INTERNET-DRAFT
July, 2004
Expires: January, 2005
Intended category: Standards track

Magnus Nystrom
RSA Security
Alexey Melnikov
Isode Ltd.

# SASL in HTTP/1.1

draft-nystrom-http-sasl-12.txt

## Status of this Memo



## Abstract

This memo suggest the use of SASL [RFC2222] as a framework to enable the use of strong authentication mechanisms in HTTP/1.1 [RFC2616], and describes one approach to accomplish this.

Please send comments on this document directly to authors or to the relevant mailing lists, e.g. ietf-sasl@imc.org.

Table of contents

1 Conventions used in this memo ... ..... 3
2 Introduction ..... 4
2.1 The HTTP/1.1 challenge-response framework ..... X
3 Relationship with the HTTP/1.1 specification ..... X
4 SASL framework ..... 4
4.1 Introduction and examples ..... X
4.1.1 Introduction ..... X
4.1.2 Example sequence diagrams ..... X
4.2 SASL authentication scheme ..... X
4.2.1 Recognition of the scheme ..... X
4.2.2 SASL authentication response header sent by server ..... X
4.2.3 SASL authorization request header sent by client ..... X
4.3 Usage model ..... X
4.3.1 SASL handshake initiation ..... X
4.3.2 Client response ..... X
4.3.3 Server behavior upon receiving a "SASL" <auth-scheme> token ..... XX
4.3.4 Client behavior upon receiving a "SASL" <auth-scheme> token ..... XX
4.3.5 Subsequent requests ..... XX
4.3.6 Client aborting a handshake ..... XX
4.3.7 Pipelining considerations ..... XX
4.3.8 Caching considerations..... XX
4.3.9 "Web farm" considerations ..... XX
4.3.10 HTTP header and state management ..... XX
4.4 Request/response encoding ..... XX
4.4.1 SASL challenge/response encoding ..... XX
4.4.2 Security layer..... XX
4.4.3 Interaction with TLS..... XX
4.4.4 Mandatory to implement SASL mechanism..... XX
4.5 Status codes and error handling ..... XX
4.5.1 HTTP/1.1 status codes..... XX
4.5.2 Client error handling..... XX
4.6 Authorization identity ..... XX
4.7 Examples ..... XX
4.7.1 Example 1 - Server requires authentication ..... XX
4.7.2 Example 2 - Initial response ..... XX
4.7.3 Example 3 - One mechanism only ..... XX
4.7.4 Example 4 - Server sends additional data ..... XX
4.7.5 Example 5 - Abort ..... XX
4.7.6 Example 6 - Client requires authentication ..... XX
4.7.7 Example 7 - Client requires authentication, server supports multiple realm ..... XX
4.7.8 Example 8 - Client uses POST request ..... XX
4.7.9 Example 9 - Client authentication fails ..... XX
4.8 Interoperability with existing HTTP/1.1 clients and

servers ..... XX
4.9 Preferences ..... XX
4.10 SASL mechanism recommendations ..... XX
5 IANA considerations ..... XX
5.1 GSSAPI/SASL service name ..... XX
5.2 HTTP/1.1 Status codes ..... XX
6 Security considerations ..... XX
6.1 Introduction ..... XX
6.2 Active attacks ..... XX
6.2.1 Man-in-the-middle ..... XX
6.2.2 Denial of service ..... XX
6.2.3 Replay ..... XX
6.3 Passive attacks ..... XX
6.4 Protecting the body of POST/PUT requests ..... XX
6.5 Other considerations ..... XX
7 Implementation considerations (informative)..... XX
7.1 The SASL authentication exchange context ..... XX
7.2 SASL security layer handling ..... XX
7.3 SASL Profile Checklist ..... XX
8 Acknowledgements ..... XX
9 References ..... XX
9.1 Normative references ..... XX
9.2 Informative references ..... XX
10 Authors' addresses ..... XX
11 IPR Disclosure Acknowledgement ..... XX
12 Intellectual Property Statement ..... XX
13 Full Copyright Statement ..... XX
Appendix A. Changes since previous revisions ..... XX

## 1 Conventions used in this memo

The keywords "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as defined in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

General understanding of SASL [RFC2222] is required before reading of this document. [RFC2222] defines several terms used through out this document, in particular "authorization identity" and "security layer".

In the examples, "C:" and "S:" indicate lines sent by a client and a server respectively; "CP:" and "SP:" indicate lines sent by a client and a server respectively with a SASL security layer active.

