XEP-0388: Extensible SASL Profile

Dave Cridland
mailto:dave.cridland@surevine.com
xmpp:dave.cridland@surevine.com

2017-08-24
Version 0.2.1

Status	Type	Short Name
Experimental	Standards Track	sasl2

This document describes a replacement for the SASL profile documented in RFC 6120 which allows for greater extensibility.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

While SASL provides an excellent framework that has served us well over the past 18 years, a number of shortcomings in the profile - the syntax binding to XMPP - that is in use.
This specification addresses a number of shortfalls:

- Number of round trips
- Extensibility
- Support for second factor
- Support for mandatory password changes

The new SASL profile documented herein is primarily a syntactic change to allow extensibility, combined with removal of the (largely) redundant stream restart, and additional results beyond total success or abject failure.

1.1 Terminology

Although initiating entities, in general, use SASL, and receiving entities offer it, the SASL specification and common parlance both use ”Client ” and ”Server”; this specification uses Client and Server and assumes C2S links. This is not intended to preclude use of this SASL profile on S2S links. The term ”SASL2” is used to mean the new SASL profile specified in this document; however the same RFC 4422 definition of SASL (and SASL profiles) applies.
Examples often use hypothetical SASL mechanisms and sub-extensions; this specification does not intend to make a position on any particular SASL mechanism, and the Mandatory To Implement mechanisms are unaffected.

2 Overview

2.1 Discovering Support

Servers capable of SASL2 offer a stream feature of <mechanisms/>, qualified by the ”urn:xmpp:sasl:1” namespace. This in turn contains one or more <mechanism/> elements in the same namespace, and potentially other elements (for example, the <hostname/> element defined within XEP-0233).
Note that SASL2 is impossible for clients to initiate without at least one mechanism being available, and therefore MUST NOT be offered.
The feature so advertised, and its child content, SHOULD be stable for the given stream to and from attributes and encryption state, and therefore MAY be cached by clients for later connections.
The Service Name used by XMPP is unchanged from RFC 6120.

2.2 SASL Data Encoding

In all cases, both Clients and Servers encode SASL exchanges using Base 64 encoding. This SHOULD NOT include any line wrapping or other whitespace. As the form <element/> is equivalent to <element></element>, these both indicate an empty string. Challenges and responses with no data do not occur in SASL, and so require no special handling. To indicate the absence of an initial response, or the absence of success data, the element is simply not included.

2.3 Initiation

Clients, upon observing this stream feature, initiate the authentication by the use of the <authenticate/> top-level element, within the same namespace. The nature of this element is to inform the server about properties of the final stream state, as well as initiate authentication itself. To achieve the latter, it has a single mandatory attribute of "mechanism", with a string value of a mechanism name offered by the Server in the stream feature, and an optional child element of <initial-response/>, containing a base64-encoded SASL Initial Response.
On subsequent connections, if a Client has previously cache the stream feature, the Client MAY choose to send it before seeing the stream features - sending it "pipelined" with the Stream Open tag for example.

Listing 1: An authentication request

```
<authenticate xmlns='urn:xmpp:sasl:1' mechanism="BLURDLYBLOOP">
  <initial-response>Tm9ib2R5IGV2ZXIgZGVjb2RlcyB0aGUgZXhhbXBsZXMu</
     initial-response>
</authenticate>
```

In order to provide support for other desired stream states beyond authentication, additional child elements are used. For example, a hypothetical XEP-0198 session resumption element might be included, and/or Resource Binding requests.

