

ABFAB
Internet-Draft
Intended status: Informational
Expires: August 18, 2014

J. Howlett
Janet
S. Hartman
Painless Security
February 14, 2014

A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and
Confirmation Methods for SAML
draft-ietf-abfab-aaa-saml-09

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, user or machine principals. These principals may be named using an 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. These artifacts have been defined to permit application in AAA scenarios other than ABFAB, such as network access.

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 18, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. TODO	4
2. Introduction	4
3. Conventions	5
4. RADIUS SAML Attributes	5
5. SAML RADIUS Binding	6
5.1. Required Information	6
5.2. Operation	6
5.3. Processing of names	7
5.3.1. AAA names	8
5.3.2. SAML names	8
5.3.3. Use of XML Signatures	8
5.3.4. Metadata Considerations	9
6. Network Access Identifier Name Identifier Format	9
7. ABFAB Authentication Profile	9
7.1. Required Information	9
7.2. Profile Overview	10
7.3. Profile Description	12
7.3.1. User Agent Request to Relying Party	12
7.3.2. Relying Party Issues <samlp:AuthnRequest> to Identity Provider	12
7.3.3. Identity Provider Identifies Principal	12
7.3.4. Identity Provider Issues <samlp:Response> to Relying Party	13
7.3.5. Relying Party Grants or Denies Access to Principal	13
7.4. Use of Authentication Request Protocol	13
7.4.1. <samlp:AuthnRequest> Usage	13
7.4.2. <samlp:Response message> Usage	14
7.4.3. <samlp:Response Message> Processing Rules	14
7.4.4. Unsolicited Responses	15
7.4.5. Use of the SAML RADIUS Binding	15
7.4.6. Use of XML Signatures	15
7.4.7. Metadata Considerations	15
8. ABFAB Assertion Query/Request Profile	15

8.1.	Required Information	16
8.2.	Profile Overview	16
8.3.	Profile Description	17
8.3.1.	Differences from the SAML V2.0 Assertion Query/Request Profile	17
8.3.2.	Use of the SAML RADIUS Binding	17
8.3.3.	Use of XML Signatures	18
8.3.4.	Metadata Considerations	18
9.	RADIUS State Confirmation Methods	18
10.	Privacy considerations	18
11.	Acknowledgements	19
12.	Security Considerations	19
13.	IANA Considerations	20
13.1.	RADIUS Attributes	20
13.2.	ABFAB Parameters	20
13.3.	Registration of the ABFAB URN Namespace	21
14.	References	21
14.1.	Normative References	21
14.2.	Informative References	22

1. TODO

- o Clean up use of terminology (e.g., "principal") to ensure consistency with other ABFAB docs.
- o Complete the Acknowledgements and Security and Privacy Considerations sections.

2. Introduction

Within the ABFAB 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 [RFC3588], in addition to HTTP.

A companion specification [I-D.jones-diameter-abfab] specifies equivalent functionality for Diameter.

In summary this document specifies:

- o Two RADIUS attributes to encapsulate SAML assertions and protocol messages respectively.
- o A SAML RADIUS binding that defines how SAML assertions and protocol messages can be transported by RADIUS within a SAML exchange.
- o A profile of the SAML Authentication Request Protocol that uses the SAML RADIUS binding to effect SAML-based authentication and authorization.
- o 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.
- o Two SAML Subject Confirmation Methods for indicating that a user or machine principal is the subject of an assertion.

This document aspires 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 where this document provides a 'Required Information' section for the binding and profiles that enumerate:

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

3. 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].

4. RADIUS SAML Attributes

The RADIUS SAML binding defined by this binding Section 5 uses two attributes to convey SAML assertions and protocol messages respectively [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. The table below defines these attributes. The Length of both of these attributes is ≥ 5 . The More and Reserved fields are handled as described in [RFC6929] and are not depicted in this table for simplicity.

Name	Type	Extended-Type	Value
SAML-Assertion	TBD	TBD	One or more octets encoding a SAML assertion
SAML-Message	TBD	TBD	One or more octets encoding a SAML protocol message

Table 1: RADIUS SAML attribute definitions

5. 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.

5.1. Required Information

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

Contact information: iesg@ietf.org

Updates: None.

5.2. Operation

RADIUS can be used over multiple underlying transports; this binding calls out for the use of Transport Layer Security (TLS) Encryption for RADIUS [RFC6614] as REQUIRED to provide interoperability, confidentiality, improve integrity protection and support the use of longer SAML messages.

Implementations of this profile can take advantage of other mechanisms such as RADIUS packet fragmentation [I-D.perez-radext-radius-fragmentation] to permit transport of longer SAML messages over UDP-based RADIUS transports, such as those described in [RFC2865] and [I-D.ietf-radext-dtls]. Support for fragmentation over UDP is not mandatory.

There are two system models for the use of SAML over RADIUS. The first is a request-response model, using the RADIUS SAML-Message attribute defined in Section 4 to encapsulate the SAML protocol messages.

1. The RADIUS client, acting as a SAML requester, 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 [I-D.ietf-radext-nai] scheme. The SAML requester MUST NOT include more than one SAML request element.
2. The RADIUS server, acting as a SAML responder, 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 SAML responder to send a response in the context of this exchange. The SAML responder MUST NOT include more than one SAML response. A SAML responder that refuses to perform a message exchange with the

SAML requester can silently discard the SAML request (this could subsequently be followed by a RADIUS Access-Reject, as the same conditions that cause the SAML responder 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 a SAML responder to use the RADIUS SAML-Assertion attribute defined in Section 4 to encapsulate an unsolicited, unencrypted SAML assertion. This attribute MAY 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-Message and the SAML-Assertion attribute in the same RADIUS message. If a SAML responder is producing a response to a SAML request, then the first system model is used. A SAML responder 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, SAML responders 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 and successful authentication, the RADIUS server SHOULD include a SAML-specified <samlp:Status> element in the SAML response that is transported within the Access-Accept packet sent by the RADIUS server.

In the case of a SAML processing error and failed authentication, the RADIUS server MAY include a SAML-specified <samlp:Status> element in the SAML response that is transported within the Access-Reject packet sent by the RADIUS server.

5.3. Processing of names

SAML entities using profiles of this binding will typically possess both the SAML and AAA names of their correspondents. Frequently these entities will need to apply policy 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.

5.3.1. AAA names

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

- o SAML responders SHOULD apply policy based on the NAS identity associated with the RADIUS Access-Request.
- o SAML requesters SHOULD apply policy based on the NAI realm associated with the RADIUS Access-Accept.

5.3.2. SAML names

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

SAML issuers MAY apply policy based on the requester's <entityId> after validating that the request comes from the NAS. The following methods are sufficient:

- o NAS identity in trusted digitally signed request.
- o NAS identity in trusted SAML federation metadata.

A digitally signed request alone is not sufficient. A RADIUS entity can observe a SAML message and include it in a RADIUS message without the consent of the issuer of that SAML message. If a SAML consumer were to process the SAML message without confirming that it applied to the RADIUS message, inappropriate policy would be used.

SAML consumers MAY apply policy based on the SAML issuer's <entityId> after validating that the response comes from the RADIUS server. The following methods are sufficient:

- o RADIUS realm in trusted digitally signed request.
- o RADIUS realm in trusted SAML federation metadata.

A digitally signed request alone is not sufficient.

5.3.3. Use of XML Signatures

This bindings 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, but this usage is outside of the scope of this binding.

5.3.4. Metadata Considerations

There are no metadata considerations particular to this binding, because this binding and profiles of this binding are intended to be used without metadata. In this usage, RADIUS infrastructure is used to provide integrity and naming. RADIUS configuration is used to provide policy including which attributes are accepted from a SAML responder and which attributes are sent by a SAML responder.

Implementations MAY support other configurations including the use of metadata.

6. 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 [I-D.ietf-radext-nai].

7. ABFAB Authentication Profile

In the scenario supported by the ABFAB Authentication Profile, a Principal controlling a User Agent requests access to a Relying Party. The User Agent and Relying Party uses RADIUS to authenticate the Principal. The Relying Party, acting as a NAS, attempts to validate the Principal's credentials against a RADIUS server acting the Principal's Identity Provider. If the Identity Provider successfully authenticates the Principal, it produces an authentication assertion which is consumed by the Relying Party. During this process, a name identifier might also be established between the Relying Party and the Identity Provider.

7.1. Required Information

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

Contact information: iesg@ietf.org

SAML Confirmation Method Identifiers: The SAML V2.0 "sender vouches" confirmation method identifier, urn:oasis:names:tc:SAML:2.0:cm:sender-vouches, is used by this profile.

