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

S. Cantor
Internet2

May 27, 2010

A SASL Mechanism for SAML Enhanced Clients
draft-cantor-ietf-sasl-saml-ec-00.txt

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on November 28, 2010.

Copyright Notice

Table of Contents

1. Introduction

Security Assertion Markup Language (SAML) 2.0 [OASIS.saml-core-2.0-os] is a modular specification that provides various means for a user to be identified to a relying party (RP) through the exchange of (typically signed) assertions issued by an identity provider (IdP). It includes a number of protocols, protocol bindings [OASIS.saml-bindings-2.0-os], and interoperability profiles [OASIS.saml-profiles-2.0-os] designed for different use cases.

Simple Authentication and Security Layer (SASL) [RFC4422] is a generalized mechanism for identifying and authenticating a user and for optionally negotiating a security layer for subsequent protocol interactions. SASL is used by application protocols like IMAP, POP and XMPP. The effect is to make authentication modular, so that newer authentication mechanisms can be added as needed.

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

The SAML mechanism described in this document is an adaptation of an existing SAML profile, the Enhanced Client or Proxy (ECP) Profile [OASIS.saml-profiles-2.0-os], and therefore does not establish a separate authentication, integrity and confidentiality mechanism. It is anticipated that existing security layers, such as Transport Layer Security (TLS), will continued to be used.

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

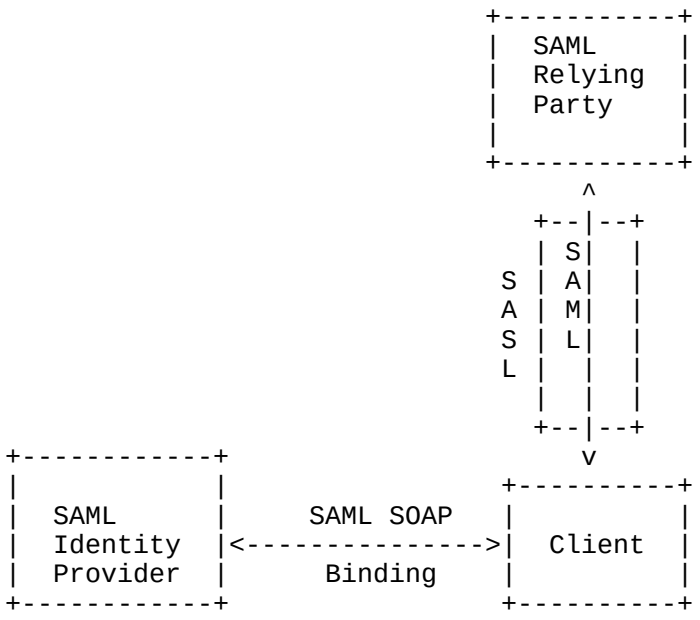

Figure 1: Interworking Architecture

2. Terminology

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

The reader is also assumed to be familiar with the terms used in the SAML 2.0 specification, and an understanding of the Enhanced Client or Proxy (ECP) Profile [OASIS.saml-profiles-2.0-os] is necessary, as part of this mechanism explicitly reuses and references it.

3. Applicability for Non-HTTP Use Cases

While SAML is designed to support a variety of application scenarios, the profiles for authentication defined in the original standard are designed around HTTP applications. They are not, however, limited to browsers, because it was recognized that browsers suffer from a variety of functional and security deficiencies that would be useful to avoid where possible. Specifically, the notion of an "Enhanced Client" (or a proxy acting as one on behalf of a browser, thus the term "ECP") was specified for a software component that acted like a browser from an application perspective, but included sufficient awareness of SAML to play a more conscious role in the authentication exchange between the RP and the IdP. What follows is an outline of the Enhanced Client or Proxy (ECP) Profile [OASIS.saml-profiles-2.0-os], as applied to the web/HTTP service use case:

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

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

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

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

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

This completes the SAML exchange.

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

When considering this flow in the context of an arbitrary application protocol and SASL, the RP and the client both must change their code to implement this SASL mechanism, but the IdP can remain untouched. The existing RP/client exchange that is tunneled through HTTP also maps well to the tunneling of that same exchange in SASL. In the parlance of SASL [RFC4422], this mechanism is "variable", in that the client can accompany its authentication request with an "initial response" consisting of a SAML <Response> obtained from an IdP. The steps are shown from below:

1. The server MAY advertise the SAML20EC capability.

2. The client initiates a SASL authentication with SAML20EC. It MAY include an initial response.

3. The server sends the client one of two responses:

    1. an indication of success or failure (if the client included an initial response).

    2. a challenge containing a BASE64-encoded SOAP envelope containing a SAML <AuthnRequest>.

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

5. Upon completion of the exchange with the IdP, the client responds to the SASL server with a BASE64-encoded SOAP envelope containing the SAML <Response> it obtained, or a SOAP fault, as warranted.

6. The SASL Server indicates success or failure.

Note: The details of the SAML processing, which are consistent with the existing Enhanced Client or Proxy (ECP) Profile [OASIS.saml-profiles-2.0-os], are such that the client MUST interact with the IdP in order to complete any SASL exchange with the RP. The assertions issued by the IdP for the purposes of the profile, and by extension this SASL mechanism, are short lived, and therefore cannot be cached by the client for later use.

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

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

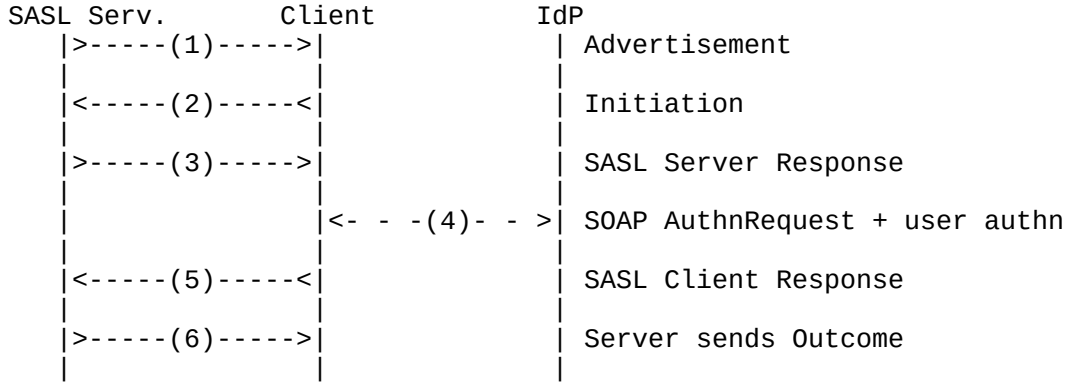

Figure 2: Authentication flow (no initial response)

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

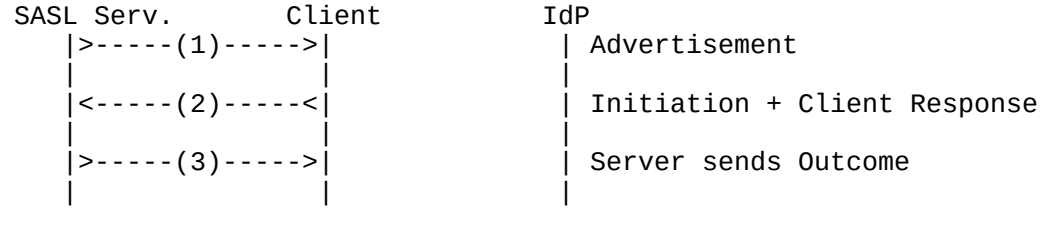

Figure 3: Authentication flow (with initial response)

4. SAML SASL Mechanism Specification

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

4.1. Advertisement

To advertise that a server supports this mechanism, during application session initiation, it displays the name "SAML20EC" in the list of supported SASL mechanisms.

4.2. Initiation

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

4.3. Server Response

Assuming no initial response from the client, the SASL server responds with a BASE64 [RFC4648] encoded SOAP envelope constructed in accordance with section 4.2.3.2 of [OASIS.saml-profiles-2.0-os]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing profile.

4.4. User Authentication with Identity Provider

Upon receipt of the Server Response (Section 4.3), the steps described in sections 4.2.3.3 through 4.2.3.6 of [OASIS.saml-profiles-2.0-os] are performed between the client and the chosen IdP. The means by which the client determines the IdP to use, and where it is located, are out of scope of this mechanism. The exact means of authentication to the IdP are also out of scope, but clients supporting this mechanism MUST support HTTP Basic Authentication as defined in [RFC2617] and SHOULD support client authentication via TLS as defined in [RFC5246].

4.5. Client Response

Assuming a response is obtained from the IdP, the client responds to the SASL server with a BASE64 [RFC4648] encoded SOAP envelope constructed in accordance with section 4.2.3.7 of [OASIS.saml-profiles-2.0-os]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing profile. If the client is unable to obtain a response from the IdP, it responds to the SASL server with a base64-encoded SOAP envelope

containing a SOAP fault.

4.6. Outcome

The SAML protocol exchange having completed, the SASL server will transmit the outcome to the client.

4.7. Additional Notes

Because this mechanism is an adaptation of an HTTP-based profile, there are a few requirements outlined in [OASIS.saml-profiles-2.0-os] that make reference to a response URL that is normally used to regulate where the client returns information to the RP. There are also security-related checks built into the profile that involve this location.