Listing 2: An authentication request with a (hypothetical) bind request

```
<authenticate xmlns='urn:xmpp:sasl:1' mechanism='BLURDYBLOOP'>
  <initial-response>
    SSBzaG91bGQgbWFrZSB0aGlzIGEgY29tcGV0aXRpb24=
  </initial-response>
  <bind xmlns='urn:xmpp:bind:example'/>
</authenticate>
```

2.4 Challenges and Responses

Server Challenges MAY then be sent. Each Challenge MUST be responded to by a Client in a Client Response. These are not extensible, and contain the corresponding base64 encoded

SASL data:

Listing 3: A challenge and response exchange

```
<!-{}- A server might send: -{}->
<challenge xmlns='urn:xmpp:sasl:1'>
  U28sIG5leHQgRk9TREVNIC0gMjAxOCwgdGhhdCBpcy4uLg==
</challenge>

<!-{}- A client might respond: -{}->
<response xmlns='urn:xmpp:sasl:1'>
  Li4uSSdsbCBidXkgYSBiZWVyIGZvciB0aGUgZmlyc3QgcGVyc29uIHdoby4uLg==
</response>
```

2.5 During Authentication

At any time while authentication is in progress, neither Client nor Server sends any element (including stanzas) or other data except the top-level elements defined herein. Clients MUST NOT send whitespace, and MUST send only <response/> elements as appropriate or an <abort/> element to immediately cause an error. Servers MUST disconnect Clients immediately if any other traffic is received. Servers are similarly REQUIRED to send no whitespace, and only the <response/> and completion elements from the section below.

2.6 Completing Authentication

Authentication may complete in one of three ways. It may complete successfully, in which case the client is authenticated. It may also fail, in which case the client is not authenticated and the stream and session state remain entirely unchanged.
Finally, it may have completed successfully, but further interaction is required - for example, a password change or second-factor authentication.

2.6.1 Success

If the Client is now authenticated, the Server sends a <success/> element, which contains contains an <authorization-identity/> element containing the negotiated identity - this is a bare JID, unless resource binding has occurred, in which case it is a full JID.
It MAY contain an <additional-data> element, containing additional data from the exchange (task or SASL mechanism) that has just completed.

Listing 4: Successful authentication

```
<success xmlns='urn:xmpp:sasl:1'>
  <additional-data>
```

```
    ip/AeIOfZXKBV+fW2smE0GUB3I//nnrrLCYkt0Vj
  </additional-data>
  <authorization-identifier>juliet@montague.example/Balcony/
     a987dsh9a87sdh</authorization-identifier>
</success>
```

Other extension elements MAY also be contained by the <success/> element.

Listing 5: Successful re-authentication and resumption

```
<success xmlns='urn:xmpp:sasl:1'>
  <additional-data>
    SGFkIHlvdSBnb2luZywgdGhlcmUsIGRpZG4ndCBJPw==
  </additional-data>
  <authorization-identifier>juliet@montague.example/Balcony/
     a987dsh9a87sdh</authorization-identifier>
  <sm:resumed xmlns='urn:xmpp:sm:3:example' h='345' previd='124'/>
</success>
```

Any security layer negotiated SHALL take effect after the ">" octet of the closing tag (ie, immediately after "</success>"), if it has not already taken effect at a <continue> - see Continue below.
The <success> element is immediately followed by a <features> element containing the applicable stream features of the newly authenticated stream. Note that no stream restart occurs.

2.6.2 Failure

A <failure/> element is used by the server to terminate the authentication attempt. It MAY contain application-specific error codes, and MAY contain a textual error. It MUST contain one of the SASL error codes from RFC 6120 Section 6.5.

Listing 6: Failure

```
<failure xmlns='urn:xmpp:sasl:1'>
  <aborted xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
  <optional-application-specific xmlns='urn:something:else'/>
  <text>This is a terrible example.</text>
</failure>
```

2.6.3 Continue

A <continue/> element is used to indicate that while the SASL exchange was successful, it is insufficient to allow authentication at this time.
This can be used to indicate that the Client needs to perform a Second Factor Authentication

("2FA"), or is required to change password.
Such tasks are presented within a <tasks> element, which contains a sequence of <task> elements, each containing a name. These tasks are analogous to a SASL mechanism, but have a number of differences - they may never attempt to negotiate a new authorization identifier, nor a new security layer.
A client MAY choose any one of the offered tasks; if multiple are required a sequence of <continue> exchanges will occur until all mandatory tasks are complete.
The <continue element therefore always contains a <tasks/> element, as defined above. It MAY contain an <additional-data/> element, as the <success/> element does.
Finally, it MAY contain a <text/> element, which can contain human-readable data explaining the nature of the step required.