## 2 Introduction

The Hypertext Transfer Protocol, HTTP/1.1 [RFC2616], supports only two authentication schemes, namely the "Basic Access Authentication Scheme" and the "Digest Access Authentication Scheme" [RFC2617]. Neither of these can be considered to be strong authentication schemes. The former is extremely insecure unless used in conjunction with a lower-level protocol offering security services, since it sends cleartext passwords. The latter is an improvement, but is still vulnerable to man-in-the-middle attacks.

The Simple Authentication and Security Layer (SASL [RFC2222]) provides a method for adding authentication and security services to connection-oriented protocols in a flexible manner, enabling a variety of authentication and security mechanisms (e.g. mechanisms based on one-time-passwords, public key technology or password-based public-key cryptography), and also a flexible means to negotiate these mechanisms subject to local policies and security requirements. This memo therefore suggests a method to use SASL in HTTP/1.1 and solicit comments on the suggested approach.

This document is using the HTTP/1.1 challenge-response framework to implement SASL in HTTP/1.1. The challenge-response framework is outlines in Section 2.1.

### 2.1 The HTTP/1.1 challenge-response framework

HTTP/1.1 provides a simple challenge-response mechanism that can be used by a server or proxy to challenge a client request and by a client to provide authentication information. The reader is referred to [RFC2616] and [RFC2617] for a more detailed description of this mechanism. The relevant ABNF productions are:

```
challenge = auth-scheme 1*SP 1#auth-param

auth-scheme = token

auth-param = token "=" (token | quoted-string)
```

The challenge will be found in a WWW-Authenticate or a Proxy-Authenticate header field.

The client response, containing the client's credentials is defined as follows:

```
credentials = auth-scheme 1*SP 1#auth-param
```

The response will be found in an Authorization or a Proxy-

Authorization header field.

# 3 Relationship with the HTTP/1.1 specification

This memo relies on the HTTP/1.1 [RFC2616] specification. As with RFC 2616, it uses the ABNF [RFC2234] grammar of that document and relies on both non-terminals and other aspects of it.

Further, this memo REQUIRES persistent connections whenever a SASL security layer (see Section 4.4.2) is negotiated. Note, that a SASL security layer is an optional (to negotiate) feature of SASL, however, once negotiated it can't be turned off (or not used), until a subsequent reauthentication completes successfully on the same TCP connection. It is also RECOMMENDED to use a persistent connection while performing a SASL authentication exchange. See also Section 4.3.10 for additional discussions of this issue.

# 4 SASL framework

## 4.1 Introduction and examples

### 4.1.1 Introduction

The SASL protocol itself is relatively straightforward. It consists of a number of exchanges between the client and the server. Typically, the initial exchange negotiates the authentication mechanism and then remaining exchanges actually authenticate the client to the server.

The following figure shows, in schematic fashion a typical SASL authentication handshake which authenticates the client using the CRAM-MD5 mechanism. See Section 4.7.1 for the detailed example on how this will look in HTTP.

```
Client                                          Server
------                                          ------
                        <- Please authenticate, I speak
                                  CRAM-MD5, GSSAPI, and
                                  DIGEST-MD5

I choose CRAM-MD5       ->

                        <- Go ahead, your challenge
                                        is "abcdef"

I am user "user8", and  ->
my response is
```

"0123456789ABCDEF"

<- Ok user "user8", your are
authenticated.

Note that other mechanisms may require a larger number of round trips.

This document describes how to use SASL as an authentication mechanism for HTTP. Standard HTTP authentication headers are used, but they contain SASL data. SASL messages sent by the client are carried in the Authorization header. SASL messages sent by the server are carried in the WWW-Authenticate header.

### 4.1.2 Example sequence diagrams

Server initiated authentication:

Client Server

---------- Initial Request ---------->

<------ 401 WWW-Authenticate SASL (mechanisms,realm,id) --

--- Authorization (mechanism,id[,realm]) ---------->

<------ 401 WWW-Authenticate SASL (id,challenge) ---------

--- Authorization (id,credential)---------->

[
<------ 401 WWW-Authenticate SASL (id,challenge) ---------

--- Authorization (id,credential)---------->

](0 or more times depending on the SASL mechanism)

<------ 235 WWW-Authenticate SASL (id) ----------

---------- Initial Request (retry) ---------->

<------ 200 Server performs the requested operation ------

Client initiated authentication:

Client Server

```
   --- OPTIONS request with Authorization ([realm]) -------->

   <------ 401 WWW-Authenticate SASL (mechanisms,realm,id) --

   --- Authorization (mechanism,id) ----------------------->

   <------ 401 WWW-Authenticate SASL (id,challenge) ---------

   --- Authorization (id,credential)----------------------->

   [
   <------ 401 WWW-Authenticate SASL (id,challenge) ---------

   --- Authorization (id,credential)----------------------->

   ](0 or more times depending on the SASL mechanism)

   <------ 235 WWW-Authenticate SASL (id) -------------------

   ----------------- Initial Request ---------------------->

   <------ 200 Server performs the requested operation ------

   <<All subsequent requests are carried out as usual.>>
```

### 4.2 SASL authentication scheme

#### 4.2.1 Recognition of the scheme

A server MUST use the auth-scheme token "SASL" if it supports SASL and is willing to perform authentication using a SASL-based mechanism.

#### 4.2.2 SASL authentication response header sent by server

For the "SASL" <auth-scheme>, the authentication response header is as follows:

```
     challenge       = SASL 1*SP sasl-response-parameters

     sasl-response-parameters
                     = [sasl-mechanisms WSAC] [realm WSAC] sasl-sid
                       [WSAC sasl-challenge] [WSAC sasl-status]
                       [WSAC http-authzid]

     sasl-mechanisms = "mechanisms" "=" <"> 1#sasl-mech-name <">
```

```
realm           = "realm" "=" quoted-string
   ; See RFC 2617

sasl-sid        = "id" "=" quoted-string

sasl-challenge  = "challenge" "=" <"> base64-string <">

sasl-status     = "status" "=" quoted-string

http-authzid   = "http-authzid" "=" sasl-authzid

sasl-authzid    = <"> URI <">
   ; Usually a URI using the "http" scheme.
   ; URI is defined in [RFC2396]

sasl-mech-name  = 1*20 SASLCHAR
   ; Name must be from IANA set of registered SASL mechanisms,
   ; e.g. "SECURID"

base64-string   = *base64-group [base64-fingroup]
   ; Encoding must be in accordance with Section 3 of [RFC3548],
   ; except not limited to 76 chars/line.
   ; Spaces are not allowed.

base64-group    = 4*BASE64

base64-fingroup = 4*BASE64 | (3*BASE64 "=") | (2*BASE64 "==")

SASLCHAR        = UPALPHA | DIGIT | "-" | "_"
   ; Characters allowed in SASL mechanism name

BASE64          = DIGIT | ALPHA | "+" | "/"

WSAC            = *LWS "," *LWS
```

Note: All directives ("mechanisms", "id", "realm", "challenge", etc.) are case-insensitive. All directive values are case-sensitive.

The meanings of the values of the directives used above are as follows:

sasl-mechanisms
: A list of registered SASL mechanisms acceptable to the server. MUST be sent by the server unless a mechanism already has been agreed upon (see example 2 in Section 4.7.2). A server should list supported SASL mechanisms in its preferred order - from the most preferred to the least preferred. However a client MUST NOT blindly trust the order of the mechanisms in the received

sasl-mechanisms directive. The client must enforce own mechanism selection policy first, e.g. "only use mechanisms that provide mutual authentication", and only use the order specified by the server if everything else is equal.

realm
As defined in [RFC2617]. The directive MUST be present in initial challenges and when the realm otherwise would not be known by the client.

sasl-sid
A session identifier identifying a particular SASL authentication exchange (handshake) context (see also Section 7.1). MUST always be present. Sasl-sids are chosen by the server and at any given point in time MUST be unique for each established connection.

sasl-challenge
A Base64-encoded challenge (or server credentials, at the end of an authentication exchange) in accordance with a selected SASL mechanism. MUST NOT be sent unless there is exactly one SASL mechanism in the <sasl-mechanisms> directive.

sasl-status
A string indicating the resulting status of a SASL authentication exchange. For this version of this profile, this parameter is only used when client authentication has failed, in which case the parameter's value shall be "failed" (see further Section 4.3.3).

http-authzid
Upon successful authentication the server MAY (and if the client specified options="http-authzid" the server MUST) return the resulting protocol-specific authorization identifier for the authenticated client. The returned identifier informs the client of the established HTTP/1.1 authorization identity.