For compatibility with existing IdP and profile behavior, one or more URLs MUST be associated with the SASL server and used to populate the responseConsumerURL and AssertionConsumerServiceURL XML attributes described in the profile. The parties then perform the steps described in [OASIS.saml-profiles-2.0-os] as usual.

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

5. Example

Suppose the user has an identity at the SAML IdP saml.example.org and a Jabber Identifier (jid) "somenode@example.com", and wishes to authenticate his XMPP connection to xmpp.example.com (and example.com and example.org have established a SAML-capable trust relationship). The authentication on the wire would then look something like the following:

Step 1: Client initiates stream to server:

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'
to='example.com' version='1.0'>
```

Step 2: Server responds with a stream tag sent to client:

```
<stream:stream
xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams'
id='some_id' from='example.com' version='1.0'>
```

Step 3: Server informs client of available authentication mechanisms:

```
<stream:features>
 <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <mechanism>DIGEST-MD5</mechanism>
  <mechanism>PLAIN</mechanism>
  <mechanism>SAML20EC</mechanism>
 </mechanisms>
</stream:features>
```

Step 4: Client selects an authentication mechanism:

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

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

<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUNCiAgICB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoy
LjA6YXNzZXJ0aW9uIg0KICAgIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6cHJvdG9jb2wiDQogICAgeG1sbnM6Uz0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5v
cmcvc29hcC9lbnZlbG9wZS8iPg0KICA8UzpIZWFkZXI+DQogICAgPHBhb3M6UmVxdWVzdCB4
bWxuczpwYW9zPSJ1cm46bGliZXJ0eTpwYW9zOjIwMDMtMDgiDQogICAgICBtZXNzYWdlSUQ9
ImMzYTRmOGI5YzJkIiBTOm11c3RVbmRlcnN0YW5kPSIxIg0KICAgICAgUzphY3Rvcj0iaHR0
cDovL3NjaGVtYXMueG1sc29hcC5vcmcvc29hcC9hY3Rvci9uZXh0Ig0KICAgICAgcmVzcG9u
c2VDb25zdW1lclVSTD0iaHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tIg0KICAgICAgc2Vydmlj
ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb2ZpbGVzOlNTTzplY3AiLz4NCiAg
ICA8ZWNwOlJlcXVlc3QNCiAgICAgIHhtbG5zOmVjcD0idXJuOm9hc2lzOm5hbWVzOnRjOlNB
TUw6Mi4wOnByb2ZpbGVzOlNTTzplY3AiDQogICAgICBTOmFjdG9yPSJodHRwOi8vc2NoZW1h
cy54bWxzb2FwLm9yZy9zb2FwL2FjdG9yL25leHQiDQogICAgICBTOm11c3RVbmRlcnN0YW5k
PSIxIiBQcm92aWRlck5hbWU9IkphYmJlciBhdCBleGFtcGxlLmNvbSI+DQogICAgICA8c2Ft
bDpJc3N1ZXI+aHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tPC9zYW1sOklzc3Vlcj4NCiAgICA8
L2VjcDpSZXF1ZXN0Pg0KICA8L1M6SGVhZGVyPg0KICA8UzpCb2R5Pg0KICAgIDxzYW1sOkFz
c2VydGlvbkNvbnN1bWVyU2VydmljZVVSTD0iaHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tIj4N
CiAgICAgIDxzYW1sOklzc3VlciB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6YXNzZXJ0aW9uIj4NCiAgICAgICBodHRwczovL3htcHAuZXhhbXBsZS5jb20NCiAg
ICAgIDwvc2FtbDpJc3N1ZXI+DQogICAgICA8c2FtbHA6TmFtZUlEUG9saWN5IEFsbG93Q3Jl
YXRlPSJ0cnVlIg0KICAgICAgICBGb3JtYXQ9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIu
MDpuYW1laWQtZm9ybWF0OnBlcnNpc3RlbnQiLz4NCiAgICA8L3NhbWxwOlJlcXVlc3Q+DQog
ICAgPHNhbWw6QXV0aG5SZXF1ZXN0Pg0KICA8L1M6Qm9keT4NCjwvUzpFbnZlbG9w
ZT4NCg==
</challenge>

The decoded envelope:

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

Step 5 (alt): Server returns error to client:

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

Step 6: Client relays the request to IdP in a SOAP message transmitted over HTTP (over TLS). HTTP portion not shown, use of

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

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

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

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

Step 8: Client sends BASE64 [RFC4648] encoded SOAP envelope

containing the SAML <Response> as a response to the SASL server's challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUNCiAgICB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoy
LjA6YXNzZXJ0aW9uIg0KICAgIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6cHJvdG9jb2wiDQogICAgeG1sbnM6Uz0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5v
cmcvc29hcC9lbnZlbG9wZS8iPg0KICA8UzpIZWFkZXI+DQogICAgPHBhb3M6UmVzcG9uc2Ug
eG1sbnM6cGFvcz0idXJuOmxpYmVydHk6cGFvczoyMDAzLTA4Ig0KICAgICAgUzphY3Rvcj0i
aHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvc29hcC9hY3Rvci9uZXh0Ig0KICAgICAgUzpt
dXN0VW5kZXJzdGFuZD0iMSIgcmVmVG9NZXNzYWdlSUQ9IjZjM2E0ZjhiOWMyZCIvPg0KICA8
L1M6SGVhZGVyPg0KICA8UzpCb2R5Pg0KICAgIDxzYW1scDpSZXNwb25zZSBJRD0iZDQzaDk0
cjM4OTMwOXIiIFZlcnNpb249IjIuMCINCiAgICAgICAgSXNzdWVJbnN0YW50PSIyMDA3LTEy
LTEwVDExOjQyOjM0WiIgSW5SZXNwb25zZVRvPSJjM2E0ZjhiOWMyZCINCiAgICAgICAgRGVz
dGluYXRpb249Imh0dHBzOi8veG1wcC5leGFtcGxlLmNvbSI+DQogICAgICA8c2FtbDpJc3N1
ZXI+aHR0cHM6Ly9zYW1sLmV4YW1wbGUub3JnPC9zYW1sOklzc3Vlcj4NCiAgICAgIDxzYW1s
cDpTdGF0dXM+DQogICAgICAgIDxzYW1scDpTdGF0dXNDb2RlDQogICAgICAgICAgICBWYWx1
ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1czpTdWNjZXNzIi8+DQogICAg
ICA8L3NhbWxwOlN0YXR1cz4NCiAgICAgIDxzYW1sOkFzc2VydGlvbiB4bWxuczpzYW1sPSJ1
cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIg0KICAgICAgICAgICAgSUQ9
Ii1lSBjb250ZW50cyBlbGlkZWQgLS0+DQogICAgICA8L3NhbWw6RW5jcnlwdGVk
QXNzZXJ0aW9uPg0KICAgIDwvc2FtbHA6UmVzcG9uc2U+DQogICAgPHVzZXJfaHA6UmVzcG9u
dmVsb3BlPg0K
</response>
```

The decoded envelope:

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

Step 9: Server informs client of successful authentication:

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

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

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

Step 10: Client initiates a new stream to server:

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'
to='example.com' version='1.0'>
```

Step 11: Server responds by sending a stream header to client along with any additional features (or an empty features element):

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'
id='c2s_345' from='example.com' version='1.0'>
<stream:features>
 <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
 <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</stream:features>
```

Step 12: Client binds a resource:

```
   <iq type='set' id='bind_1'>
     <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
       <resource>someresource</resource>
     </bind>
   </iq>
```

Step 13: Server informs client of successful resource binding:

```
   <iq type='result' id='bind_1'>
     <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
       <jid>somenode@example.com/someresource</jid>
     </bind>
   </iq>
```

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

6. Security Considerations

This section will address only security considerations associated with the use of SAML with SASL applications. For considerations relating to SAML in general, the reader is referred to the SAML specification and to other literature. Similarly, for general SASL Security Considerations, the reader is referred to that specification.

6.1. Risks Left Unaddressed

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

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

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

6.2. User Privacy

The IdP is aware of each RP that a user logs into. There is nothing in the protocol to hide this information from the IdP. It is not a requirement to track the activity, but there is nothing technically that prohibits the collection of information. SASL servers should be aware that SAML IdPs will track - to some extent - user access to their services.

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

6.3. Collusion between RPs

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

7. IANA Considerations

The IANA is requested to register the following SASL profile:

SASL mechanism profile: SAML20EC

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

8. Normative References

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

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

[OASIS.saml-profiles-2.0-os]
Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., and E. Maler, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard OASIS.saml-profiles-2.0-os, March 2005.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

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

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

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

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

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

Appendix A. Acknowledgments

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

Appendix B. Changes

This section to be removed prior to publication.

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

Author's Address

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

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 23, 2012

J. Howlett
JANET(UK)
S. Hartman
Painless Security
October 21, 2011

Key Negotiation Protocol (KNP)
draft-howlett-radsec-knp-02

Abstract

The Key Negotiation Protocol enables an untrusting RADIUS client and RADIUS server to derive a key by reference to a mutually trusted actor called the Introducer. This key may subsequently be used for one of two purposes. First, it can credential a TLS PSK ciphersuite applied to a RadSec connection between the RADIUS client and RADIUS server; or secondly, to establish a trust relationship between the RADIUS client and a second Introducer that is trusted by the first Introducer.

