request/response protocol that is used to established a shared secret between a AAA Client and a AAA Server that can be used for subsequent AAA exchanges. The shared secret is established via a Diffie-Helman (DH) exchange, and it therefore cannot be duplicated by the Trust Routers that forward the Temporary Identity Protocol messages.

5.1. Temporary Identity Request

A Temporary Identify request initially includes the RP Realm for which the identity is being requested, the Target Realm of the request, the Community in which the request is scoped, and a set of client DH parameters used to generate the shared secret.

As a Temporary Identity request is forwarded across the Trust Router chain, it may accumulate additional information, such at APC that corresponds to a COI in the original request and a set of Realm Constraints and Domain Constraints that will be stored by the AAA Server and used for Channel Binding to ensure that the later AAA Request comes from an appropriate AAA Client.

5.2. Temporary Identity Response

A Temporary Identity Response includes most of the fields in the original request. However, the client's DH information has been replaced by a list of AAA Servers for the target realm and a DH block corresponding to each server. The AAA Client can use that information to generate a shared key with each server.

5.3. Role of the Trust Router in Temporary Identity Requests

Trust Routers forward Temporary Identity Requests on behalf of their local AAA Proxies and their neighboring Trust Routers. As part of forwarding these requests, Trust Routers perform a COI to API conversion on the Community field, storing the original COI in an "orig_coi" field. They also add Realm and/or Domain Constraints that can later be used by the AAA Server to ensure that AAA Requests are coming from the correct AAA Client.

6. Trust Router Protocol

The Trust Router protocol is a TCP-based protocol that is used to exchange information between Trust Routers about available Trust Links within an ABFAB Federation.

As discussed in the multihop federation document, When a Trust Router advertises a Trust Link, such as A(T) -> B(T), it is making an assertion that Trust Router A is able, and willing, to provide temporary identities (via KNP) that can be used to reach Trust Router B.

Trust Routers use the information they receive about available Trust Links to construct Trust Paths that can be used to reach AAA Servers (i.e. RADIUS or DIAMETER servers) for a set of Identity Providers (IDPs) within a ABFAB federation. They then return the shortest path to a specific IDP in response to Trust Path Queries.

6.1. Trust Router Messages

6.1.1. Hello Message

Hello Messages are the first messages exchanged by Trust Routers when they bring up a new TCP connection, and they may be exchanged at other times to ensure that database information is synchronized, or to trigger a full Trust Link Database download. The first Hello messages exchanged over a new TCP connection are also used as the vehicle to establish an authenticated and encrypted GSS-API session.

6.1.2. Trust Link Database Message

A Trust Link Database Message contains a full (potentially filtered) set of Trust Links that can be reached through the sending Trust Router. This message may be quite large, and is only sent when solicited by the receiver.

6.1.3. Trust Link Update Message

Trust Routers send Trust Link Update messages to other Trust Routers to whom they are connected whenever their Trust Link Database is updated. Trust Link Update messages contain the portions of the Trust Link Database that have changed since the last update. They also contain a serial number that can be used by the receiving Trust Router to determine if any updates have been missed, in which case a full Trust Router Database download is needed.

6.2. Trust Router Operation

This section describes how Trust Routers work, in general. Detailed message formats are described in later sections of the document.

6.2.1. Hello Message Exchange

6.2.2. Exchanging Trust Link Databases

6.2.3. Trust Link Updates

6.2.4. Serial Numbers

6.2.5. TCP Connection Handling

Trust Routers communicate by exchanging full JSON-encoded messages over a TCP connection. If incomplete messages are received, or if the TCP connection is interrupted before a complete message is received, the incomplete messages will be discarded, and no protocol actions will be taken based on the contents of the incomplete message.

In the Trust Router Protocol, no information about the availability of Trust Links is inferred from a TCP reset, or a retransmission timeout on the TCP connection to another Trust Router. A Trust Router is only considered unreachable after an attempt to reestablish a TCP connection to that Trust Router is reset or times out.

When a Trust Router is found to be unreachable, the Trust Links supplied by that Trust Router are not removed from the local Trust Link Database. They will however, be marked as deprecated until a connection can be reestablished with the Trust Router that sent them, and it can be verified that the sequence number of that Trust Router's Database still matches the sequence number of the most recent Trust Link information received.