Updates: None.

7.2. Profile Overview

To implement this scenario a profile of the SAML Authentication Request protocol is used in conjunction with the SAML RADIUS binding defined in Section 5.

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:

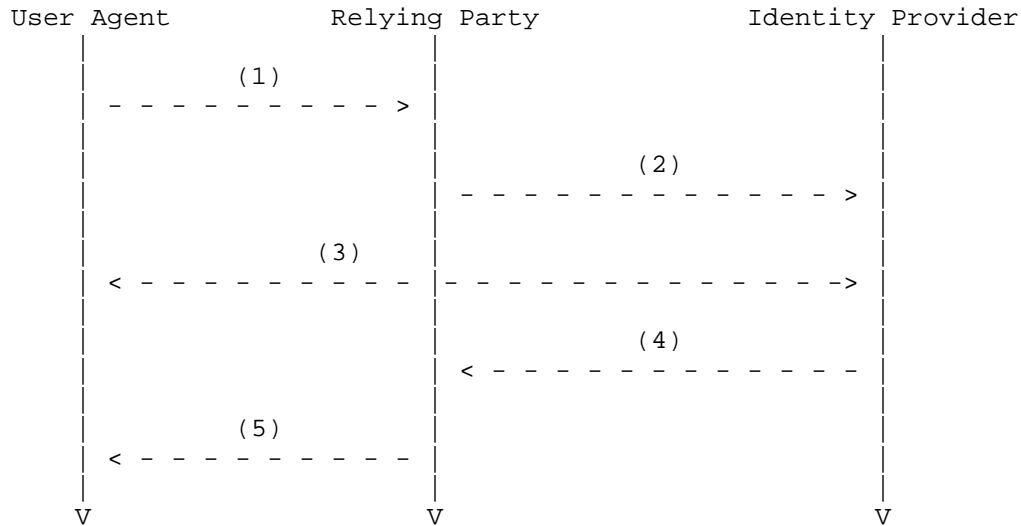
Authentication: This profile does not require the use of any particular authentication method. The ABFAB architecture does require the use of EAP [RFC3579], but this specification may be used in other non-ABFAB scenarios.

Bindings: This profile does not require the use of HTTP-based bindings. Instead all SAML protocol messages are transported using the SAML RADIUS binding defined in Section 5. 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 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 1 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 1

1. User Agent Request to Relying Party (Section 7.3.1): In step 1, the Principal, 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 User Agent exists and initiates authentication of the Principal.
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 RADIUS binding.
3. Identity Provider Identifies Principal (Section 7.3.3). In step 3, the Principal 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 an authentication statement in exactly one assertion.

5. Relying Party Grants or Denies Access to Principal (Section 7.3.5). In step 5, having received the response from the Identity Provider, the Relying Party can respond to the Principal's User Agent with its own error, or can establish its own security context for the Principal 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 this specification conflicts with Core, the former takes precedence.

7.3.1. User Agent Request to Relying Party

The profile is initiated by an arbitrary User Agent 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 NAS, attempts to authenticate the User Agent.

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

The Relying Party uses RADIUS to communicate with the Principal's Identity Provider. The Relying Party MAY include a <samlp:AuthnRequest> within this RADIUS Access-Request message using the SAML RADIUS binding. 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 Principal

The Identity Provider MUST establish the identity of the Principal using RADIUS authentication, or else it will return an error. If the ForceAuthn attribute on the <samlp:AuthnRequest> element (if sent by the requester) 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 Principal (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 Principal, 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, and MAY issue a <samlp:Response> message to the Relying Party consistent with the authentication result and as described in [OASIS.saml-core-2.0-os] and delivered to the Relying Party using the SAML RADIUS binding.

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 Principal

If issued by the Identity Provider, the Relying Party MUST process the <samlp:Response> message and any enclosed <saml:Assertion> elements as described in [OASIS.saml-core-2.0-os]. Any subsequent use of the <saml:Assertion> elements is at the discretion of the Relying Party, subject to any restrictions on use contained within the assertions themselves or previously established out-of-band policy governing interactions 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, the Relying Party is the requester, the User Agent is the attesting entity and the Principal 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 user identifies the principal 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 principal if none exists, it MUST include a <saml:NameIDPolicy> element with the AllowCreate attribute set to "true". Otherwise, only a principal 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 RADIUS SAML binding.

7.4.2. <samlp:Response message> Usage

If the Identity Provider cannot or will not satisfy the request, it MAY respond with a <samlp:Response> message containing an appropriate error status code or codes.

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 MUST conform to the following:

- o It MAY be signed.
- o It MUST contain exactly one <saml:Assertion>. The <saml:Subject> element of this assertion MUST refer to the authenticated RADIUS user.
- o The assertion MUST contain a <saml:AuthnStatement>. This MUST contain a <saml:Subject> element with at least one <saml:SubjectConfirmation> element containing a Method of `urn:oasis:names:tc:SAML:2.0:cm:sender-vouches` that reflects the authentication of the Principal to the Identity Provider. If the containing message is in response to an <samlp:AuthnRequest>, then the `InResponseTo` attribute MUST match the request's ID.
- o 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:

- o Assume that the principal implied by a SAML <Subject> element, if present, takes precedence over a principal implied by the RADIUS User-Name attribute.
- o Verify that the `InResponseTo` attribute in the sender-vouches <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.
- o If a <saml:AuthnStatement> used to establish a security context for the Principal contains a `SessionNotOnOrAfter` attribute, the

security context SHOULD be discarded once this time is reached, unless the service provider reestablishes the Principal's identity by repeating the use of this profile.

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

7.4.4. Unsolicited Responses

An Identity Provider MAY initiate this profile by delivering an unsolicited <saml:Assertion> to a Relying Party. This MUST NOT contain any sender-vouches <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 improve 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, but this usage is outside of the scope of this profile.

7.4.7. Metadata Considerations

There are no metadata considerations particular to this 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] or the SAML RADIUS binding defined elsewhere in this document.

While the SAML V2.0 Assertion Query/Request Profile is independent of the underlying binding, it is nonetheless useful to describe the use of this profile with the SAML RADIUS binding 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

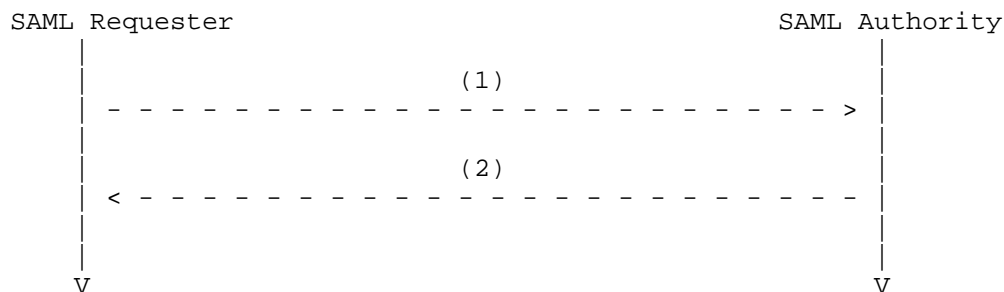
Description: Given below.

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 2 below illustrates the basic template for the query/request profile.



The following steps are described by the profile.

Figure 2

1. Query/Request issued by SAML Requester: In step 1, a SAML requester 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 SAML requester.

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:

- o When processing the SAML request, the SAML responder MUST give precedence to the principal implied by RADIUS State attribute, if present, over the principal implied by the SAML request's <Subject>, if any.
- o In respect to section 6.3.1 and 6.5, this profile does not consider the use of metadata (as in [OASIS.saml-metadata-2.0-os]); see Section 8.3.4.
- o In respect to sections 6.3.2, 6.4.1 and 6.4.2, 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 SAML requester:

- o MUST use a RADIUS User-Name attribute whose value is "@REALM", where REALM is the destination NAI realm.
- o MUST include an instance of the RADIUS Service-Type attribute, having a value of Authorize-Only.
- o SHOULD include the RADIUS State attribute, where this Query/Request pertains to previously authenticated principal.

When processing the SAML request, the SAML responder MUST give precedence to the principal implied by RADIUS State attribute over the principal 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 improve 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, but this usage is outside of the scope of this profile.

8.3.4. Metadata Considerations

There are no metadata considerations particular to this binding.

9. RADIUS State Confirmation Methods

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

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

The RADIUS State Confirmation Methods indicate 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, in the sense of [I-D.ietf-emu-eap-tunnel-method].