The composition of these capabilities enables a RADIUS client to establish a RadSec connection with any RADIUS server with whom it shares a direct or indirect trust relationship via one or more Introducers.

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 April 23, 2012.

Copyright Notice

Table of Contents

1. Introduction

TLS encryption for RADIUS (RadSec) [I-D.ietf-radext-radsec] provides a mechanism for securing the communication between a RADIUS [RFC2865] client and server on the transport layer by using TLS [RFC5246].

RadSec mandates the use of one of the [RFC5246] ciphersuites and recommends the use of two other ciphersuites specified in that document. However any ciphersuite, including the TLS Pre-Shared Key (PSK) ciphersuites [RFC4279], may be used providing that it supports encryption.

The Key Negotiation Protocol enables an untrusting RADIUS client and RADIUS server to derive a key by means of a mutually trusted actor called the Introducer. This key may subsequently for two purposes.

First, the key can be used to credential a TLS PSK ciphersuite when applied to a RadSec connection between the RADIUS client and RADIUS server, permitting a trusted exchange of RADIUS messages in the absence of a pre-existing relationship between the RADIUS client and RADIUS server. This is described as "Credentialing".

Secondly, the key can used as a credential by a RADIUS client to establish a trust relationship with a second Introducer that happens to be trusted by the first Introducer. This is described as "Introduction".

The composition of Credentialing and Introduction enables a RADIUS client to establish a RadSec connection with any RADIUS server with whom it shares an indirect trust relationship via one or more Introducers.

2. Motivation

The KNP is motivated by the following requirements:

- In the case of a non-federated RADIUS environment where a RADIUS client and RADIUS AS.server shares a direct trust relationship, a shared secret credential is used as the trust anchor between these systems. In transitioning to the use of RadSec, it may be more convenient if these systems are able to continue using the existing credential technology rather than introduce a new credential technology (such as X.509 certificates), as this may impose significant changes to operational practices (such as deploying a Public Key Infrastructure).

- In the case of a federated RADIUS environment where RADIUS clients and RADIUS servers are associated with different domains,

transitioning to the use of RadSec may impose a requirement to distribute and manage multiple trust anchors. It may be more convenient if the systems within these domains were able to use a single trust anchor for RADIUS systems in all other domains, in addition to those systems within its own domain. This may facilitate the scaling of large heterogeneous RADIUS environments where it may be difficult - for technical and/or administrative reasons - to impose support for even a small set of trust anchors.

- o The use of multiple trust anchors within complex federated environments may impede essential trust management functions such as timely revocation. Reducing the number of trust anchors may therefore improve trust management within these environments, particularly if it can be reduced to a single trust anchor.

3. Overview

The Key Negotiation Protocol (KNP) enables a RADIUS client and RADIUS server that do not share a direct trust relationship to derive a shared key by virtue of both systems having a trust relationship with an EAP server called the Introducer. This key may be used for the following purposes:

1. Credentialing: the RADIUS client and RADIUS server can use the key to credential a TLS PSK ciphersuite applied to a RadSec connection.

2. Introduction: a credential can be derived from the key that can be used to authenticate the RADIUS client against a second Introducer that is trusted by the first Introducer.

The composition of these capabilities enables a RADIUS client to derive a key that can be used to credential a RadSec connection with any other RADIUS server with whom it shares a common Introducer and, through transitivity, any number of intermediate Introducers.

This transitivity of trust between a RADIUS client and RADIUS server across a chain of intermediate Introducers may appear very similar to the use of RADIUS proxies to establish a chain of trust between a RADIUS client and RADIUS server. There is however a very significant difference:

- o In the case of RADIUS proxy, the RADIUS messages emitted by the RADIUS client and RADIUS server must transit through the intermediate RADIUS proxy(ies). There is no end-to-end relationship between the RADIUS client and RADIUS server, either in terms of connectivity or trust.

- o In the case of KNP, the RADIUS messages are able to transit directly between RADIUS client and RADIUS server. The path of transmissions between these systems is therefore entirely decoupled from the path of trust . There is an end-to-end relationship between the RADIUS client and RADIUS server, both in terms of connectivity and trust.

The use of RADIUS Proxies and Introducers are not mutually exclusive; deployers may choose to use both.

4. Conventions used in this document

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

5. The KNP Actors

In the KNP, the RADIUS client and RADIUS server do not initially share a trust relationship. Instead, these actors share a trust relationship with a mutually trusted third party known as the "Introducer".

Figure 1 below depicts the trust relationships for a RADIUS client, RADIUS server and Introducer before the KNP has been invoked.

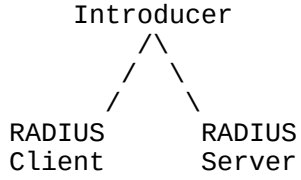

Figure 1

Figure 2 below depicts the new trust relationship between the RADIUS client, RADIUS server and Introducer after the KNP has been invoked.

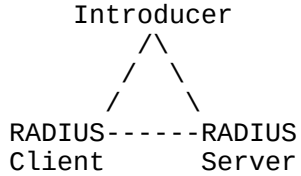

Figure 2

Figure 3 below depicts the flow of RADIUS packets from the RADIUS

client to the RADIUS server using the new trust relationship.

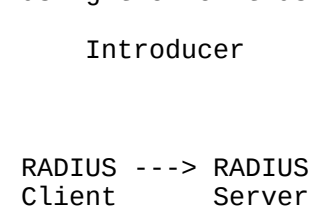

Figure 3

Note that the RADIUS messages are not routed by the Introducer, as they would in the case of a RADIUS Proxy. Instead, they flow directly from RADIUS client to RADIUS server.

6. Relationships With Other Protocols

The KNP builds on a variety of protocols. This section describes the relationship of KNP to these.

6.1. Relationship to EAP

In the KNP the RADIUS client assumes the role of an EAP peer. In this role, it attempts to authenticate against a RADIUS server that assumes the role of a pass-through EAP authenticator. An EAP server acts as the Introducer.

The KNP enables all three actors - RADIUS client (EAP peer), RADIUS server (EAP authenticator) and Introducer (EAP server) - to establish a common view of their mutual relationships as described by the EAP names and keys that the EAP exchange yields, using the norms established by the EAP Key Management Framework [RFC5247].

The RADIUS client must possess an EAP credential for the Introducer, allowing mutual authentication of both parties.

Figure 4 below depicts the relationships between these actors:

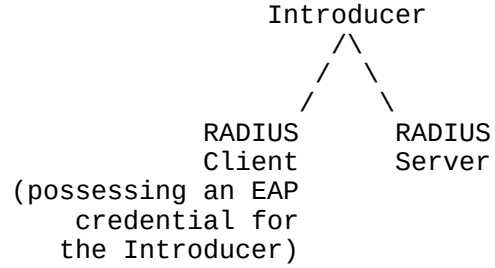

Figure 4

6.2. Relationship to RADIUS

The RADIUS server uses the RADIUS protocol to forward the EAP transaction to the Introducer.

The RADIUS server must share a RADIUS secret with the Introducer, allowing mutual authentication of both actors.

Figure 5 below depicts the relationships between these actors:

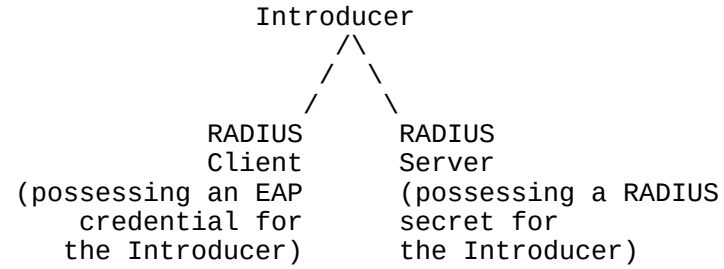

Figure 5

6.3. Relationship to the GSS API

The KNP builds on the GSS API [RFC2743] framework by using the GSS EAP mechanism [I-D.ietf-abfab-gss-eap] and EAP [RFC3748]. The GSS EAP tokens are transported between the RADIUS client and RADIUS server using the HTTP Negotiate authentication scheme [RFC4559].

6.4. Relationship to the HTTP

The KNP uses HTTP to transport its request and response protocol messages between the RADIUS Client and RADIUS server.

7. Key Negotiation Protocol

As described previously, the KNP provides two operations: Credentialing and Introduction.

The KNP provides these operations using a common protocol pattern. This pattern is applied against two types of Target actor, depending on the operation in question:

- In the case of Credentialing, the Target actor is a RADIUS server. If Credentialing is successful, the RADIUS client and RADIUS server will derive a common PSK that can be applied with a TLS-PSK

ciphersuite and RadSec.

- In the case of Introduction, the Target actor is an Introducer. If an Introduction is successful, the RADIUS client and Introducer will derive an EAP credential that can subsequently be used for subsequent Credentialing or Introduction operations.

For both operations it is essential that all three actors - RADIUS Client, Introducer and Target (whether a RADIUS server, in the case of Credentialing, or another Introducer, in the case of Introduction) - are able to authorise the claims that the other actors make about their respective names. These claims are validated using different processes for each relationship; these are summarised in Figure 6 below.