When Trust Links are marked as deprecated, they will not be used if another, non-deprecated path exists to reach the target Identity Provider. If there are no paths to the target Identity Provider that traverse only non-deprecated Trust Links, a path containing a deprecated Trust Link will be used.

6.2.6. Conceptual Data Structures

6.2.7. Peer Table

6.2.8. Trust Link Database

7. Message Representation

This section provides details about the contents and encoding of both Trust Router Protocol messages and Trust Path Query messages.

7.1. Message Encoding

The Trust Router Protocol and Trust Path Query messages are encoded in JavaScript Object Notation (JSON) [RFC4627].

7.2. Temporary Identity Protocol Representation

7.2.1. Temporary Identity Request

7.2.2. Temporary Identity Response

7.3. Trust Router Protocol Message Representation

7.3.1. Hello Message Representation

Name or Realm (??) Auth-Token (??) Database-Serial-Number Database-Request

Database-Serial-Number field contains the current serial number of the sending Trust Router's Trust Link Database. This information may be used by a receiving Trust Router to determine whether it should request a full Trust Link Database download.

The Database-Request field indicates whether the receiving Trust Router should respond to this message with a Trust Link Database message, to share its full Trust Link Database with the sending Trust Router. If this field has a value of "true", a download is requested. If it is "false", a download is not requested.

7.3.2. Trust Link Database/Update Representation

In the Trust Router Protocol, each Trust Router will send a (potentially filtered) set of Trust Links to its neighboring Trust Routers. The representation of these Trust Links is designed for efficient encoding, and to allow easy population of a conceptual Trust Link Table on the receiving Trust Router. Each Trust Router will only distribute a set of Trust Links that form a connected tree rooted at the sending Trust Router.

Conceptually, a Trust Link consists:

- A Trust Router that is willing to provide a temporary identity.
- The Trust Router or AAA Server which the identity can be provided.
- The Communities-of-Interest to whom the link is available.
- A lifetime for this link, in seconds.

However, the actual Trust Links passed in the Trust Router protocol rely on inference and ordering to eliminate the need to include the first Trust Router identity in each distributed link. Instead, we use an Index variable, which indicates each Trust Link's level in a conceptual tree, and we order the Trust Links, so that a Trust Link with an Index of N is subordinate to the closest previous Trust Link with an index of N-1 that applies to the same Community-of-Interest. Each conceptual tree is rooted at the sending Trust Router, which is represented by an an entry with an Index value of 0.

7.3.3. Trust Link Ordering

7.3.4. Entity Identity

When we send Trust Router or AAA Server identities in the Trust Router Protocol, that information will be sent in an Entity Identity structure containing the following fields:

- Name
- Type
- Realm

The Name field will typically contain a fully-qualified domain name (FQDN) that can be used to reach the indicated entity (e.g. "tr-A.example.net").

The Type field indicates that the entity is a Trust Router (Type = "T") or a AAA Server (Type = "R", "D", or "S" for a RADIUS Server, DIAMETER Server or RADSEC Server, respectively).

The Realm field contains the security realm associated with the entity (e.g. "example.net").

7.3.5. Trust Link Entry

As transmitted in the Trust Router Protocol, a Trust Link entry will have the following fields:

- Index
- Target-Entity
- Communities-of-Interest
- Lifetime

The Index field contains a non-zero integer value, indicating the depth of this Trust Link in a conceptual tree of links rooted at the sending Trust Router. The maximum value of this field is 255.

The Target-Entity field contains a the Trust Router or AAA Server for which temporary identities can be generated. This also represents the Trust Router that can generate identities for any directly subordinate nodes in the conceptual tree.

The Communities-of-Interest field contains an array of strings, each containing a Community-of-Interest for which this link is available.

The Lifetime field contains an integer that indicates the lifetime of this Trust Link in seconds. Links are removed from the the conceptual Trust Link Table if their lifetime expires.

7.3.6. Trust Link Database Message

A Trust Link Database will consist two fields:

- Serial-Number
- Trust-Links

The Serial-Number field contains an integer indicating the version of the information contained in this database. The maximum value for this field is (2^32 - 1).

The Trust-Links field contains an array of Trust Link Entries.