10. Privacy considerations

The profiles defined in this document allow a SAML requester to request specific information about the principal and allow a SAML responder to disclose information about a requester. Responders **MUST** apply policy to decide what information is released. The SAML requester does not typically know the identity of the principal unless informed by the SAML responder or RADIUS server. The SAML requester does typically know the realm of the IDP. Information that is released **MAY** include generic attributes such as affiliation shared by many principals. Even these generic attributes can help to identify a specific principal. Other attributes **MAY** provide a SAML requester with the ability to link the same principals between sessions with the same SAML requester. Other attributes **MAY** provide the requester with the ability to link the principal between requesters or with personally identifiable information about the principal.

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

The SAML requester **MAY** disclose privacy-sensitive information about

itself as part of the request. This is unlikely in typical deployments.

If RADIUS proxies are used, then attributes disclosed by the SAML responder 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 NAS 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 observers can observe the assertions.

11. Acknowledgements

TODO: Need to acknowledge OASIS SSTC, UoMurcia, Scott, Jim, and Steven.

12. Security Considerations

TODO: Elaborate on the following

The RADIUS server vouches for its SAML messages. The NAS trusts any statement in the SAML messages from the RADIUS server in the same way that it trusts information contained in RADIUS attributes. The NAS MUST apply policy and filter the information based on what information the RADIUS server is permitted to assert and on what trust is reasonable to place in proxies between the NAS and RADIUS server.

SAML entities' level of trust in the SAML messages that they receive from other entities should be consistent with the trust it holds in the RADIUS infrastructure. That is SAML entities SHOULD trust RADIUS to authenticate the principal and to reach the right IDP. SAML entities trust the RADIUS infrastructure to provide integrity of the SAML messages. However policy MUST be applied to limit what statements are permitted.

XML signatures and encryption are provided as an OPTIONAL mechanism for end-to-end security. These mechanisms 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 NAS 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 ro by parties not involved in the authentication exchange.

13. IANA Considerations

13.1. RADIUS Attributes

Assignments of additional enumerated values for the RADIUS attribute defined in this document are to be processed as described in [RFC6929], subject to the additional requirements of a published specification.

13.2. 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 permit early registration related to specifications under development when the community believes they have reach sufficient maturity. The expert SHOULD evaluate the maturity and stability of such an IETF-stream specification. Experts SHOULD review anything not from the IETF stream for consistency and consensus with current practice. Today such requests would not typically be approved.

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

Parameter	Reference
bindings:radius	Section 5
nameid-format:nai	Section 6
profiles:authentication	Section 7
profiles:query	Section 8
cm:user	Section 9
cm:machine	Section 9

ABFAB Parameters

13.3. 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 13.2)

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

14. References

14.1. Normative References

- | | |
|-----------|---|
| [RFC2119] | Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. |
| [RFC2865] | Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000. |
| [RFC3579] | Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003. |
| [RFC6614] | Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012. |
| [RFC6929] | DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, April 2013. |

- [I-D.ietf-radext-nai] DeKok, A., "The Network Access Identifier", draft-ietf-radext-nai-03 (work in progress), May 2013.
- [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.

14.2. 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, June 2003.

- [RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.
- [RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [I-D.perez-radext-radius-fragmentation] Perez-Mendez, A., Lopez, R., Pereniguez-Garcia, F., Lopez-Millan, G., Lopez, D., and A. DeKok, "Support of fragmentation of RADIUS packets", draft-perez-radext-radius-fragmentation-01 (work in progress), February 2012.
- [I-D.jones-diameter-abfab] Jones, M. and H. Tschofenig, "The Diameter 'Application Bridging for Federated Access Beyond Web (ABFAB)' Application", draft-jones-diameter-abfab-00 (work in progress), March 2011.
- [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-radext-dtls] DeKok, A., "DTLS as a

Transport Layer for RADIUS",
draft-ietf-radext-dtls-05
(work in progress),
April 2013.

[I-D.ietf-emu-eap-tunnel-method]

Zhou, H., Cam-Winget, N.,
Salowey, J., and S. Hanna,
"Tunnel EAP Method (TEAP)
Version 1", draft-ietf-emu-
eap-tunnel-method-06 (work
in progress), March 2013.

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

Phone:
EMail: hartmans-ietf@mit.edu

ABFAB
Internet-Draft
Intended status: Informational
Expires: August 17, 2014

J. Howlett
JANET(UK)
S. Hartman
Painless Security
H. Tschofenig
ARM Ltd.
E. Lear
Cisco Systems GmbH
J. Schaad
Soaring Hawk Consulting
February 13, 2014

Application Bridging for Federated Access Beyond Web (ABFAB)
Architecture
draft-ietf-abfab-arch-12.txt

Abstract

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

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

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 17, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	5
1.1.1. Channel Binding	6
1.2. An Overview of Federation	7
1.3. Challenges for Contemporary Federation	10
1.4. An Overview of ABFAB-based Federation	10
1.5. Design Goals	13
2. Architecture	14
2.1. Relying Party to Identity Provider	15
2.1.1. AAA, RADIUS and Diameter	16
2.1.2. Discovery and Rules Determination	18
2.1.3. Routing and Technical Trust	19
2.1.4. AAA Security	20
2.1.5. SAML Assertions	21
2.2. Client To Identity Provider	23
2.2.1. Extensible Authentication Protocol (EAP)	23
2.2.2. EAP Channel Binding	25
2.3. Client to Relying Party	25
2.3.1. GSS-API	26
2.3.2. Protocol Transport	27
2.3.3. Reauthentication	27
3. Application Security Services	28
3.1. Authentication	28
3.2. GSS-API Channel Binding	29
3.3. Host-Based Service Names	30
3.4. Additional GSS-API Services	32

4.	Privacy Considerations	32
4.1.	Entities and their roles	33
4.2.	Privacy Aspects of ABFAB Communication Flows	35
4.2.1.	Client to RP	35
4.2.2.	Client to IdP (via Federation Substrate)	36
4.2.3.	IdP to RP (via Federation Substrate)	37
4.3.	Relationship between User and Entities	37
4.4.	Accounting Information	38
4.5.	Collection and retention of data and identifiers	38
4.6.	User Participation	38
5.	Security Considerations	39
6.	IANA Considerations	40
7.	Acknowledgments	40
8.	References	40
8.1.	Normative References	40
8.2.	Informative References	41
8.3.	URIs	43
	Authors' Addresses	43

1. Introduction

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

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

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

Federated access management has evolved over the last decade through specifications like SAML [OASIS.saml-core-2.0-os], OpenID [1], OAuth

[RFC6749] and WS-Trust [WS-TRUST]. The benefits of federated access management include:

Single or Simplified sign-on:

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

Data Minimization and User Participation:

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

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

Provisioning:

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

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

1.1. Terminology

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

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

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

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

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

Table 1. Terminology

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

1.1.1. Channel Binding

This document uses the term channel binding with two different meanings.

EAP channel binding is used to implement GSS-API naming semantics. EAP channel binding sends a set of attributes from the peer to the EAP server either as part of the EAP conversation or as part of a secure association protocol. In addition, attributes are sent in the backend protocol from the EAP authenticator to the EAP server. The EAP server confirms the consistency of these attributes and provides the confirmation back to the peer. In this document, channel binding without qualification refers to EAP channel binding.

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

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

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

protocol, mutual authentication is happening in the overall system: the user is able to visually authenticate the content. This is consistent with all uses of channel binding without protocol level mutual authentication found so far.

1.2. An Overview of Federation

In the previous section we introduced the following entities:

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

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

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

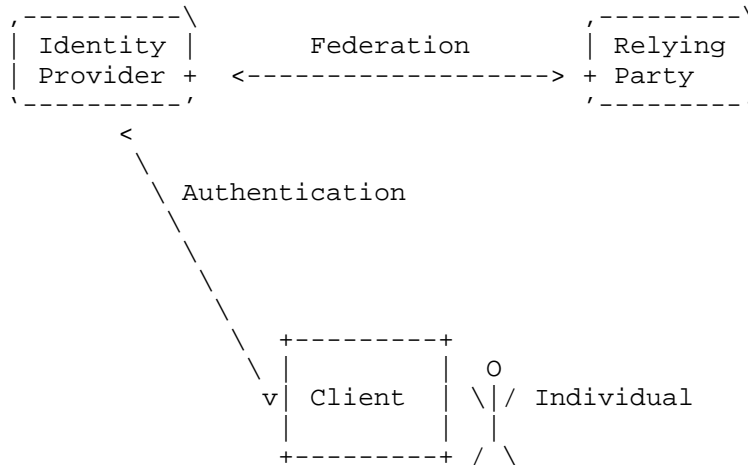


Figure 1: Entities and their Relationships

The relationships between the entities in Figure 1 are:

Federation

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

Authentication

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

A federation agreement typically encompasses operational specifications and legal rules:

Operational Specifications:

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

Legal Rules:

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

The Operational Specifications can demand the usage of a sophisticated technical infrastructure, including requirements on the message routing intermediaries, to offer the required technical functionality. In other environments, the Operational Specifications require fewer technical components in order to meet the required technical functionality.