Subject	Relying Party	Process	Evidence from
RADIUS Client	Introducer	GSS EAP authentication	EAP method w/ qualifying Security Claims
Introducer	RADIUS Client		
Introducer	Target	RADIUS authentication	RADIUS shared secret
Target	Introducer		
Target	RADIUS Client	Channel bindings	Assertion by Introducer
RADIUS Client	Target	RADIUS attribute	Assertion by Introducer

Figure 6

7.1. Operation Independent Flow

The RADIUS Client invokes the KNP by establishing an HTTP connection with the Target’s KNP end-point.

The RADIUS Client MUST use the GSS EAP mechanism [I-D.ietf-abfab-gss-eap] to authenticate to the Introducer, requesting mutual authentication from the GSS layer.

The RADIUS Client, Target and Introducer MUST support EAP channel bindings [I-D.ietf-emu-chbind]. The Introducer MUST use validate the EAP channel bindings [I-D.ietf-emu-chbind] provided by the RADIUS Client. If the channel binding verification fails, the Introducer MUST reject the authentication.

The completion of the EAP method exchange results in the derivation of an EAP MSK known only to the RADIUS Client and Introducer and Peer-Id(s) and Server-Id(s) identifying these respectively. The Introducer MUST replicate the keying material and Server-Id to the Target.

The RADIUS Client and Target, in possession of the EAP MSK, create a GSS-API security context as described in section 6 of [I-D.ietf-abfab-gss-eap].

The RADIUS Client POSTs a key negotiation request, encoded as an HTML form dataset, to the Target. This request contains a set of operation-specific controls that specifies key negotiation parameters. A key negotiation request MUST contain the following controls:

- Version: the version of the KNP.
- Request-Identifier: a unique alphanumeric identifier for the request.
- Requestor-Name: the requestor’s GSS EAP initiator name.
- Operation-Type: the type of operation.
- Authenticator-Type: message authentication algorithm.
- Authenticator-Value: message authenticator value.

The Target extracts the key negotiation parameters and assesses their compliance to the Target’s key negotiation policies. The Target MUST return an operation-specific document providing information about the resulting key negotiation context.

- Version: the version of the KNP.
- Request-Identifier: the identifier for the request that this is a reponse to.
- Responder-Name: the requestor’s GSS EAP acceptor name.
- Operation-Type: the type of operation.
- Status-Code: a status code.
- Expires-After: a timestamp indicating the time of expiration.

- Authenticator-Type: message authentication algorithm.

- Authenticator-Value: message authenticator value.

TODO: consider use of SAML authentication assertion instead?

The RADIUS server and client SHOULD cache the GSS context until expiry of the GSS context. However, either party MAY delete a GSS context at any time. If a GSS context is deleted, any operation-specific derived materials SHOULD also be deleted, although such materials MAY be retained for auditing purposes.

7.2. The Credentialing Operation

This section describes the Credentialing operation-specific extensions to the general KNP flow.

The RADIUS Client MUST specify the following control values within the key negotiation request:

- Operation-Type: Credentialing

The PSK identity and value shall be outputs of GSS_Pseudo_random() [RFC4401] using the Pseudo-Random Function defined for the GSS EAP mechanism [I-D.ietf-abfab-gss-eap].

For the PSK identity, the prf_in input string MUST be prepended with the string "tls-psk-knp-identity"; desired_out_len MUST be set to 128 octets.

For the PSK value, the prf_in input string MUST be prepended with the string "tls-psk-knp-value"; desired_out_len MUST be set to 64 octets.

Note: these output values should use base64 encoding

7.3. The Introduction Operation

This section describes the Introduction operation-specific extensions to the general KNP flow.

The RADIUS Client MUST specify the following control values within the key negotiation request:

- Operation-Type: Introduction"

The EAP identity and credential shall be outputs of GSS_Pseudo_random() [RFC4401] using the Pseudo-Random Function defined for the GSS EAP mechanism [I-D.ietf-abfab-gss-eap].

For the EAP identity, the prf_in input string MUST be prepended with the string "tls-psk-eap-identity"; desired_out_len MUST be set to 128 octets. The output string MUST be appended with the realm of the Introducer to form an NAI.

For the EAP credential, the prf_in input string MUST be prepended with the string "tls-psk-eap-psk"; desired_out_len MUST be set to 64 octets.

Note: these output values should use base64 encoding.

8. Security Considerations

TODO

9. IANA Considerations

TODO

10. Acknowledgements

TODO

11. Normative References

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

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

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

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

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4401] Williams, N., "A Pseudo-Random Function

(PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.

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

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.

[I-D.ietf-abfab-gss-eap] Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", draft-ietf-abfab-gss-eap-03 (work in progress), October 2011.

[I-D.ietf-radext-radsec] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "TLS encryption for RADIUS", draft-ietf-radext-radsec-09 (work in progress), July 2011.

[I-D.ietf-emu-chbind] Hartman, S., Clancy, T., and K. Hoeper, "Channel Binding Support for EAP Methods", draft-ietf-emu-chbind-10 (work in progress), October 2011.

Authors' Addresses

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

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

Sam Hartman
Painless Security

EMail: hartmans-ietf@mit.edu

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

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

May 31, 2012

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

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on December 2, 2012.

Copyright Notice

Table of Contents

1. Conventions used in this document

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

2. Introduction

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

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

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

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

3. Name Attribute Authenticity

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

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

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

4. Name Attributes/Values as ACL Subjects

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

5. Naming Contexts

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

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

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

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

acceptor and federation.

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

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

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

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

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

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

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

6. Representation of Attribute Names

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

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

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

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

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

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

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

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

7. API

7.1. SET OF OCTET STRING

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

7.2. Const types

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

else from that document is required to implement this document.

7.3. GSS_Display_name_ext()

Inputs:

- name INTERNAL NAME,
- display_as_name_type OBJECT IDENTIFIER

Outputs:

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

Return major_status codes:

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

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

7.3.1. C-Bindings

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

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

7.4. GSS_Inquire_name()

Inputs:

- name INTERNAL NAME

Outputs:

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

Return major_status codes:

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

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

7.4.1. C-Bindings

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

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

7.5. GSS_Get_name_attribute()

Inputs:

- name INTERNAL NAME,
- attr OCTET STRING

Outputs:

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

Return major_status codes:

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

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

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

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

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

7.5.1. C-Bindings

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

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

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

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

7.6. GSS_Set_name_attribute()

Inputs:

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

- attr OCTET STRING,
- values SET OF OCTET STRING

Outputs:

- major_status INTEGER,
- minor_status INTEGER

Return major_status codes:

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

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

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

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

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

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

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

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

7.6.1. C-Bindings

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

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

7.7. GSS_Delete_name_attribute()

Inputs:

- name INTERNAL NAME,
- attr OCTET STRING,

Outputs:

- major_status INTEGER,
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates no error.

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

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

7.7.1. C-Bindings

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

7.8. GSS_Export_name_composite()

Inputs:

- name INTERNAL NAME

Outputs:

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

Return major_status codes:

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

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

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

The OID for GSS_C_NT_COMPOSITE_EXPORT is <TBD>.

7.8.1. C-Bindings

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

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

8. IANA Considerations

This specification has no actions for IANA.

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

9. Security Considerations

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

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

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

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

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

10. References

10.1. Normative References

[GFD.024] Argonne National Laboratory, National Center for Supercomputing Applications, Argonne National Laboratory, and Argonne National Laboratory, "GSS-API Extensions", GFD GFD.024, June 2004.

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

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

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

[RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.

10.2. Informative References

[ANSI.X3-4.1986]

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

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

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

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

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

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

Authors' Addresses

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com

Leif Johansson
Swedish University Network
Thulegatan 11
Stockholm
Sweden

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

Sam Hartman
Painless Security

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

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

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

RADIUS Extensions Working Group
Internet-Draft
Intended status: Experimental
Expires: August 17, 2012

S. Winter
RESTENA
M. McCauley
OSC
S. Venaas
K. Wierenga
Cisco

February 14, 2012

Transport Layer Security (TLS) encryption for RADIUS
draft-ietf-radext-radsec-12

Abstract

This document specifies a transport profile for RADIUS using Transport Layer Security (TLS) over TCP as the transport protocol. This enables dynamic trust relationships between RADIUS servers.

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, 2012.

Copyright Notice

Table of Contents

1. Introduction

The RADIUS protocol [RFC2865] is a widely deployed authentication and authorisation protocol. The supplementary RADIUS Accounting specification [RFC2866] also provides accounting mechanisms, thus delivering a full Authentication, Authorization, and Accounting (AAA) solution. However, RADIUS is experiencing several shortcomings, such as its dependency on the unreliable transport protocol UDP and the lack of security for large parts of its packet payload. RADIUS security is based on the MD5 algorithm, which has been proven to be insecure.

The main focus of RADIUS over TLS is to provide a means to secure the communication between RADIUS/TCP peers using TLS. The most important use of this specification lies in roaming environments where RADIUS packets need to be transferred through different administrative domains and untrusted, potentially hostile networks. An example for a world-wide roaming environment that uses RADIUS over TLS to secure communication is "eduroam", see [eduroam].

There are multiple known attacks on the MD5 algorithm which is used in RADIUS to provide integrity protection and a limited confidentiality protection (see [MD5-attacks]). RADIUS over TLS wraps the entire RADIUS packet payload into a TLS stream and thus mitigates the risk of attacks on MD5.