#### 4.2.3 SASL authorization request header sent by client

For the SASL scheme, the authorization request header is as follows:

```
credentials      = SASL [1*SP sasl-request-parameters]

sasl-request-parameters
                 = [sasl-mechanism WSAC] [sasl-sid WSAC]
                   [realm WSAC] [sasl-options WSAC] [sasl-
credentials]

sasl-mechanism   = "mechanism" "=" <"> sasl-mech-name <">
```

```
sasl-credentials = "credentials" "="
                   <"> (base64-string <"> | cancel-token) <">

sasl-options     = "options" "=" <"> 1#token <">

cancel-token     = "*"
```

The meanings of the values of the directives used above are as follows:

sasl-mechanism
A SASL mechanism acceptable to the client, chosen from the list provided by the server or set by some configuration. MUST be sent by the client unless a mechanism already has been agreed upon.

sasl-sid
A session identifier identifying a particular SASL authentication exchange context, previously set by a server. MUST always be sent by
the client except for the case of "initial responses," see Section 4.3.1 below.

realm
As defined in [RFC2617]. MUST always be sent by the client unless the realm is possible to determine by other means.

sasl-credentials
Base64-encoded credentials in accordance with a selected SASL mechanism, or a <cancel-token> ("*"). MUST be sent if a <sasl-challenge> directive has been received by the client.

sasl-options
Allows the client to request SASL specific options. Currently only a single option "http-authzid" is defined. Sending of the "http-authzid" option instructs the server to return a <http-authzid> directive upon successful authentication (see also Section 4.2.2). The "http-authzid" option MUST only be sent in the request in which the client selects the authentication mechanism to be used. Other options may have other restrictions.

## 4.3 Usage model

### 4.3.1 SASL handshake initiation

#### 4.3.1.1 Server initiated authentication

When a client makes a request for a resource on a server that requires SASL-based authentication, the server MUST respond with a

401 - Unauthorized (407 - Proxy Authentication Required) response including a WWW-Authenticate (or Proxy-Authenticate) header field that contains a "SASL" <auth-scheme>.

The server MUST list all supported and acceptable SASL mechanisms in the <sasl-mechanisms> directive. If the server only supports one SASL mechanism, it MAY include a <sasl-challenge> directive in order to reduce the number of roundtrips (see the example in Section 4.7.3). The server MUST include a <sasl-sid> directive to identify the particular authenticaton exchange context. This value MUST be the same for all messages associated with that authentication exchange.

Further, the server MUST include a <realm> directive in accordance with [RFC2617], however if a particular SASL mechanism defines its own "realm" as a part of its authentication exchange, the mechanism specific version of "realm" MUST be used by the mechanism.

If the server supports multiple realms for the requested resource, it MUST return multiple SASL challenges formatted as described above, each including different <realm>s (and potentially different <sasl-sid>s for different realms).

The server MAY also return additional challenges if Basic and/or Digest [RFC2617] access authentication is supported for the requested resource.

4.3.1.2 Client initiated authentication

A client, which is about to issue a request to a server, and knows that the server requires a certain SASL mechanism, MAY include a a "SASL" <auth-scheme> token in an Authorization (or Proxy-Authorization) header field in its request. If the client chooses to do so, it MUST include a <sasl-mechanism> directive identifying the used SASL mechanism, but MUST NOT include a <sasl-sid> directive, as session identifiers are chosen by the server. The client MAY also specify a <realm> directive (if it is known) and a <sasl-options> directive in the request. If the chosen SASL mechanism requires that the client sends data first, the client MUST also include a <sasl-credentials> directive, c.f. the "initial response" in [RFC2222] (see also the example in Section 4.7.2). This minimizes the number of roundtrips, since otherwise the server would be required to send an empty challenge.

If the client requires authentication, but doesn't know which mechanisms are supported by the server, the client SHOULD issue an OPTIONS request that includes a Request-URI header for the desired resource and an Authorization (or Proxy-Authorization) header field containing a "SASL" <auth-scheme> token that MAY contain <realm>, but

MUST NOT contain any of the <sasl-mechanism>, <sasl-sid> or <sasl-credentials> directives. This provides a way for the client to query the server about supported SASL mechanisms for the requested resource.

This document REQUIRES that a compliant SASL-aware server handles an OPTIONS request with the "SASL" <auth-scheme> token described in the previous paragraph by listing all supported and acceptable SASL mechanisms in the <sasl-mechanisms> directive in the WWW-Authenticate (or Proxy-Authenticate) header field as described in Section 4.3.1.1. When replying to OPTIONS request the server SHALL use the 401 - Unauthorized (407 - Proxy Authentication Required) response, if the requested resource requires client authentication. <<Note that Section 10.4.2 of HTTP/1.1 requires that a 401 response includes a WWW-Authenticate header>> The server SHALL use the 200 - OK response, if unauthenticated users are allowed to see the resource. In both cases, the presense of the WWW-Authenticate (or Proxy-Authenticate) header field containing "SASL" <auth-scheme> signifies that SASL authentication is supported for the requested resource; the absence of any WWW-Authenticate (or Proxy-Authenticate) header field with "SASL" <auth-scheme> signifies that SASL authentication is not supported for the requested resource. For example, the server SHALL use the 200 - OK response including a WWW-Authenticate (or Proxy-Authenticate) header field with "SASL" <auth-scheme>, if unauthenticated users are allowed to see the resource and SASL authentication is supported for the resource. See also an example in Section 4.7.6.