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

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

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

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

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

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

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

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

1.3. Challenges for Contemporary Federation

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

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

1.4. An Overview of ABFAB-based Federation

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

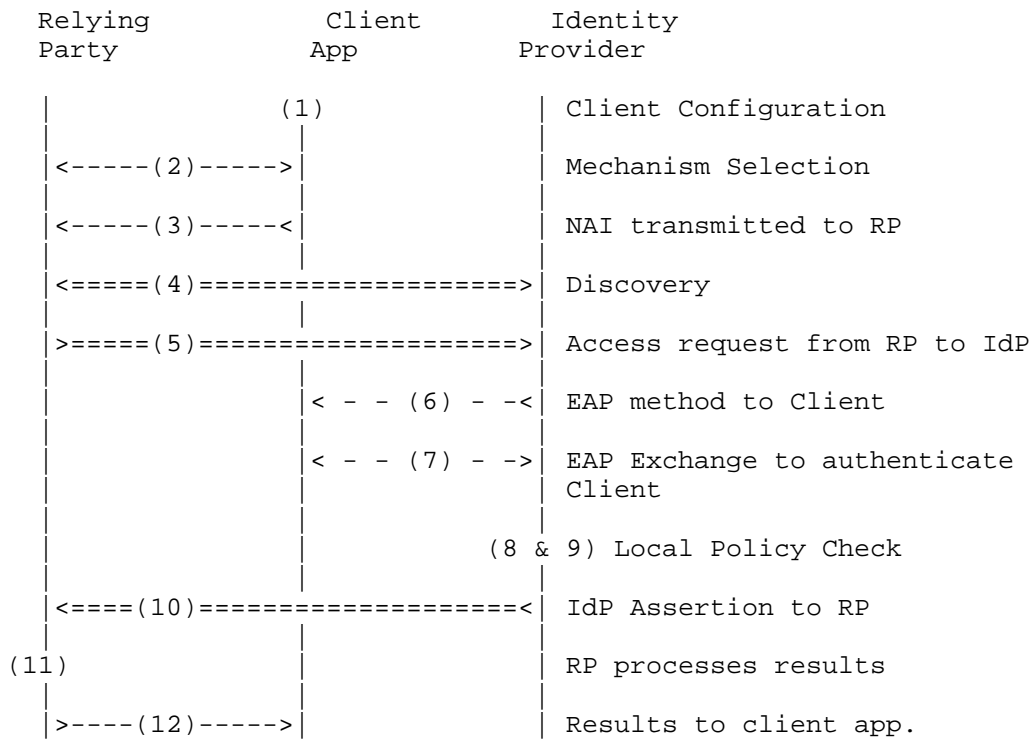
In this example, a client is attempting to connect to a server in order to either get access to some data or perform some type of transaction. In order for the client to mutually authenticate with the server, the following steps are taken in an ABFAB architecture (a graphical view of the steps can be found in figure Figure 2):

1. Client Configuration: The Client Application is configured with an NAI assigned by the IdP. It is also configured with any keys, certificates, passwords or other secret and public

information needed to run the EAP protocols between it and the IdP.

2. Authentication mechanism selection: The Client Application is configured to use the GSS-EAP GSS-API mechanism for authentication/authorization.
3. Client provides an NAI to RP: The client application sets up a transport to the RP and begins the GSS-EAP authentication. In response, the RP sends an EAP request message (nested in the GSS-EAP protocol) asking for the Client's name. The Client sends an EAP response with an NAI form that, at a minimum, contains the realm portion of its full NAI.
4. Discovery of federated IdP: The RP uses pre-configured information or a federation proxy to determine what IdP to use based on policy and the realm portion of the provided Client NAI. This is discussed in detail below (Section 2.1.2).
5. Request from Relying Party to IdP: Once the RP knows who the IdP is, it (or its agent) will send a RADIUS request to the IdP. The RADIUS access request encapsulates the EAP response. At this stage, the RP will likely have no idea who the client is. The RP sends its identity to the IdP in AAA attributes, and it may send a SAML Attribute Request in a AAA attribute. The AAA network checks that the identity claimed by the RP is valid.
6. IdP begins EAP with the client: The IdP sends an EAP message to the client with an EAP method to be used. The IdP should not re-request the client's name in this message, but clients need to be able to handle it. In this case the IdP must accept a realm only in order to protect the client's name from the RP. The available and appropriate methods are discussed below in this memo (Section 2.2.1).
7. The EAP protocol is run: A bunch of EAP messages are passed between the client (EAP peer) and the IdP (EAP server), until the result of the authentication protocol is determined. The number and content of those messages depends on the EAP method selected. If the IdP is unable to authenticate the client, the IdP sends an EAP failure message to the RP. As part of the EAP protocol, the client sends a channel bindings EAP message to the IdP (Section 2.2.2). In the channel binding message the client identifies, among other things, the RP to which it is attempting to authenticate. The IdP checks the channel binding data from the client with that provided by the RP via the AAA protocol. If the bindings do not match the IdP sends an EAP failure message to the RP.

8. Successful EAP Authentication: At this point, the IdP (EAP server) and client (EAP peer) have mutually authenticated each other. As a result, the client and the IdP hold two cryptographic keys: a Master Session Key (MSK), and an Extended MSK (EMSK). At this point the client has a level of assurance about the identity of the RP based on the name checking the IdP has done using the RP naming information from the AAA framework and from the client (by the channel binding data).
9. Local IdP Policy Check: At this stage, the IdP checks local policy to determine whether the RP and client are authorized for a given transaction/service, and if so, what if any, attributes will be released to the RP. If the IdP gets a policy failure, it sends an EAP failure message to the RP.[CREF1] (The RP will have done its policy checks during the discovery process.)
10. IdP provides the RP with the MSK: The IdP sends a positive result EAP to the RP, along with an optional set of AAA attributes associated with the client (usually as one or more SAML assertions). In addition, the EAPMSK is returned to the RP.
11. RP Processes Results: When the RP receives the result from the IdP, it should have enough information to either grant or refuse a resource access request. It may have information that associates the client with specific authorization identities. If additional attributes are needed from the IdP the RP may make a new SAML Request to the IdP. It will apply these results in an application-specific way.
12. RP returns results to client: Once the RP has a response it must inform the client application of the result. If all has gone well, all are authenticated, and the application proceeds with appropriate authorization levels. The client can now complete the authentication of the RP by the use of the EAP MSK value.



----- = Between Client App and RP

===== = Between RP and IdP

- - - = Between Client App and IdP (via RP)

Figure 2: ABFAB Authentication Steps

1.5. Design Goals

Our key design goals are as follows:

- o Each party in a transaction will be authenticated, although perhaps not identified, and the client will be authorized for access to a specific resource.
- o Means of authentication is decoupled from the application protocol so as to allow for multiple authentication methods with minimal changes to the application.

- o The architecture requires no sharing of long term private keys between clients and RPs.
- o The system will scale to large numbers of identity providers, relying parties, and users.
- o The system will be designed primarily for non-Web-based authentication.
- o The system will build upon existing standards, components, and operational practices.

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

2. Architecture

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

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

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

The architecture consists of several building blocks, which is shown graphically in Figure 3. In the following sections, we discuss the

data flow between each of the entities, the protocols used for that data flow and some of the trade-offs made in choosing the protocols.