Because of the static trust establishment between RADIUS peers (IP address and shared secret) the only scalable way of creating a massive deployment of RADIUS-servers under control by different administrative entities is to introduce some form of a proxy chain to route the access requests to their home server. This creates a lot of overhead in terms of possible points of failure, longer transmission times as well as middleboxes through which authentication traffic flows. These middleboxes may learn privacy-relevant data while forwarding requests. The new features in RADIUS over TLS obsolete the use of IP addresses and shared MD5 secrets to identify other peers and thus allow the use of more contemporary trust models, e.g. checking a certificate by inspecting the issuer and other certificate properties.

1.1. Requirements Language

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

1.2. Terminology

RADIUS/TLS node: a RADIUS over TLS client or server

RADIUS/TLS Client: a RADIUS over TLS instance which initiates a new connection.

RADIUS/TLS Server: a RADIUS over TLS instance which listens on a RADIUS over TLS port and accepts new connections

RADIUS/UDP: classic RADIUS transport over UDP as defined in [RFC2865]

1.3. Document Status

This document is an Experimental RFC.

It is one out of several approaches to address known cryptographic weaknesses of the RADIUS protocol (see also Section 4). The specification does not fulfill all recommendations on a AAA transport profile as per [RFC3539]; in particular, by being based on TCP as a transport layer, it does not prevent head-of-line blocking issues.

If this specification is indeed selected for advancement to standards track, certificate verification options (section 2.3.2) need to be refined.

Another experimental characteristic of this specification is the question of key management between RADIUS/TLS peers. RADIUS/UDP only allowed for manual key management, i.e. distribution of a shared secret between a client and a server. RADIUS/TLS allows manual distribution of long-term proofs of peer identity as well (by using TLS-PSK cipher suites, or identifying clients by a certificate fingerprint), but as a new feature enables use of X.509 certificates in a PKIX infrastructure. It remains to be seen if one of these methods prevail, or if both will find their place in real-life deployments. The authors can imagine pre-shared keys to be popular in small-scale deployments (SOHO or isolated enterprise deployments) where scalability is not an issue and the deployment of a CA is considered too much a hassle; but can also imagine large roaming consortia to make use of PKIX. Readers of this specification are encouraged to read the discussion of key management issus within [RFC6421] as well as [RFC4107].

It has yet to be decided whether this approach is to be chosen for standards track. One key aspect to judge whether the approach is usable at large scale is by observing the uptake, usability and operational behaviour of the protocol in large-scale, real-life deployments.

An example for a world-wide roaming environment that uses RADIUS over TLS to secure communication is "eduroam", see [eduroam].

2. Normative: Transport Layer Security for RADIUS/TCP

2.1. TCP port and packet types

The default destination port number for RADIUS over TLS is TCP/2083. There are no separate ports for authentication, accounting and dynamic authorisation changes. The source port is arbitrary. See section Section 3.4 for considerations regarding separation of authentication, accounting and dynamic authorization traffic.

2.2. TLS negotiation

RADIUS/TLS has no notion of negotiating TLS in an established connection. Servers and clients need to be preconfigured to use RADIUS/TLS for a given endpoint.

2.3. Connection Setup

RADIUS/TLS nodes

1. establish TCP connections as per [I-D.ietf-radext-tcp-transport]. Failure to connect leads to continuous retries, with exponentially growing intervals between every try. If multiple servers are defined, the node MAY attempt to establish a connection to these other servers in parallel, in order to implement quick failover.

2. after completing the TCP handshake, immediately negotiate TLS sessions according to [RFC5246] or its predecessor TLS 1.1. The following restrictions apply:

   * Support for TLS v1.1 [RFC4346] or later (e.g. TLS 1.2 [RFC5246]]) is REQUIRED. To prevent known attacks on TLS versions prior to 1.1, implementations MUST NOT negotiate TLS versions prior to 1.1.

   * Support for certificate-based mutual authentication is REQUIRED.

   * Negotiation of mutual authentication is REQUIRED.

   * Negotiation of a ciphersuite providing for confidentiality as well as integrity protection is REQUIRED. Failure to comply with this requirement can lead to severe security problmes, like user passwords being recoverable by third parties. See

Section 6 for details.

* Support for and negotiation of compression is OPTIONAL.

* Support for TLS-PSK mutual authentication [RFC4279] is OPTIONAL.

* RADIUS/TLS implementations MUST at a minimum support negotiation of the TLS_RSA_WITH_3DES_EDE_CBC_SHA), and SHOULD support TLS_RSA_WITH_RC4_128_SHA and TLS_RSA_WITH_AES_128_CBC_SHA as well (see Section 3.3).

* In addition, RADIUS/TLS implementations MUST support negotiation of the mandatory-to-implement ciphersuites required by the versions of TLS that they support.

3. Peer authentication can be performed in any of the following three operation models:

   * TLS with X.509 certificates using PKIX trust models (this model is mandatory to implement):

     + Implementations MUST allow to configure a list of trusted Certification Authorities for incoming connections.

     + Certificate validation MUST include the verification rules as per [RFC5280].

     + Implementations SHOULD indicate their trusted Certification Authorities (CAs). For TLS 1.2, this is done using [RFC5246] section 7.4.4 "certificate authorities" (server side) and [RFC6066] Section 6 "Trusted CA Indication" (client side). See also Section 3.2.

     + Peer validation always includes a check on whether the locally configured expected DNS name or IP address of the server that is contacted matches its presented certificate. DNS names and IP addresses can be contained in the Common Name (CN) or subjectAltName entries. For verification, only one of these entries is to be considered. The following precedence applies: for DNS name validation, subjectAltName:DNS has precedence over CN; for IP address validation, subjectAltName:iPAddr has precedence over CN. Implementors of this specification are advised to read [RFC6125] Section 6 for more details on DNS name validation.

+ Implementations MAY allow to configure a set of additional properties of the certificate to check for a peer's authorisation to communicate (e.g. a set of allowed values in subjectAltName:URI or a set of allowed X509v3 Certificate Policies)

+ When the configured trust base changes (e.g. removal of a CA from the list of trusted CAs; issuance of a new CRL for a given CA) implementations MAY re-negotiate the TLS session to re-assess the connecting peer's continued authorisation.

* TLS with X.509 certificates using certificate fingerprints (this model is optional to implement): Implementations SHOULD allow to configure a list of trusted certificates, identified via fingerprint of the DER encoded certificate octets. Implementations MUST support SHA-1 as the hash algorithm for the fingerprint. To prevent attacks based on hash collisions, support for a more contemporary hash function such as SHA-256 is RECOMMENDED.

* TLS using TLS-PSK (this model is optional to implement)

4. start exchanging RADIUS datagrams (note Section 3.4 (1)). The shared secret to compute the (obsolete) MD5 integrity checks and attribute encryption MUST be "radsec" (see Section 3.4 (2)).

2.4. Connecting Client Identity

In RADIUS/UDP, clients are uniquely identified by their IP address. Since the shared secret is associated with the origin IP address, if more than one RADIUS client is associated with the same IP address, then those clients also must utilize the same shared secret, a practice which is inherently insecure as noted in [RFC5247].

RADIUS/TLS supports multiple operation modes.

In TLS-PSK operation, a client is uniquely identified by its TLS identifier.

In TLS-X.509 mode using fingerprints, a client is uniquely identified by the fingerprint of the presented client certificate.

In TLS-X.509 mode using PKIX trust models, a client is uniquely identified by the tuple (serial number of presented client certificate;Issuer).

Note well: having identified a connecting entity does not mean the

server necessarily wants to communicate with that client. E.g. if the Issuer is not in a trusted set of Issuers, the server may decline to perform RADIUS transactions with this client.

There are numerous trust models in PKIX environments, and it is beyond the scope of this document to define how a particular deployment determines whether a client is trustworthy. Implementations which want to support a wide variety of trust models should expose as many details of the presented certificate to the administrator as possible so that the trust model can be implemented by the administrator. As a suggestion, at least the following parameters of the X.509 client certificate should be exposed:

- Originating IP address
- Certificate Fingerprint
- Issuer
- Subject
- all X509v3 Extended Key Usage
- all X509v3 Subject Alternative Name
- all X509v3 Certificate Policies

In TLS-PSK operation, at least the following parameters of the TLS connection should be exposed:

- Originating IP address
- TLS Identifier

2.5. RADIUS Datagrams

Authentication, Accounting and Authorization packets are sent according to the following rules:

RADIUS/TLS clients transmit the same packet types on the connection they initiated as a RADIUS/UDP client would (see Section 3.4 (3) and (4)). E.g. they send

- Access-Request
- Accounting-Request

- Status-Server
- Disconnect-ACK
- Disconnect-NAK
- ...

and they receive

- Access-Accept
- Accounting-Response
- Disconnect-Request
- ...

RADIUS/TLS servers transmit the same packet types on connections they have accepted as a RADIUS/UDP server would. E.g. they send

- Access-Challenge
- Access-Accept
- Access-Reject
- Accounting-Response
- Disconnect-Request
- ...

and they receive

- Access-Request
- Accounting-Request
- Status-Server
- Disconnect-ACK
- ...

Due to the use of one single TCP port for all packet types, it is required for a RADIUS/TLS server to signal to a connecting peer which types of packets are supported on a server. See also section

Section 3.4 for a discussion of signaling.