Listing 7: Continue Required

```
<continue xmlns='urn:xmpp:sasl:1'>
  <additional-data>
   SSdtIGJvcmVkIG5vdy4=
  </additional-data>
  <tasks>
    <task>HOTP-EXAMPLE</task>
    <task>TOTP-EXAMPLE</task>
  </tasks>
  <text>This account requires 2FA</text>
</continue>
```

After the final octet of the first <continue> element, any SASL security layer negotiated in the preceding exchange SHALL be immediately in effect.
Clients respond with a <next/> element, which has a single mandatory attribute of "task", containing the selected task name, and contains an OPTIONAL base64 encoded initial response contained in an <initial-response> element.

Listing 8: Client Continues

```
<next xmlns='urn:xmpp:sasl:1' task='TOTP-EXAMPLE'>
  <initial-response>
    SSd2ZSBydW4gb3V0IG9mIGlkZWFzIGhlcmUu
  </initial-response>
</next-authenticate>
```

3 SASL Profile Definition

This provides pointers and/or clarifications to the Overview in the order and manner defined in RFC 4422, section 4.

3.1 Service Name

The service name SHALL be ”xmpp”, as defined by RFC 6120.

3.2 Mechanism negotiation

Servers list mechanisms during stream features (See Discovering Support).

3.3 Message Definitions

3.3.1 Initiation

Clients initiate using the <authenticate/> top level element (See Initiation.

3.3.2 Server Challenges and Client Responses

See Challenges and Responses.

3.3.3 Outcome

See Completing Authentication.

3.4 Non-Empty Authorization Strings

If a Client specifies an authorization string which is non-empty, the identifier is normalized by treating it as a JID, and performing normalization as described in RFC 7622.
In general, implementors are advised that a non-empty authorization string MAY be considered an error if the stream’s from attribute (if present) does not match.

3.5 Aborting

Clients MAY abort unilaterally by sending <abort/> as specified in Client Aborts.
Servers MAY abort unliterally by sending <failure/> with the <aborted/> error code as defined in Failure.

3.6 Security Layer Effect

Security Layers take effect after the SASL mechanism itself (ie, the first negotiation) has completed successfully, after the final octet of the server's <success> or <continue>. See Success and Continue.

3.7 Security Layer Order

Option (a) is used - any SASL Security Layer is applied first to data being sent, and TLS applied last.

3.8 Multiple Authentication

Although the <continue/> concept does use tasks analogous to multiple SASL sequences, only the first SASL mechanism used is considered an authentication, and only the first can negotiate a security layer.
In particular, once <success/> has been sent by the server, any further <authenticate/> element MUST result in a stream error.

4 Example Flows

This section provides a fictional example. It is important to note that many of the extensions used wihtin this section do not, in fact, exist and therefore are to be avoided.

4.1 Syntactic Equivalence

Where no additional features that SASL2 makes available are used, the flow of information is identical to the original SASL profile. This example shows the new syntax and draws the reader's attention to the differences.

Listing 9: PLAIN Authentication

```
<!-{}-
  Server sends stream features.
  These are identical to SASL1, but within a different namespace.
-{}->
<stream:features>
  <mechanisms xmlns='urn:xmpp:sasl:1'>
    <mechanism>PLAIN</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
  </mechanisms>
</stream:features>
```

```
<!-{}-
  Client intiates authentication.
  Beyond the element local name and namespace,
  the main distinction is that initial-response data is held within an
      element,
  so the "=" special case no longer applies.
-{}->
<authenticate xmlns='urn:xmpp:sasl:1' mechanism='PLAIN'>
  <initial-response>AGFsaWNlQGV4YW1wbGUub3JnCjM0NQ==</initial-response
     >
</authenticate>

<!-{}-
  This completes in one step, so the Server sends a success.
  A SASL2 success always includes the authorization identifier:
-{}->
<success xmlns='urn:xmpp:sasl:1'>
  <authorization-identifier>alice@example.org</authorization-
     identifier>
</success>
<!-{}-
  The server immediately sends a new set of stream features at this
     point.
  There is no stream restart.
-{}->
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <required/>
  </bind>
</stream:features>

<!-{}-
  The client now continues with resource binding.
-{}->
```

Use of SASL2 in this simple scenario saves one round-trip (due to the lack of stream restart).