7.3.7. Trust Link Update Message

8. Message Examples

8.1. Temporary Identity Request Example

```
{"msg_type": "TIDRequest",
 "msg_body":
    {"rp_realm": "foo.example.com",
     "target_realm": "bar.example.net",
     "community": "trust-router-hackers.baz.example.edu",
     "dh_info":
        {"dh_p": "FFFFFFFF...",
         "dh_g": "02",
         "dh_pub_key": "FBF98ABB..."
        }
    }
}
```

8.2. Temporary Identity Response Example

```
{"msg_type": "TIDResponse",
 "msg_body":
    {"rp_realm": "foo.example.com",
     "target_realm": "bar.example.net",
     "community": "apc.baz.example.edu">
     "orig_coi": "trust-router-hackers.baz.example.edu",
     "aaa_servers":
        [{"server_name":"aaa-server.bar.example.net",
          "dh_info:
             {"dh_p": "FFFFFFFF...",
              "dh_g": "02",
              "dh_pub_key": "FBF98ABB..."
             }
          }]
     "realm_constraints":"*.foo.example.com"
    }
}
```

9. Security Considerations

[TBD]

10. IANA Considerations

IANA has allocated the following TCP port numbers for use by protocols described in this document:

[TBD]

11. Acknowledgements

This document was written using the xml2rfc tool described in RFC 2629 [RFC2629].

The following people provided useful comments or feedback on this document: Daniel Kouril, Linus Nordberg, Jim Schaad, Rhys Smith, Kevin Wasserman.

12. Change Log

12.1. Changes between -01 and -02

- Changed Trust Path Query protocol to Temporary Identity Request Protocol
- Added TID details based on implemented code.
- Restructured document to remove need for separate multihop federations document.

12.2. Changes between -00 and -01

- Minor revisions, added authors.

13. References

13.1. Normative References

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

13.2. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

Authors' Addresses

Margaret Wasserman
Painless Security
14 Summer Street, Suite 202
Malden, MA 02148
USA

Phone: +1 781 405-7464
Email: mrw@painless-security.com
URI: http://www.painless-security.com

Sam Hartman
Painless Security
14 Summer Street, Suite 202
Malden, MA 02148
USA

Email: hartmans@painless-security.com
URI: http://www.painless-security.com

Margaret Wasserman
JANET (UK)

Email: josh.howlett@ja.net

ABFAB Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 17, 2013

S. Winter
RESTENA
J. Salowey
Cisco

July 16, 2012

Update to the EAP Applicability Statement for ABFAB
draft-winter-abfab-eapapplicability-02

Abstract

This document updates the EAP applicability statement from RFC3748 to reflect recent usage of the EAP protocol in application oriented use cases proposed in ABFAB

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 17, 2013.

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 (abfab working group) specifies the use of EAP over GSSAPI for generic application layer access. In the past, using EAP in this context has met resistance due to the lack of channel bindings [I-D.ietf-emu-chbind]. Without channel bindings, a peer does not know what service will be provided by the authenticator. In most network access use cases all access servers that are served by a particular EAP server are providing the same or very similar types of service. The peer does not need to differentiate between different access network services supported by the same EAP server.

However as additional services use EAP for authentication, the distinction of which service is being contacted becomes more important. Consider an environment with multiple printers; if a peer printed a document in the wrong location then potentially sensitive information might be printing in a location where the user associated with the peer would be unable to retrieve it. It is also likely that services might have different security properties. For example, 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.

For these reasons, channel binding MUST be implemented by peers, EAP servers and AAA servers in environments where EAP authentication is used to access application layer services. In addition, channel binding MUST default to being required by peers for non-network authentication. If the EAP server is aware that authentication is for something other than a network service, it too MUST default to requiring channel binding. 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 non-network service. A device can be created that impersonates a Network Access Service to peers, but actually proxies the authentication to the service that newly accepts EAP authentications may decrease the security of this service even for users who previously used non-EAP means of authentication to the service.

It is important for the application layer to prove possession of the EAP MSK between the EAP Peer and EAP Authenticator. In addition, the application should define an channel binding attributes that are sufficient to validate that the application service is being correctly represented to the peer.

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. It is important for the protocol carrying EAP to prove possession of the EAP MSK between the EAP Peer and EAP Authenticator.

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 enforced to bind the authentications together as described in [I-D.hartman-emu-mutual-crypto-bind]

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.

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.

[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.

7.2. Informational References

[I-D.hartman-emu-mutual-crypto-bind] Hartman, S., Wasserman, M., and D. Zhang, "EAP Mutual Cryptographic Binding", draft-hartman-emu-mutual-crypto-bind-

00 (work in progress),
March 2012.

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 98121
USA

EMail: jsalowey@cisco.com