- When receiving an unwanted packet of type 'CoA-Request' or 'Disconnect-Request', it needs to be replied to with a 'CoA-NAK' or 'Disconnect-NAK' respectively. The NAK SHOULD contain an attribute Error-Cause with the value 406 ("Unsupported Extension"); see [RFC5176] for details.

- When receiving an unwanted packet of type 'Accounting-Request', the RADIUS/TLS server SHOULD reply with an Accounting-Response containing an Error-Cause attribute with value 406 "Unsupported Extension" as defined in [RFC5176]. A RADIUS/TLS accounting client receiving such an Accounting-Response SHOULD log the error and stop sending Accounting-Request packets.

3. Informative: Design Decisions

This section explains the design decisions that led to the rules defined in the previous section.

3.1. Implications of Dynamic Peer Discovery

One mechanism to discover RADIUS over TLS peers dynamically via DNS is specified in [I-D.ietf-radext-dynamic-discovery]. While this mechanism is still under development and therefore is not a normative dependency of RADIUS/TLS, the use of dynamic discovery has potential future implications that are important to understand.

Readers of this document who are considering the deployment of DNS-based dynamic discovery are thus encouraged to read [I-D.ietf-radext-dynamic-discovery] and follow its future development.

3.2. X.509 Certificate Considerations

(1) If a RADIUS/TLS client is in possession of multiple certificates from different CAs (i.e. is part of multiple roaming consortia) and dynamic discovery is used, the discovery mechanism possibly does not yield sufficient information to identify the consortium uniquely (e.g. DNS discovery). Subsequently, the client may not know by itself which client certificate to use for the TLS handshake. Then it is necessary for the server to signal which consortium it belongs to, and which certificates it expects. If there is no risk of confusing multiple roaming consortia, providing this information in the handshake is not crucial.

(2) If a RADIUS/TLS server is in possession of multiple certificates from different CAs (i.e. is part of multiple roaming consortia), it

will need to select one of its certificates to present to the RADIUS/TLS client. If the client sends the Trusted CA Indication, this hint can make the server select the appropriate certificate and prevent a handshake failure. Omitting this indication makes it impossible to deterministically select the right certificate in this case. If there is no risk of confusing multiple roaming consortia, providing this indication in the handshake is not crucial.

3.3. Ciphersuites and Compression Negotiation Considerations

Not all TLS ciphersuites in [RFC5246] are supported by available TLS tool kits, and licenses may be required in some cases. The existing implementations of RADIUS/TLS use OpenSSL as cryptographic backend, which supports all of the ciphersuites listed in the rules in the normative section.

The TLS ciphersuite TLS_RSA_WITH_3DES_EDE_CBC_SHA is mandatory-to-implement according to [RFC4346] and thus has to be supported by RADIUS/TLS nodes.

The two other ciphersuites in the normative section are widely implemented in TLS toolkits and are considered good practice to implement.

3.4. RADIUS Datagram Considerations

(1) After the TLS session is established, RADIUS packet payloads are exchanged over the encrypted TLS tunnel. In RADIUS/UDP, the packet size can be determined by evaluating the size of the datagram that arrived. Due to the stream nature of TCP and TLS, this does not hold true for RADIUS/TLS packet exchange. Instead, packet boundaries of RADIUS packets that arrive in the stream are calculated by evaluating the packet's Length field. Special care needs to be taken on the packet sender side that the value of the Length field is indeed correct before sending it over the TLS tunnel, because incorrect packet lengths can no longer be detected by a differing datagram boundary. See section 2.6.4 of [I-D.ietf-radext-tcp-transport] for more details.

(2) Within RADIUS/UDP [RFC2865], a shared secret is used for hiding of attributes such as User-Password, as well as in computation of the Response Authenticator. In RADIUS accounting [RFC2866], the shared secret is used in computation of both the Request Authenticator and the Response Authenticator. Since TLS provides integrity protection and encryption sufficient to substitute for RADIUS application-layer security, it is not necessary to configure a RADIUS shared secret. The use of a fixed string for the obsolete shared secret eliminates possible node misconfigurations.

(3) RADIUS/UDP [RFC2865] uses different UDP ports for authentication, accounting and dynamic authorisation changes. RADIUS/TLS allocates a single port for all RADIUS packet types. Nevertheless, in RADIUS/TLS the notion of a client which sends authentication requests and processes replies associated with it's users' sessions and the notion of a server which receives requests, processes them and sends the appropriate replies is to be preserved. The normative rules about acceptable packet types for clients and servers mirror the packet flow behaviour from RADIUS/UDP.

(4) RADIUS/UDP [RFC2865] uses negative ICMP responses to a newly allocated UDP port to signal that a peer RADIUS server does not support reception and processing of the packet types in [RFC5176]. These packet types are listed as to be received in RADIUS/TLS implementations. Note well: it is not required for an implementation to actually process these packet types; it is only required to send the NAK as defined above.

(5) RADIUS/UDP [RFC2865] uses negative ICMP responses to a newly allocated UDP port to signal that a peer RADIUS server does not support reception and processing of RADIUS Accounting packets. There is no RADIUS datagram to signal an Accounting NAK. Clients may be misconfigured to send Accounting packets to a RADIUS/TLS server which does not wish to process their Accounting packet. To prevent a regression of detectability of this situation, the Accounting-Response + Error-Cause sgnaling was introduced.

4. Compatibility with other RADIUS transports

Ongoing work in the IETF defines multiple alternative transports to the classic UDP transport model as defined in [RFC2865], namely RADIUS over TCP [I-D.ietf-radext-tcp-transport], RADIUS over Datagram Transport Layer Security (DTLS) [I-D.ietf-radext-dtls] and this present document on RADIUS over TLS.

RADIUS/TLS does not specify any inherent backwards compatibility to RADIUS/UDP or cross compatibility to the other transports, i.e. an implementation which implements RADIUS/TLS only will not be able to receive or send RADIUS packet payloads over other transports. An implementation wishing to be backward or cross compatible (i.e. wishes to serve clients using other transports than RADIUS/TLS) will need to implement these other transports along with the RADIUS/TLS transport and be prepared to send and receive on all implemented transports, which is called a multi-stack implementation.

If a given IP device is able to receive RADIUS payloads on multiple transports, this may or may not be the same instance of software, and it may or may not serve the same purposes. It is not safe to assume

that both ports are interchangeable. In particular, it can not be assumed that state is maintained for the packet payloads between the transports. Two such instances MUST be considered separate RADIUS server entities.

5. Diameter Compatibility

Since RADIUS/TLS is only a new transport profile for RADIUS, compatibility of RADIUS/TLS - Diameter [RFC3588] vs. RADIUS/UDP [RFC2865] - Diameter [RFC3588] is identical. The considerations regarding payload size in [I-D.ietf-radext-tcp-transport] apply.

6. Security Considerations

The computational resources to establish a TLS tunnel are significantly higher than simply sending mostly unencrypted UDP datagrams. Therefore, clients connecting to a RADIUS/TLS node will more easily create high load conditions and a malicious client might create a Denial-of-Service attack more easily.

Some TLS ciphersuites only provide integrity validation of their payload, and provide no encryption. This specification forbids the use of such ciphersuites. Since the RADIUS payload’s shared secret is fixed to the well-known term "radsec" (see Section 2.3 (4)) , failure to comply with this requirement will expose the entire datagram payload in plain text, including User-Password, to intermediate IP nodes.

By virtue of being based on TCP, there are several generic attack vectors to slow down or prevent the TCP connection from being established; see [RFC4953] for details. If a TCP connection is not up when a packet is to be processed, it gets re-established, so such attacks in general lead only to a minor performance degradation (the time it takes to re-establish the connection). There is one notable exception where an attacker might create a bidding-down attack though: If peer communication between two devices is configured for both RADIUS/TLS (i.e TLS security over TCP as a transport, shared secret fixed to "radsec") and RADIUS/UDP (i.e. shared secret security with a secret manually configured by the administrator), and where the RADIUS/UDP transport is the failover option if the TLS session cannot be established, a bidding-down attack can occur if an adversary can maliciously close the TCP connection, or prevent it from being established. Situations where clients are configured in such a way are likely to occur during a migration phase from RADIUS/ UDP to RADIUS/TLS. By preventing the TLS session setup, the attacker can reduce the security of the packet payload from the selected TLS cipher suite packet encryption to the classic MD5 per-attribute encryption. The situation should be avoided by disabling the weaker

RADIUS/UDP transport as soon as the new RADIUS/TLS connection is established and tested. Disabling can happen at either the RADIUS client or server side:

- Client side: de-configure the failover setup, leaving RADIUS/TLS as the only communication option
- Server side: de-configure the RADIUS/UDP client from the list of valid RADIUS clients

RADIUS/TLS provides authentication and encryption between RADIUS peers. In the presence of proxies, the intermediate proxies can still inspect the individual RADIUS packets, i.e. "end-to-end" encryption is not provided. Where intermediate proxies are untrusted, it is desirable to use other RADIUS mechanisms to prevent RADIUS packet payload from inspection by such proxies. One common method to protect passwords is the use of the Extensible Authentication Protocol (EAP) and EAP methods which utilize TLS.

When using certificate fingerprints to identify RADIUS/TLS peers, any two certificates which produce the same hash value (i.e. which have a hash collision) will be considered the same client. It is therefore important to make sure that the hash function used is cryptographically uncompromised so that an attacker is very unlikely to be able to produce a hash collision with a certificate of his choice. While this specification mandates support for SHA-1, a later revision will likely demand support for more contemporary hash functions because as of issuance of this document there are already attacks on SHA-1.