4.2 Once More, This Time With CRAM

Again, this is an equivalent flow to a common SASL1 flow, although using the CRAM-MD5 mechanism which is (thankfully) rarely used in practise.

Listing 10: CRAM-MD5 Authentication

```
<!-{}-
  Server sends stream features.
-{}->
```

```
<stream:features>
  <mechanisms xmlns='urn:xmpp:sasl:1'>
    <mechanism>PLAIN</mechanism>
    <mechanism>CRAM-MD5</mechanism>
  </mechanisms>
</stream:features>

<!-{}-
  Client intiates authentication.
  Here, no initial response is needed, so none is sent.
-{}->
<authenticate xmlns='urn:xmpp:sasl:1' mechanism='CRAM-MD5'/>

<!-{}-
  CRAM-MD5 starts with a server challenge:
-{}->
<challenge xmlns='urn:xmpp:sasl:1'>
  PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
</challenge>

<!-{}-
  And the client responds:
-{}->
<response xmlns='urn:xmpp:sasl:1>
  dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
</response>

<!-{}-
  This completes, so the Server sends a success.
-{}->
<success xmlns='urn:xmpp:sasl:1'>
  <authorization-identifier>tim@example.org</authorization-identifier>
</success>
<!-{}-
  The server immediately sends a new set of stream features at this
    point.
  There is no stream restart.
-{}->
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <required/>
  </bind>
</stream:features>

<!-{}-
  The client now continues with resource binding.
-{}->
```

Use of SASL2 in this simple scenario again simply saves one round-trip.

4.3 Advanced Usage

This moves into the deeply hypothetical. A binding extension is posited, alongside an unrealistic 2FA mechanism which somehow mutually authenticates because why not.

Listing 11: CRAM-MD5 Authentication

```
<!-{}-
  Server sends stream features.
  Note that whilst megabind is advertised, the unrealistic 2FA
     mechanism is not.
-{}->
<stream:features>
  <mechanisms xmlns='urn:xmpp:sasl:1'>
    <mechanism>BLURDYBLOOP</mechanism>
  </mechanisms>
  <megabind xmlns='urn:example:megabind'/>
</stream:features>

<!-{}-
  Client intiates authentication.
-{}->
<authenticate xmlns='urn:xmpp:sasl:1' mechanism='BLURDYBLOOP'>
  <initial-response>
    SW5pdGlhbCBSZXNwb25zZQ==
  </initial-response>
  <megabind xmlns='urn:example:megabind'>
    <resource>this-one-please</resource>
  </megabind>
</authenticate>

<!-{}-
  Maybe BLURDYBLOOP does this:
-{}->
<challenge xmlns='urn:xmpp:sasl:1'>
  PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
</challenge>

<!-{}-
  And the client responds:
-{}->
<response xmlns='urn:xmpp:sasl:1>
␣␣dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
</response>

<!-{}-
```

```
  This completes, so the Server sends a continue, with the
  success data (probably mutual auth) from BLURDYBLOOP.
  The next task required is the unrealistic 2FA.
  Note that the session is - probably - bound at this point, but
  the authorization-identifier is not passed to the client until
  the full authentication sequence is completed.
-{}->
<continue xmlns='urn:xmpp:sasl:1'>
  <additional-data>
    QWRkaXRpb25hbCBEYXRh
  </additional-data>
  <tasks>
    <task>UNREALISTIC-2FA</task>
  </tasks>
</continue>

<!-{}-
  The client supports the unrealistic 2FA, so can move onto the next
     task:
-{}->
<next xmlns='urn:xmpp:sasl:1' task='UNREALISTIC-2FA'>
  <initial-response>
    VW5yZWFsaXN0aWMgMkZBIElS
  </initial-response>
  <!-{}-
    In principle, another option could be negotiated here, but this
     seems
    unlikely to be useful in practise.
  -{}->
</next>

<!-{}-
  This 2FA process is both unrealistic and also uses multiple round-
     trips.
-{}->
<challenge xmlns='urn:xmpp:sasl:1'>
  PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
</challenge>

<!-{}-
  The client responds here.
-{}->
<response xmlns='urn:xmpp:sasl:1>
  dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
</response>

<!-{}-
  Finally, the server sends a success.
  Here, we have an authzid which is a full jid, since the session is
```

```
      now bound.
  The additional-data element here contains the mutual authentication
      data for the
  unrealistic 2FA, and doesn't␣refer␣back␣to␣the␣BLURDYBLOOP␣exchange.
-{}->
<success␣xmlns='urn:xmpp:sasl:1'>
␣␣<additional-data>
␣␣␣␣VW5yZWFsaXN0aWMgMkZBIG11dHVhbCBhdXRoIGRhdGE=
␣␣</additional-data>
␣␣<authorization-identifier>
␣␣␣␣alice@example.org/this-one-please
␣␣</authorization-identifier>
</success>
<!-{}-
␣␣The␣Server␣also␣sends␣an␣(empty)␣features␣element.
␣␣The␣client␣can␣simply␣continue␣and␣send␣stanzas.
-{}->
<stream:features/>
```