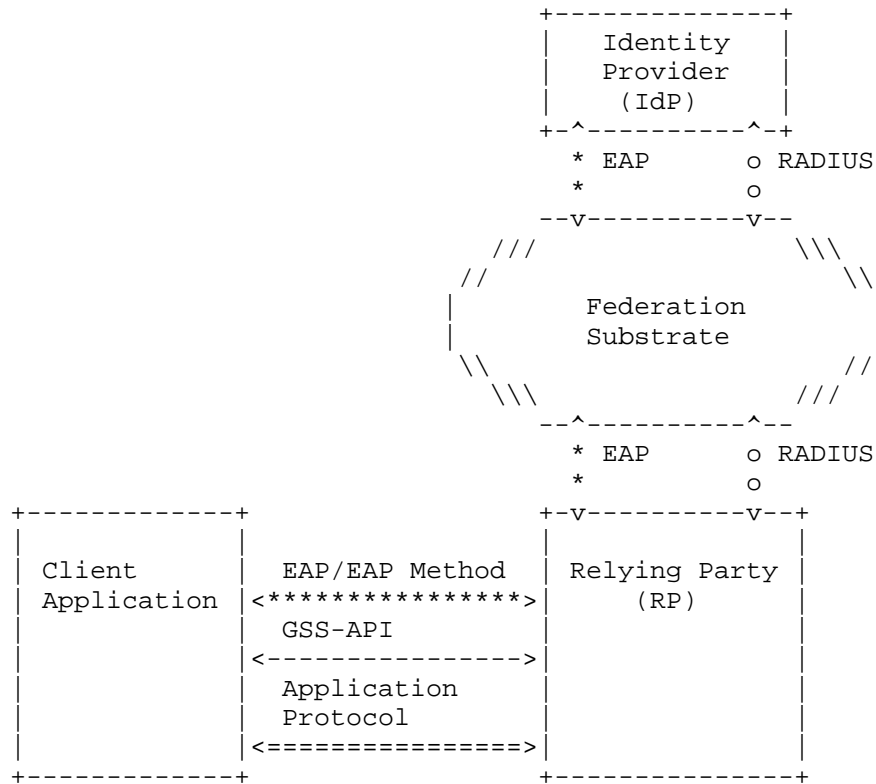


Figure 3: ABFAB Protocol Instantiation

2.1. Relying Party to Identity Provider

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

- o Establishing the trust relationship between the RP and the IdP.

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

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

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

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

2.1.1. AAA, RADIUS and Diameter

The usage of the AAA framework with RADIUS [RFC2865] and Diameter [RFC6733] for network access authentication has been successful from a deployment point of view. To map to the terminology used in Figure 1 to the AAA framework the IdP corresponds to the AAA server, the RP corresponds to the AAA client, and the technical building blocks of a federation are AAA proxies, relays and redirect agents (particularly if they are operated by third parties, such as AAA

brokers and clearing houses). The front-end, i.e. the end host to AAA client communication, is in case of network access authentication offered by link layer protocols that forward authentication protocol exchanges back-and-forth. An example of a large scale RADIUS-based federation is EDUROAM [4].

By using the AAA framework, ABFAB can be built on the federation agreements already exist, the agreements can then merely be expanded to cover the ABFAB. The AAA framework has already addressed some of the problems outlined above. For example,

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

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

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

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

One demand that the AAA substrate makes of the upper layers is that they must properly identify the end points of the communication. It

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

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

2.1.2. Discovery and Rules Determination

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

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

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

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

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

2.1.3. Routing and Technical Trust

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

AAA Proxy:

The simplest model is that an RP is an AAA client and can send the request directly to an AAA proxy. The hop-by-hop integrity protection of the AAA fabric provides technical trust. An RP can submit a request directly to the correct federation. Alternatively, a federation disambiguation fabric can be used. Such a fabric takes information about what federations the RP is part of and what federations the IdP is part of and routes a message to the appropriate federation. The routing of messages across the fabric plus attributes added to requests and responses provides rules determination. For example, when a disambiguation fabric routes a message to a given federation, that federation's rules are chosen. Name validation is enforced as messages travel across the fabric. The entities near the RP confirm its identity and validate names it claims. The fabric routes the message towards the appropriate IdP, validating the name of the IdP in the process. The routing can be statically configured. Alternatively a routing protocol could be developed to exchange reachability information about a given IdP and to apply policy across the AAA fabric. Such a routing protocol could flood naming constraints to the appropriate points in the fabric.

Trust Broker:

Instead of routing messages through AAA proxies, some trust broker could establish keys between entities near the RP and entities near the IdP. The advantage of this approach is efficiency of message handling. Fewer entities are needed to be involved for

each message. Security may be improved by sending individual messages over fewer hops. Rules determination involves decisions made by trust brokers about what keys to grant. Also, associated with each credential is context about rules and about other aspects of technical trust including names that may be claimed. A routing protocol similar to the one for AAA proxies is likely to be useful to trust brokers in flooding rules and naming constraints.

Global Credential:

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

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

2.1.4. AAA Security

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

The default link security for RADIUS is showing its age as it uses MD5 and a shared secret to both obfuscate passwords and to provide integrity on the RADIUS messages. While some EAP methods include the ability to protect the client authentication credentials, the MSK returned from the IdP to the RP is protected only by the RADIUS

security. In many environments this is considered to be insufficient, especially as not all attributes are obfuscated and can thus leak information to a passive eavesdropper. The use of RADIUS with TLS [RFC6614] and/or DTLS [I-D.ietf-radext-dtls] addresses these attacks. The same level of security is included in the base Diameter specifications.

2.1.5. SAML Assertions

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

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

There are several issues that need to be highlighted:

- o The security of SAML assertions.
- o Namespaces and mapping of SAML attributes.
- o Subject naming of entities.
- o Making multiple queries about the subject(s).
- o Level of Assurance for authentication.

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

in the federation is used as an alternate method of validating the origination and integrity of the SAML Assertion.

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

The RADIUS SAML RFC [I-D.ietf-abfab-aaa-saml] has defined a new SAML name format that corresponds to the NAI name form defined by RFC XXXX [I-D.ietf-radext-nai]. This allows for easy name matching in many cases as the name form in the SAML statement and the name form used in RADIUS or Diameter will be the same. In addition to the NAI name form, the document also defines a pair of implicit name forms corresponding to the Client and the Client's machine. These implicit name forms are based on the Identity-Type enumeration defined in TEAP [I-D.ietf-emu-eap-tunnel-method]. If the name form returned in a SAML statement is not based on the NAI, then it is a requirement on the EAP server that it validate that the subject of the SAML assertion, if any, is equivalent to the subject identified by the NAI used in the RADIUS or Diameter session.

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

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

2.2. Client To Identity Provider

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

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

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

2.2.1. Extensible Authentication Protocol (EAP)

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

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

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

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

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

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

- o The first is how to carry EAP payloads from the end host to the relying party.
- o Another is to verify statements the relying party has made to the client, confirm these statements are consistent with statements made to the identity provider and confirm all of the above are consistent with the federation and any federation-specific policy or configuration.
- o Another challenge is choosing which identity provider to use for which service.

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

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

- o It needs to support channel binding.

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

2.2.2. EAP Channel Binding

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

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

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

2.3. Client to Relying Party

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

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

2.3.1. GSS-API

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

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

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

The API view of GSS-API does have significant value as well, since the abstract operations are well defined, the set of information that a mechanism gets from the application is well defined. Also, the set of assumptions the application is permitted to make is generally well defined. As a result, an application protocol that supports GSS-API or SASL is very likely to be usable with a new approach to authentication including this one with no required modifications. In some cases, support for a new authentication mechanism has been added using plugin interfaces to applications without the application being modified at all. Even when modifications are required, they can often be limited to supporting a new naming and authorization model. For example, this work focuses on privacy; an application that

assumes it will always obtain an identifier for the client will need to be modified to support anonymity, unlinkability or pseudonymity.

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

2.3.2. Protocol Transport

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

In addition we highlight the following:

- o The transport does not need to provide either confidentiality or integrity. After GSS-EAP has finished negotiation, GSS-API can be used to provide both services. If the negotiation process itself needs protection from eavesdroppers then the transport would need to provide the necessary services.
- o The transport needs to provide reliable transport of the messages.
- o The transport needs to ensure that tokens are delivered in order during the negotiation process.
- o GSS-API messages need to be delivered atomically. If the transport breaks up a message it must also reassemble the message before delivery.

2.3.3. Reauthentication

There are circumstances where the RP will want to have the client reauthenticate itself. These include very long sessions, where the original authentication is time limited or cases where in order to complete an operation a different authentication is required. GSS-

EAP does not have any mechanism for the server to initiate a reauthentication as all authentication operations start from the client. If a protocol using GSS-EAP needs to support reauthentication that is initiated by the server, then a request from the server to the client for the reauthentication to start needs to be placed in the protocol.

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

3. Application Security Services

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

3.1. Authentication

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

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

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

- o If no target name is configured for the initiator, then the initiator trusts that the acceptor name, supplied by the acceptor, correctly names the entity it is communicating with.
- o Both the initiator and acceptor have the same key material for per-message keys and both parties have confirmed they actually have the key material. In EAP terms, there is a protected indication of success.