7. IANA Considerations

No new RADIUS attributes or packet codes are defined. IANA is requested to update the already-assigned TCP port number 2083 in the following ways:

- Reference: list the RFC number of this document as the reference
- Assignment Notes: add the text "The TCP port 2083 was already previously assigned by IANA for "RadSec", an early implementation of RADIUS/TLS, prior to issuance of this RFC. This early implementation can be configured to be compatible to RADIUS/TLS as specified by the IETF. See RFC (RFC number of this document), Appendix A for details."

8. Notes to the RFC Editor

[I-D.ietf-radext-tcp-transport] is currently in the publication queue because it has a normative reference on this draft; it has no other blocking dependencies. The two drafts should be published as an RFC simultaneously, ideally with consequtive numbers. The references in this draft to [I-D.ietf-radext-tcp-transport] should be changed to references to the corresponding RFC prior to publication.

This section, "Notes to the RFC Editor" should be deleted from the draft prior to publication.

9. Acknowledgements

RADIUS/TLS was first implemented as "RADSec" by Open Systems Consultants, Currumbin Waters, Australia, for their "Radiator" RADIUS server product (see [radsec-whitepaper]).

Funding and input for the development of this Internet Draft was provided by the European Commission co-funded project "GEANT2" [geant2] and further feedback was provided by the TERENA Task Force Mobility [terena].

10. References

10.1. 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.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, 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.

[RFC5176] Chiba, M., Dommety, G., Eklund, M., Mitton, D., and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.

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

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.

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

[I-D.ietf-radext-tcp-transport] DeKok, A., "RADIUS Over TCP", draft-ietf-radext-tcp-transport-09 (work in progress), October 2010.

10.2. Informative References

[I-D.ietf-radext-dtls] DeKok, A., "DTLS as a Transport Layer for RADIUS", draft-ietf-radext-dtls-01 (work in progress), October 2010.

[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-03 (work in

progress), July 2011.

[RFC3539] Aboba, B. and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6421] Nelson, D., "Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)", RFC 6421, November 2011.

[radsec-whitepaper] Open System Consultants, "RadSec - a secure, reliable RADIUS Protocol", May 2005, <http://www.open.com.au/radiator/radsec-whitepaper.pdf>.

[MD5-attacks] Black, J., Cochran, M., and T. Highland, "A Study of the MD5 Attacks: Insights and Improvements", October 2006, <http://www.springerlink.com/content/40867l85727r7084/>.

[radsecproxy-impl] Venaas, S., "radsecproxy Project Homepage", 2007, <http://software.uninett.no/radsecproxy/>.

[eduroam] Trans-European Research and Education Networking Association, "eduroam Homepage", 2007, <http://www.eduroam.org/>.

[geant2] Delivery of Advanced Network Technology to Europe, "European Commission Information Society and Media: GEANT2", 2008, <http://www.geant2.net/>.

[terena] TERENA, "Trans-European Research and Education Networking Association", 2008, <http://www.terena.org/>.

Appendix A. Implementation Overview: Radiator

Radiator implements the RadSec protocol for proxying requests with the <Authby RADSEC> and <ServerRADSEC> clauses in the Radiator configuration file.

The <AuthBy RADSEC> clause defines a RadSec client, and causes Radiator to send RADIUS requests to the configured RadSec server using the RadSec protocol.

The <ServerRADSEC> clause defines a RadSec server, and causes Radiator to listen on the configured port and address(es) for connections from <Authby RADSEC> clients. When an <Authby RADSEC> client connects to a <ServerRADSEC> server, the client sends RADIUS requests through the stream to the server. The server then handles the request in the same way as if the request had been received from a conventional UDP RADIUS client.

Radiator is compliant to RADIUS/TLS if the following options are used:

<AuthBy RADSEC>

* Protocol tcp

* UseTLS

* TLS_CertificateFile

* Secret radsec

<ServerRADSEC>

* Protocol tcp

* UseTLS

* TLS_RequireClientCert

* Secret radsec

As of Radiator 3.15, the default shared secret for RadSec connections is configurable and defaults to "mysecret" (without quotes). For compliance with this document, this setting needs to be configured for the shared secret "radsec". The implementation uses TCP keepalive socket options, but does not send Status-Server packets. Once established, TLS connections are kept open throughout the server instance lifetime.

Appendix B. Implementation Overview: radsecproxy

The RADIUS proxy named radsecproxy was written in order to allow use of RadSec in current RADIUS deployments. This is a generic proxy that supports any number and combination of clients and servers, supporting RADIUS over UDP and RadSec. The main idea is that it can be used on the same host as a non-RadSec client or server to ensure RadSec is used on the wire, however as a generic proxy it can be used in other circumstances as well.

The configuration file consists of client and server clauses, where there is one such clause for each client or server. In such a clause one specifies either "type tls" or "type udp" for RadSec or UDP transport. For RadSec the default shared secret "mysecret" (without quotes), the same as Radiator, is used. For compliance with this document, this setting needs to be configured for the shared secret "radsec". A secret may be specified by putting say "secret somesharedsecret" inside a client or server clause.

In order to use TLS for clients and/or servers, one must also specify

where to locate CA certificates, as well as certificate and key for the client or server. This is done in a TLS clause. There may be one or several TLS clauses. A client or server clause may reference a particular TLS clause, or just use a default one. One use for multiple TLS clauses may be to present one certificate to clients and another to servers.

If any RadSec (TLS) clients are configured, the proxy will at startup listen on port 2083, as assigned by IANA for the OSC RadSec implementation. An alternative port may be specified. When a client connects, the client certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests coming from a RadSec client are treated exactly like requests from UDP clients.

The proxy will at startup try to establish a TLS connection to each (if any) of the configured RadSec (TLS) servers. If it fails to connect to a server, it will retry regularly. There is some back-off where it will retry quickly at first, and with longer intervals later. If a connection to a server goes down it will also start retrying regularly. When setting up the TLS connection, the server certificate will be verified, including checking that the configured FQDN or IP address matches what is in the certificate. Requests are sent to a RadSec server just like they would to a UDP server.

The proxy supports Status-Server messages. They are only sent to a server if enabled for that particular server. Status-Server requests are always responded to.

This RadSec implementation has been successfully tested together with Radiator. It is a freely available open-source implementation. For source code and documentation, see [radsecproxy-impl].

Appendix C. Assessment of Crypto-Agility Requirements

The RADIUS Crypto-Agility Requirements [RFC6421] defines numerous classification criteria for protocols that strive to enhance the security of RADIUS. It contains mandatory (M) and recommended (R) criteria which crypto-agile protocols have to fulfill. The authors believe that the following assessment about the crypto-agility properties of RADIUS/TLS are true.

By virtue of being a transport profile using TLS over TCP as a transport protocol, the cryptographically agile properties of TLS are inherited, and RADIUS/TLS subsequently meets the following points:

(M) negotiation of cryptographic algorithms for integrity and auth

(M) negotiation of cryptographic algorithms for encryption

(M) replay protection

(M) define mandatory-to-implement cryptographic algorithms

(M) generate fresh session keys for use between client and server

(R) support for Perfect Forward Secrecy in session keys

(R) support X.509 certificate based operation

(R) support Pre-Shared keys

(R) support for confidentiality of the entire packet

(M/R) support Automated Key Management

The remainder of the requirements is discussed individually below in more detail:

(M) "avoid security compromise, even in situations where the existing cryptographic alogrithms used by RADIUS implementations are shown to be weak enough to provide little or no security" - The existing algorithm, based on MD5, is not of any significance in RADIUS/TLS; its compromise does not compromise the outer transport security.

(R) mandatory-to-implement alogrithms are to be NIST-Acceptable with no deprecation date - The mandatory-to-implement algorithm is TLS_RSA_WITH_3DES_EDE_CBC_SHA. This ciphersuite supports three-key 3DES operation, which is classified as Acceptable with no known deprecation date by NIST.

(M) demonstrate backward compatibility with RADIUS - There are multiple implementations supporting both RADIUS and RADIUS/TLS, and the translation between them.

(M) After legacy mechanisms have been compromised, secure algorithms MUST be used, so that backward compatibility is no longer possible - In RADIUS, communication between client and server is always a manual configuration; after a compromise, the legacy client in question can be de-configured by the same manual configuration.

(M) indicate a willingness to cede change control to the IETF - Change control of this protocol is with the IETF.

(M) be interoperable between implementations based purely on the information in the specification - At least one implementation was created exclusively based on this specification and is interoperable with other RADIUS/TLS implementations.

(M) apply to all packet types - RADIUS/TLS operates on the transport layer, and can carry all packet types.

(R) message data exchanged with Diameter SHOULD NOT be affected - The solution is Diameter-agnostic.

(M) discuss any inherent assumptions - The authors are not aware of any implicit assumptions which would be yet-unarticulated in the draft

(R) provide recommendations for transition - The Security Considerations section contains a transition path.

(R) discuss legacy interoperability and potential for bidding-down attacks - The Security Considerations section contains an corresponding discussion.

Summarizing, it is believed that this specification fulfills all the mandatory and all the recommended requirements for a crypto-agile solution and should thus be considered UNCONDITIONALLY COMPLIANT.