Although the unrealistic 2FA here uses 2 round-trips (real ones will probably use one), the embedding of resource binding as shown here means that a second RTT is saved by SASL2, and there's no net change. A more realistic example would see RTTs saved, and additional negotiations could be added to further reduce RTTs.

5 Security Considerations

Relative to the SASL profile documented in RFC 6120, this introduces more data unprotected by any security layer negotiated by SASL itself.
While no actual exchanges are introduced that are unprotected, the nature of this exchange might allow for (for example) a resource binding extension to be introduced.
SASL security layers are sparingly used in the field, however, so this is thought to be a theoretical, rather than practical, concern.

6 IANA Considerations

This XEP requires no interaction with the Internet Assigned Numbers Authority (IANA) [1].

[1] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

7 XMPP Registrar Considerations

None.

8 XML Schema

```
<?xml version='1.0' encoding='utf-8'?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'
           targetNamespace="urn:xmpp:sasl:1"
           xmlns="urn:xmpp:sasl:1"
           elementFormDefault="qualified">

    <xs:element name="mechanisms">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="mechanism" type="SaslMechName"
                    minOccurs="1" maxOccurs="unbounded"/>
                <xs:any/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="authenticate">
        <xs:complexType>
            <xs:attribute name="mechanism" type="SaslMechName"/>
            <xs:sequence>
                <xs:element name="initial-response"/>
                <xs:any/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="next">
        <xs:complexType>
            <xs:attribute name="task" type="SaslMechName"/>
            <xs:sequence>
                <xs:element name="initial-response"/>
                <xs:any/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="challenge" type="SaslData"/>

    <xs:element name="response" type="SaslData"/>

    <xs:element name="initial-response" type="SaslData"/>
```

```
    <xs:element name="success">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="additional-data" type="SaslData"/>
                <xs:element name="authorization-identifier" type="Jid"
                    />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="continue">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="additional-data" type="SaslData"/>
                <xs:element name="tasks">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="tasK" type="SaslMechName
                                "/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:simpleType name="Jid">
        <xs:restriction base="xs:string">
            <xs:maxLength value="3071"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="SaslMechName">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="20"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="SaslData">
        <xs:restriction base="xs:base64Binary"/>
    </xs:simpleType>
</xs:schema>
```

9 Acknowledgements

The author wishes to share any credit with many members of the community, including Lance Stout, Ralph Meijer, Phil Roberts and Florian Schmaus.