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

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

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

3.2. GSS-API Channel Binding

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

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

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

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

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

3.3. Host-Based Service Names

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

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

- o The public-key infrastructure (PKI) used by the web has chosen to have a number of trust anchors (root certificate authorities) each of which can map any host name to a public key.
- o A number of GSS-API mechanisms, such as Kerberos [RFC1964], have split the problem into two parts. A new concept called a realm is introduced, the realm is responsible for host mapping within that realm. The mechanism then decides what realm is responsible for a given name. This is the approach adopted by ABFAB.

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

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

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

Applications may retrieve information about providers of services from DNS. Service Records (SRV) [RFC2782] and Naming Authority Pointer (NAPTR) [RFC2915] records are used to help find a host that provides a service; however the necessity of having DNSSEC on the queries depends on how the information is going to be used. If the

host name returned is not going to be validated by EAP channel binding, because only the service is being validated, then DNSSEC [RFC4033] is not required. However, if the host name is going to be validated by EAP channel binding then DNSSEC needs to be used to ensure that the correct host name is validated. In general, if the information that is returned from the DNS query is to be validated, then it needs to be obtained in a secure manner.

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

3.4. Additional GSS-API Services

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

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

4. Privacy Considerations

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

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

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

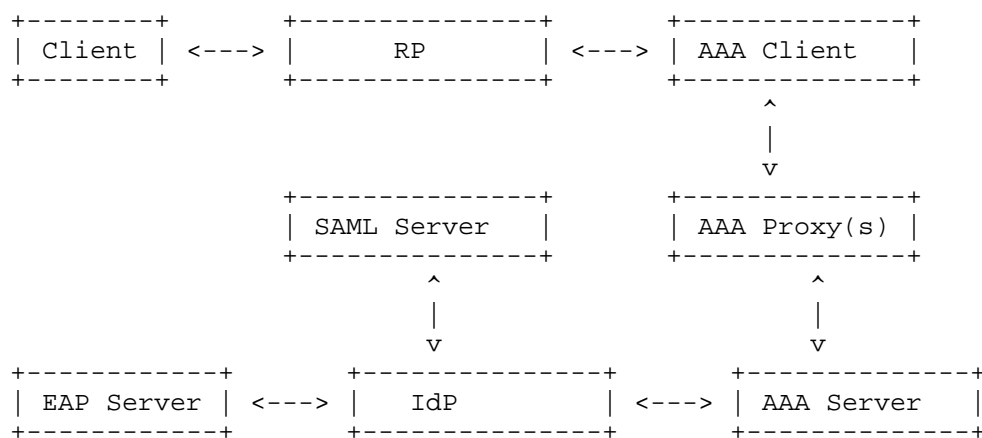


Figure 4: Entities and Data Flow

4.1. Entities and their roles

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

In the Client to RP communication phase, we have:

Initiator: Client.

Observers: Client, RP.

Recipient: RP.

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

Initiator: Client.

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

Recipient: IdP

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

Initiator: RP.

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

Recipient: IdP

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

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

- o If two RPs are colluding, they have the information available to both nodes. This can be analyzed as if a single RP was offering multiple services.
- o If an RP and a AAA proxy are colluding, then the trust of the system is broken as the RP would be able to lie about its own identity to the IdP. There is no known way to deal with this situation.
- o If multiple AAA proxies are colluding, it can be treated as a single node for analysis.

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

In Figure 4 the IdP has been divided into multiple logical pieces, in actual implementations these pieces will frequently be tightly coupled. The links between these pieces provide the greatest

opportunity for attackers and eavesdroppers to acquire information, however, as they are all under the control of a single entity they are also the easiest to have tightly secured.

4.2. Privacy Aspects of ABFAB Communication Flows

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

4.2.1. Client to RP

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

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

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

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

4.2.2. Client to IdP (via Federation Substrate)

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

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

- o The Relying Party knows the IP address of the Client. It is possible that the Relying Party could choose to expose this IP address by including it in a RADIUS header such as Calling Station ID. This is a privacy consideration to take into account of the application protocol.
- o The EAP MSK is transported between the IdP and the RP over the AAA infrastructure, for example through RADIUS headers. This is a particularly important privacy consideration, as any AAA Proxy that has access to the EAP MSK is able to decrypt and eavesdrop on any traffic encrypted using that EAP MSK (i.e., all communications between the Client and RP). This problem can be mitigated by the application protocol setting up a secure tunnel between the Client and the RP and performing a cryptographic binding between the tunnel and EAP MSK.
- o Related to the above, the AAA server has access to the material necessary to derive the session key, thus the AAA server can observe any traffic encrypted between the Client and RP. This "feature" was chosen as a simplification and to make performance faster; if it was decided that this trade-off was not desirable for privacy and security reasons, then extensions to ABFAB that make use of techniques such as Diffie-Helman key exchange would mitigate against this.

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

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

4.2.3. IdP to RP (via Federation Substrate)

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

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

4.3. Relationship between User and Entities

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

be that which is strictly necessary for the operation of the service.

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

4.4. Accounting Information

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

4.5. Collection and retention of data and identifiers

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

However, in cases where Relying Parties require the ability to identify a particular individual (e.g. so they can link this identity information to a particular account in their service, or where identity information is required for audit purposes), the service will need to collect and store such information, and to retain it for as long as they require. Deprovisioning of such accounts and information is out of scope for ABFAB, but obviously for privacy protection any identifiers collected should be deleted when they are no longer needed.

4.6. User Participation

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

5. Security Considerations

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

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

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

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

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

6. IANA Considerations

This document does not require actions by IANA.

7. Acknowledgments

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

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

8. References

8.1. Normative References

- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC4072] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.
- [RFC7055] Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", RFC 7055, December 2013.
- [I-D.ietf-abfab-aaa-saml] Howlett, J. and S. Hartman, "A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML", draft-ietf-abfab-aaa-saml-08 (work in progress), November 2013.

[I-D.ietf-radext-nai]

DeKok, A., "The Network Access Identifier", draft-ietf-radext-nai-05 (work in progress), November 2013.

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

8.2. Informative References

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

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

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

[I-D.ietf-radext-radius-fragmentation]

Perez-Mendez, A., Lopez, R., Pereniguez-Garcia, F., Lopez-Millan, G., Lopez, D., and A. DeKok, "Support of fragmentation of RADIUS packets", draft-ietf-radext-radius-fragmentation-02 (work in progress), November 2013.

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

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

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

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

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

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

- [RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.
- [RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, July 2010.
- [RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.
- [OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.
- [RFC7029] Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, October 2013.
- [I-D.ietf-emu-eap-tunnel-method]
Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel EAP Method (TEAP) Version 1", draft-ietf-emu-eap-tunnel-method-09 (work in progress), September 2013.
- [I-D.ietf-radext-dtls]
DeKok, A., "DTLS as a Transport Layer for RADIUS", draft-ietf-radext-dtls-07 (work in progress), October 2013.
- [I-D.ietf-radext-dynamic-discovery]
Winter, S. and M. McCauley, "NAI-based Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS", draft-ietf-radext-dynamic-discovery-08 (work in progress), October 2013.
- [WS-TRUST]
Lawrence, K., Kaler, C., Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.4", OASIS Standard ws-trust-200902, February 2009, <<http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>>.

- [NIST-SP.800-63] Burr, W., Dodson, D., and W. Polk, "Electronic Authentication Guideline", NIST Special Publication 800-63, April 2006.
- [RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2915] Mealling, M. and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, September 2000.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

8.3. URIs

- [1] <http://www.openid.net>
- [2] <https://community.ja.net/system/files/288/Trust-Router-Overview-IETF86.pptx>
- [3] <https://commmunity.ja.net/system/files/288/Trust-Router-Overview-IETF86.pptx>
- [4] <https://www.eduroam.org>

Editorial Comments

[CREF1] JLS: Should this be an EAP failure to the client as well?

Authors' Addresses

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

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

Sam Hartman
Painless Security

Email: hartmans-ietf@mit.edu

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

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

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

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

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com

ABFAB
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

R. Smith
Cardiff University
July 4, 2014

Application Bridging for Federated Access Beyond web (ABFAB) Usability
and User Interface Considerations
draft-ietf-abfab-usability-ui-considerations-01

Abstract

The real world use of ABFAB-based technologies requires that any identity to be used to authenticate be configured on the ABFAB-enabled client device. Achieving this requires software on that device (either built into the operating system or a standalone utility) that will interact with the user, managing their identity information and identity-to-service mappings. All designers of software to fulfil this role will face the same set of challenges. This document aims to document these challenges with the aim of producing well-thought out UIs with some degree of consistency between implementations.