Authors' Addresses

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

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

Mike McCauley
Open Systems Consultants
9 Bulbul Place
Currumbin Waters QLD 4223
AUSTRALIA

Phone: +61 7 5598 7474
Fax: +61 7 5598 7070
EMail: mikem@open.com.au
URI: http://www.open.com.au.

Stig Venaas
cisco Systems
Tasman Drive
San Jose, CA 95134
USA

EMail: stig@cisco.com

Klaas Wierenga
Cisco Systems International BV
Haarlerbergweg 13-19
Amsterdam 1101 CH
The Netherlands

Phone: +31 (0)20 3571752
Fax:
EMail: kwiereng@cisco.com
URI: http://www.cisco.com.

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: July 4, 2011

C. LaJoie, Ed.
Itumi, LLC

December 31, 2010

Metadata Query Protocol
draft-lajoie-md-query-01

Abstract

This document defines a simple protocol for retrieving metadata about entities. The goal of the protocol is to profile various aspects of HTTP to allow requesters to rely on certain, rigorously defined, behaviour.

Status of this Memo

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

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

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

This Internet-Draft will expire on July 4, 2011.

Table of Contents

1. Introduction

Many clients of web-based services are capable of consuming descriptive metadata about a service in order to customize or information the client's connection parameters. While the form of the metadata (e.g.. JSON, XML) and content varies between services this document attempts to specifies a set of semantics for HTTP [RFC2616] that allow clients to rely on certain behavior. The defined behavior is meant to make it easy for clients to perform queries, to be efficient for both requesters and responders, and allow the responder to scale in various ways.

1.1. Notation and Convention

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 [RFC2119].

1.2. Terminology

entity - A single logical construct for which metadata may be asserted. Generally this is a network accessible service.

metadata - A machine readable description of certain entity characteristics. Generally metadata provides information such as end point references, service contact information, etc.

2. Protocol Transport

The metadata retrieval protocol seeks to fully employ the features of the HTTP protocol. Additionally this specification makes mandatory some optional HTTP features.

2.1. HTTP Version

Metadata retrieval protocol responders MUST use HTTP, version 1.1 [RFC2616]

2.2. HTTP Method

All metadata retrieval requests MUST use the GET method.

2.3. Request Headers

All metadata retrieval requests MUST include the following HTTP headers:

Accept - this header MUST contain the content-type identifying the type, or form, of metadata to be retrieved

All metadata retrieval requests SHOULD include the following HTTP headers:

Accept-Charset

Accept-Encoding

A metadata request to the same URL, after an initial request, MUST include the following header per section 13.3.4 of RFC2616 [RFC2616]:

If-None-Match

2.4. Response Headers

All successful metadata retrieval responses (even those that return no results) MUST include the following headers:

Content-Encoding - required if, and only if, content is compressed

Content-Type

ETag

All metadata retrieval responses SHOULD include the following headers:

Cache-Control

Content-Length

Last-Modified

2.5. Status Codes

This protocol uses the following HTTP status codes:

200 - standard response code when returning requested metadata

304 - response code indicating requested metadata has not been updated since the last request

400 - response code indicating that the requester's request was malformed in some fashion

401 - response code indicating the request must be authenticated before requesting metadata

404 - indicates that the requested metadata could not be found; this MUST NOT be used in order to indicate a general service error.

405 - response code indicating that a non-GET method was used

406 - response code indicating that metadata is not available in the request content-type

500 - standard response code when something goes wrong within the responder

501 - response code indicating that a given identifier transformation is not supported

505 - response code indicating that HTTP/1.1 was not used

2.6. Base URL

Requests defined in this document are performed by issuing an HTTP GET request to a particular URL. The final component of the path to which requests are issued is defined by the requests specified within this document. A base URL precedes such paths. Such a base URL MUST contain at least the scheme and host name components. It MAY also include a port as well as a path. It MUST NOT include URL fragments. If a path is included the path required by the particular defined request is appended to the path in the base URL.

2.7. Content Negotiation

As there may be many representations for a given piece of metadata, agent-driven content negotiation is used to ensure the proper representation is delivered to the requester. In addition to the required usage of the Accept header a response SHOULD also support the use of the Accept-Charset header.

3. Metadata Query Protocol

The metadata query protocol retrieves metadata based on one or more "tag" or "keyword" identifiers. A request may return information for none, one, or a collection of entities.

3.1. Identifiers

The query protocol uses identifiers to "tag" metadata for single- and multi-entity metadata collections. An identifier MAY contain any URL-encodable character but MUST NOT start with '{' (ASCII 0x7B) as this character has a special meaning in the first position (see below). The assignment of such identifiers to a particular metadata document is the responsibility of the query responder. If a metadata collection already contains a well known identifier it is RECOMMENDED that such a natural identifier is used when possible. Any given metadata collection MAY have more than one identifier associated with it.

3.1.1. Transforms

In some cases it may be advantageous to query for metadata using a transformed identifier. For example, some protocols will transmit hashed entity identifiers. This may be done to reduce the overall size of the identifier, escape special characters, obfuscate the identifier, etc.

A transformed identifier is represented by pre-pending the identifier with '{' + transformation indicator + '}'. The transformation indicator MUST be composed exclusively of printable ASCII characters (0x21-0x7E) excluding '{' (0x7B) and '}' (0x7D). Such an identifier need only make sense in the context within which it is used. Responders MUST support the MD5 (transformation indicator 'md5') and SHA1 (transformation indicator 'sha1') hashing algorithms as identifier transformations. The responder MAY support other transformation indicators.

For example, the identifier
http://example.org/service
transformed by means of MD5 hashing would become
{md5}f3678248a29ab8e8e5b1b00bee4060e0

3.2. Protocol

3.2.1. Request

A Metadata Query request is performed by issuing an HTTP GET request. All Metadata Query requests MUST use the URL format:

<base_url>/entities/{ID}+{ID}+...
The request MUST contain at least one identifier but MAY contain more than one. Each identifier must be properly URL encoded. If more than one identifier is used the returned metadata MUST have been labelled with each identifier. That is to say a search with multiple identifiers results in the intersection of the metadata that would be retrieved by searching for each identifier individually.

3.2.2. Response

The response to a Metadata Query request MUST be a document that provides metadata for the given request identifiers in the format described by the request's Content-Type header. Note, in the event that multiple identifiers were used in the request, it is the responder's responsibility to ensure that the metadata returned is valid. If the responder can not create a valid document it MUST respond with a 500 status code. An example of such an error would be the case where the result of the query is metadata for multiple entities but the request content type does not support returning multiple results in a single document.

3.2.3. Example Request and Response

```
GET /service/entities/http%3A%2F%2Fexample.org%2Fidp HTTP/1.1
Host: metadata.example.org
Accept: application/samlmetadata+xml
```

Example Metadata Query Request

```
HTTP/1.x 200 OK
Content-Type: application/samlmetadata+xml
ETag: abcdefg
Last-Modified: Thu, 15 Apr 2010 12:45:26 GMT
Content-Length: 1234

<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor entityID="http://example.org/idp" xmlns="urn:oasis:names:tc:SA
ML:2.0:metadata">
....
```

Example Metadata Query Response

4. Efficient Retrieval and Caching

4.1. Conditional Retrieval

Upon a successful response the responder is required to return an ETag header and may return a Last-Modified header as well. Requesters SHOULD user either or both, with the ETag being preferred, in any subsequent requests for the same resource. In the event that a resource has not changed since the previous request, the requester will receive a 304 (Not Modified) status code as a response.

4.2. Content Caching

Responders SHOULD include cache control information with successful (200 status code) responses, assuming the responder knows when retrieved metadata is meant to expire. The responder should also include cache control information with 404 Not Found responses. This allows the requester to create and maintain a negative-response cache. When cache controls are used only the 'max-age' directive SHOULD be used.

4.3. Content Compression

As should be apparent from the required request and response headers this protocol encourages the use of content compression. This is in recognition that some metadata documents can be quite large or fetched with relatively high frequency.

Requesters SHOULD support, and advertise support for, gzip compression unless such usage would put exceptional demands on constrained environments. Responders MUST support gzip compression. Requesters and responders MAY support other compression algorithms.

5. Security Considerations

5.1. Integrity

As metadata often contains information necessary for the secure operation of interacting services it is RECOMMENDED that some form of content integrity checking be performed. This may include the use of SSL/TLS at the transport layer, digital signatures present within the metadata document, or any other such mechanism.

5.2. Confidentiality

In many cases service metadata is public information and therefore confidentiality is not required. In the cases where such functionality is required, it is RECOMMENDED that both the requester and responder support SSL/TLS. Other mechanisms, such as XML encryption, MAY also be supported.

5.3. Authentication

All responders which require client authentication to view retrieved information MUST support the use of HTTP basic authentication over SSL/TLS. Responders SHOULD also support the use of X.509 client certificate authentication.

6. Normative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Appendix A. Acknowledgements

The editor would like to acknowledge the following individuals for their contributions to this document:

Scott Cantor (The Ohio State University)

Leif Johansson (SUNET)

Thomas Lenggenhager (SWITCH)

Ian Young (EDINA, University of Edinburgh)

Author's Address

Chad LaJoie (editor)
Itumi, LLC

Email: lajoie@itumi.biz