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 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Conventions	3
3. Terminology	3
4. Context	4
5. Considerations around Terminology	5
5.1. Identity	5
5.2. Services	6
5.3. Identity to Service Mapping	6
6. Considerations around Management of Identities	6
6.1. Information associated with each Identity	6
6.2. Storage of Identity Information	8
6.3. Adding/Association of an Identity	8
6.3.1. Manual Addition	8
6.3.2. Manually Triggered Automated Addition	9
6.3.3. Fully Automated Addition	10
6.4. Modifying Identity Information	10
6.4.1. Manual Modification	11
6.4.2. Automated Modification	11
6.5. Verifying an identity	11
6.6. Removing an Identity	11
6.6.1. Manual Removal	12
6.6.2. Automated Removal	12
6.7. Storing an Identity with or without credentials	12
7. Considerations around Management of Service to Identity Mappings	12
7.1. Associating a Service with an Identity	13
7.1.1. User-driven Manual Association	13
7.1.2. Automated Rules-based Association	13
7.2. Disassociating a Service with an Identity	13
7.3. Listing Services and Identities	14
7.4. Showing the Service that is requesting Authentication	14
7.5. Showing the Identity currently in use	14
7.6. Multiple Identities for a Particular Service	14
7.7. Not using ABFAB for a Particular Service	15
8. Handling of Errors	15
8.1. Identity Association/Verification Errors	15
8.2. Service Errors	15
8.3. Other Errors	15
9. Handling of Successes	15

9.1. Reporting Authentication Success on First Use of Identity	15
9.2. Reporting Authentication Success	16
10. Other Considerations	16
10.1. Identity Selector Taking Focus	16
10.2. Import/Export of Credentials	16
11. Contributors	16
12. Acknowledgements	16
13. Security Considerations	17
14. Privacy Considerations	17
15. IANA Considerations	17
16. Normative References	17
Appendix A. Change Log	18
Appendix B. Open Issues	19

1. Introduction

The use of ABFAB-based technologies requires that any identity to be used to authenticate be configured on the client device. Achieving this requires software on that device (either built into the operating system or a standalone utility) that will interact with the user, and manage the user's identities and credential-to-service mappings. Anyone designing that software will face the same set of challenges.

This document does not intend to supplant evidence-based UI design guidelines; implementers of identity selectors are strongly encouraged to understand the latest in HCI and UX thought and practice. Instead, it aims to document the common challenges faced by implementers with the aim of providing a common starting point for implementers in the hope that this aids in producing well-thought out UIs with some degree of consistency.

2. Conventions

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

3. Terminology

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

- o NAI: Network Access Identifier - a standard way of identifying a user and assisting in the routing of an authentication request (see [RFC4282]).
- o Identity: In this context, an identity is a credential given to a user by a particular organisation with which they have an association. A user may have multiple identities - potentially multiple identities per organisation, and also across multiple organisations. Each identity will consist of an NAI, alongside other information that supports authentication. Note that in other contexts the usual use of "identity" would match our use of "user", whereas the usual use of "identifier" matches our use of identity.
- o Service: The thing that the user is attempting to authenticate to via ABFAB technology. See [I-D.ietf-abfab-usecases] for some example ABFAB use cases. Also known as the Relying Party.
- o Identity Selector: A piece of software that enables the process by which the GSS-API acquires the identity to use with a particular service. An Identity Selector typically would allow the user to configure a set of identities along with service to identity mappings.
- o Trust anchor: An authoritative source of verification of a particular ABFAB service or Identity Provider, used to allow authentication of a server using X.509 [RFC5280]. Typically a commercial CA to allow authentication via chain of trust, or a preconfigured non-commercial certificate (e.g. self-signed).

4. Context

When using the ABFAB architecture (see [I-D.ietf-abfab-arch]) to perform federated authentication to some service, a user will need to provide identity information that they wish to use to authenticate to that particular service. This will happen through a process of the application calling the GSS-API, which will in turn gather the user's credentials through some process. We will call this process the "identity selector" in this document (though note that this is not a recommendation on terminology for the process).

The simplest way to achieve the desired effect would be a process that simply takes the credentials from the currently logged in user (e.g. the Windows Domain Credentials) and uses those for all services that request authenticate through ABFAB. This approach gives ultimate simplicity in terms of UI (it wouldn't have one) but the least flexibility (the user has to use a single identity for everything). If there is ever to be a requirement for a user to use

a different set of credentials for a service, then something more complex will be needed.

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

- o They could be configured manually by the user in a configuration file that could be edited by hand or some such simple process, and read by the GSS-API mechanism. While this could work very well functionally, in practice only a small subset of users would be happy with - and able to - configure their identities in such a manner.
- o They could be configured through some interactive process. For ease of use this should have a simple UI, although to support some use cases a headless mode (i.e. a way of interacting with the identity selector when there is no GUI present) may need to be supported.

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

5. Considerations around Terminology

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

5.1. Identity

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

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

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

5.2. Services

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

5.3. Identity to Service Mapping

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

6. Considerations around Management of Identities

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

6.1. Information associated with each Identity

The bare minimum set of information that MUST be stored about each identity to allow ABFAB authentication to take place is a single item:

- o NAI: The user's Network Access Identifier (see [RFC4282]) for this particular credential. For example, "joe@example.com". Note that the identity selector MUST NOT store different identities that use the same NAI. This is required as the NAI is the unique key that is used by the identity selector when interacting with the GSS-API mechanism for various reasons, for example, to allow the GSS-API mechanism to report back error or success statuses.

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

- o Trust anchor: For the identity selector to be able to verify that the server it is going to talk to and attempt to authenticate against is the server that it is expecting, and that it is not being spoofed in some way. This is likely to be an X.509 certificate [RFC5280], or a tuple of (trusted root certificate, servername in Subject or subjectAltName). Storing a credential without a relevant trust anchor allows for the possibility of a malicious attacker intercepting traffic and masquerading as the server in question.
- o Credential: Whatever is used by the user to authenticate themselves with a particular NAI. What exactly this will be will be dependent on the EAP method being used, but is likely to be something like a password or a certificate. Note that this is a SHOULD, rather than a MUST, because there are use cases where a user may specifically opt for this not to be "remembered".

Finally, there is a set of optional information that MAY be stored about each identity that represent useful information for the user to have and could make an identity selector more usable. Note that this list is neither intended to be exhaustive or even particularly correct; any implementer is free to use whatever make sense in their implementation and conforms to good HCI/UX guidelines. Instead, it is simply a suggested starting point.

- o Friendly name for identity: To allow the user to differentiate between the set of identities represented in the Identity Selector. This should be editable by the user. The only restriction on this name is that it MUST be unique within that particular user's set of identities. For example: "My University", "Google Account", "Work Login", etc.
- o Friendly icon for identity: To allow the user to differentiate between the set of identities they have they should be able to set an icon for that particular identity.
- o Password changing URL: The URL the user should visit should they need to change their password for this particular identity. For example, "http://www.example.com/passwordreset".
- o Helpdesk URL: The URL the user should visit to get contact details for the helpdesk of the organisation that issued this particular identity for when the user encounters issues and needs help. For example, https://www.example.com/helpdesk.

6.2. Storage of Identity Information

Since some of the information that makes up the identity is sensitive in nature (e.g. containing passwords), then this information SHOULD be stored and accessed securely. This might involve ensuring the credential information is held in encrypted form on device and accessed using a passphrase. For deeper integration into the system, this could be done by using existing secure storage on the system such as Keychain on a Mac or the GNOME keyring on a GNOME based Linux device.

6.3. Adding/Association of an Identity

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

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

6.3.1. Manual Addition

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

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

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

- o Asynchronously: the user could be allowed to, at any time, trigger a workflow of manually adding an identity. This represents the most flexible way of adding an identity since a user can perform this at any time. It does, however, also have inherent issues when it comes to verifying the newly added identity - see Section 6.5.
- o Just In Time: when connecting to a service which has no mapping to an existing identity, the user could be given an option to add a new one, as well as associating with an existing one. This seems to present a better user experience when it comes to verifying the newly added identity (see Section 6.5), however, it represents a less direct method of adding an identity. Users who have not yet added the appropriate identity to their identity selector may find it difficult to understand that they must try to access a particular service in order to add an identity.

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

Finally, the area of verification of trust anchors is very important. An Identity Selector that allows for manual addition of identity information SHOULD try to ensure that trust anchor information is gathered and checked in a secure a manner as possible - where users have to enter and confirm all trust anchor information, or be required to explicitly agree to an insecure configuration if this is not done properly.

6.3.2. Manually Triggered Automated Addition

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

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

It is reasonable to assume that any such automated addition service is likely to be organisation specific, so that the Issuing Organisation and realm part of the NAI will be constant, as would be the trust anchor information. The user part of their NAI will have

been input on the web service. The password could be provided as a part of the provided data or the identity selector could prompt the user to enter it.

Additionally, the user SHOULD be given the opportunity to:

- o Supply or change the default friendly name for that identity - to allow the user to customise the identifier they use for that identity;
- o Indicate whether or not the identity selector should always ask before using services with this identity - to customise the way in which the identity selector interacts with the user with this particular identity;
- o Reject the addition of the identity completely - to allow the user to back out of the association process in an intuitive way.

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

6.3.3. Fully Automated Addition

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

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

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

6.4. Modifying Identity Information

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

6.4.1. Manual Modification

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

6.4.2. Automated Modification

To ease usability, organisations may wish to automatically provide updates to identity information. For example, if the user's password changes it could automatically update the password for the identity in the user's identity selector, or if the trust anchor information changes (e.g. if a certificate is changed) it could be automatically pushed out to all users.

6.5. Verifying an identity

An inherent by-product of the ABFAB architecture is that an identity cannot be verified during the addition process; it can only be verified while it is in use with a real service. This represents a definite usability issue no matter which method of identity addition is used (see Section 6.3):

- o If the user has manually added the identity (see Section 6.3) they may have gone through the whole manual process with no errors and so believe the identity has been set up correctly. However, when they attempt to access a service, they may be given an error message, thus causing some amount of confusion.
- o If the user has had the identity populated into their identity selector, then there is a much greater chance of the identity information being correct. However, if any of the information is not correct, then there is the potential for confusion as the user did not add the information in the first place.

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

6.6. Removing an Identity

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

6.6.1. Manual Removal

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

6.6.2. Automated Removal

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

6.7. Storing an Identity with or without credentials

Sometimes, a user may wish to have the identity they wish to use with a service stored by the identity selector, but not the credential (e.g. password) that goes along with that Identity. The consequence of this is that when a user attempts to authenticate to a service for which an identity, but no credential, is stored, then the user would need to be prompted to manually enter the credential.

7. Considerations around Management of Service to Identity Mappings

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

This potentially complex many-to-many association between identities and services is not easily comprehended by the user, and allowing the user to both manipulate it and control can be challenging. These obstacles are especially common when errors occur after an association has been made. In this scenario it is important that an identity can be disassociated with a service.

To further complicate the picture, users may wish for:

1. The identity to service mapping to be stored along with the credential, i.e. the user should always be authenticated to a particular service with a particular identity with no prompting.
2. The identity to service mapping to be stored but not the credential, i.e. the user should not be prompted to choose the identity for a particular service, but should be prompted to enter their credential for that identity.

3. The identity to service mapping to not be stored, i.e. the user should be asked which identity to use every time they authenticate to a particular service.

7.1. Associating a Service with an Identity

There needs to be a way for the user to create the service to identity association. It is advisable that this link be made only after the identity in question has authenticated with the service without any error.

There are a few ways this association could happen.

7.1.1. User-driven Manual Association

There are two ways in which manual association of an identity to a service could happen:

1. The user could manually associate a particular service with a particular identity using the identity selector before they first attempt to use the service. In order to do so, however, the user would need to know all the required technical details of that service beforehand, such as its GSS Acceptor Name.
2. On encountering a service new to the identity selector, the identity selector could pop up a dialogue box to the user asking if they would like to use an existing identity for this service (and might also allow them to create a new identity and use that).

7.1.2. Automated Rules-based Association

It would be beneficial from a usability perspective to minimise - or avoid entirely - situations where the user has to pick an identity for a particular service. This could be accomplished by having rules to describe services and their mapping to identities. Such a rule could match, for example, a particular identity for all IMAP servers, or a particular identity for all services in a given service realm. These rules could be configured as a part of the automated identity addition process described in Section 6.3.2 or Section 6.3.3

7.2. Disassociating a Service with an Identity

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

There should also be provision for the automated disassociation of an identity with a service for appropriate types of authentication failures.

7.3. Listing Services and Identities

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

7.4. Showing the Service that is requesting Authentication

When a user is attempting to authenticate to a service for the first time, there should be some indication given to the user as to which service is requesting authentication. In many cases, the service may be obvious (where the user has started the process of attempting to authenticate to a particular service), but in other cases this may not be obvious (e.g. if an authentication attempt is triggered by a timer or a specific event), and for this scenario some indication as to the requesting service is necessary.

7.5. Showing the Identity currently in use

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

Implementing such a feature may be hard, however, due to the layered nature of the ABFAB transaction - the identity selector will certainly know when successful (or failed) authentications to a particular service have happened, but after that it typically plays no further part in the use of the service. Therefore, knowing that a particular service is still using a particular identity in order to indicate this to the user would be challenging.

7.6. Multiple Identities for a Particular Service

An Identity Selector should be able to deal with the case where a user has multiple identities associated with a single service. For example, upon receiving a request for authentication to a service that multiple identities are configured for, ask the user which of the identities should be used in this instance.

7.7. Not using ABFAB for a Particular Service

There may be cases where a user does not wish to use ABFAB based authentication at all to a particular service, even though it is ABFAB enabled. To support this, the identity selector would have to allow the user to choose not to use ABFAB when they attempt to authenticate to a service. It would be desirable if the user could also flag that this should be remembered.

8. Handling of Errors

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

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

TODO: Lots more to discuss here...

8.1. Identity Association/Verification Errors

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

8.2. Service Errors

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

8.3. Other Errors.

TODO: e.g. network errors. TODO.

9. Handling of Successes

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

9.1. Reporting Authentication Success on First Use of Identity

The first time an identity is used with a service, it would be good practice to visually indicate in some way that the process has been

successful, in order that the user understands what is happening and is then prepared for future authentication attempts.

9.2. Reporting Authentication Success

On an on-going basis you may or may not wish to indicate visually to the user a successful authentication to a service. This relates to Section 7.5.

10. Other Considerations

This section briefly discusses other considerations that you might want to think about that don't fit in any of the other categories.

10.1. Identity Selector Taking Focus

When an ABFAB authentication request is triggered, and where it needs input from the user, the Identity Selector should take focus in some way so that it is clear to the user that they need to do something to proceed.

10.2. Import/Export of Credentials

For various reasons, an identity selector implementation might want to include functionality that allows for the export/import of identities and service to identity mappings. This could be for backup purposes, to allow a degree of mobility between identity selector instances, etc.

If providing this functionality, it would be advisable that the credential store that is the result of the export should be secure - encrypted and password protected - given the nature of the information.

11. Contributors

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

12. Acknowledgements

Jim Schaad, Stefan Winter, David Chadwick, Kevin Wasserman, Mark Donally, Dave Crocker.

13. Security Considerations

TODO: Bad trust anchors, no trust anchors, phishing.

14. Privacy Considerations

TODO: See Arch for general discussion. Particular to UI - storing credentials on a particular device, mobility considerations.

15. IANA Considerations

This document does not require actions by IANA.

16. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [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-12 (work in progress), February 2014.
- [I-D.ietf-abfab-usecases]
Smith, R., "Application Bridging for Federated Access Beyond web (ABFAB) Use Cases", draft-ietf-abfab-usecases-05 (work in progress), September 2012.

Appendix A. Change Log

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

IETF draft -00 to ietf draft -01

1. Tidying up language throughout
2. Doing some of the TODOs
3. Added language that tries to explain that this document is not a substitute for good HCI/UX design.
4. Changed terminology slightly to avoid confusion between an identity selector "mechanism" and a GSS-API mechanism.
5. Added a caveat about the potential for the UI to show the identity currently in use for a particular service.
6. Added a requirement that the identity selector must not store the same NAI for multiple identities.
7. Stopped talking about "provisioning" after saying that I wouldn't talk about "provisioning".

Draft -04 to ietf draft -00

1. Adding brief discussion of identities vs identifiers (Ken).
2. Changing assumption about credentials having a password in favour of more generic text for other auth types.
3. Adding discussion of storage of identity information.
4. Added sections on dealing with multiple identities per service, remembering credentials, remembering not to use ABFAB.
5. Added small section on ID selector needing to take focus in some way.

Draft -03 to draft -04

1. Addressing various comments from Jim and Stefan.

Draft -02 to draft -03

1. Bumping version to keep it alive.

Draft -01 to draft -02

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

Draft -00 to draft -01

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

Appendix B. Open Issues

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

Author's Address

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

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