<<Do we need a summary table here?>>

If the client has specified a wrong <realm> value (i.e. a <realm> value that is not recognized by the server or a <realm> value that doesn't control access to the requested resource) or has not provided any <realm> value and the server supports multiple realms for the requested resource, than the server MUST ignore data sent in the client's request and respond with a 401 - Unauthorized (407 - Proxy Authentication Required) response containing multiple SASL challenges formatted as described in section 4.3.1.1, each SASL challenge including different <realm>s. The client can than select a proper <realm> value and retry the authentication request. See also example in Section 4.7.7.

### 4.3.2 Client response

A client, which receives a "SASL" <auth-scheme> authentication response token containing the <sasl-mechanisms> directive in a WWW-Authenticate (Proxy-Authenticate) header in a 401 - Unauthorized (407 - Proxy Authentication Required) response, examines the list of the

available SASL mechanism found in the <sasl-mechanisms> directive. If the client can't find a supported and otherwise appropriate (for accessing the resource) SASL mechanism (see also note below), it MUST NOT continue the authentication exchange using a SASL mechanism not on the provided list. If no acceptable SASL mechanism is found, the client MAY try Digest and/or Basic authentication [RFC2617]. <<Should we mention other obvious possibilities like dropping connection?>> If the client has found an acceptable SASL mechanism, it constructs a new request as described below. This request MAY contain the headers from the original request, MUST contain an Authorization (Proxy-Authorization) header containing a "SASL" <auth-scheme> token, but SHOULD NOT contain the body of the original request (if any). We will reference any such request as a "SASL request". The purpose of SASL requests is to avoid sending the body of a request with each authentication step.

Note: In cases where the 401 - Unauthorized (407 - Proxy Authentication Required) response also contains a WWW-Authenticate (Proxy-Authenticate) header with a "Basic" and/or a "Digest" <auth-scheme> token, the selected authentication scheme will be subject to local client policy. Clients are RECOMMENDED never to select Basic authentication over any other server-suggested method.

The "SASL" <auth-scheme> token in the SASL request MUST include the <sasl-sid> value provided by the server and a <sasl-mechanism> directive with the chosen SASL mechanism name. If the chosen mechanism allows for "initial response" type messages, the client MUST also include the initial response in a <sasl-credentials> directive. If the client is transmitting an initial response of zero length, it MUST transmit the response as the empty token (i.e. credentials=""). This indicates that the response is present, but contains no data. The client MAY also include a <sasl-options> directive.

If the client is able and willing to negotiate a SASL security layer, it MUST establish an end-to-end tunnel using the CONNECT method as described in Section 5.3 of [RFC2817] before starting an authentication exchange. The Authorization header MUST NOT be used in a CONNECT request. However, in order to save round trips, a Proxy-Authorization header MAY be used in a CONNECT request.

Note: A direct connection (any intermediate proxies operating in tunnel mode) is required whenever a security layer is in effect, since at that point complete HTTP/1.1 messages may be encrypted.

When two or more authentication exchanges are performed in parallel on the same connection ("mixed"), the client MUST NOT negotiate a security layer on more than one of them. Multiple <sasl-sid>

directives SHOULD NOT be "mixed" on the same connection, except for the case when a client starts an authentication exchange with the target server and an intervening proxy server asks the client to authenticate to it first. In this case, the client must perform an authentication exchange to the proxy first and then resume authentication to the end server.

The following diagram demonstrates a "mixed" authentication exchange:

```
Client                                                Server

----------- Start Authentication Exchange 1 ------------->

<--------- Reply to Authentication Exchange 1 ------------

----------- Start Authentication Exchange 2 ------------->

<--------- Reply to Authentication Exchange 2 ------------

----------- Continue Authentication Exchange 1 ---------->
```

If the client receives a "SASL" <auth-scheme> authentication response token containing a <sasl-challenge> directive in a WWW-Authenticate (Proxy-Authenticate) header for a 401 - Unauthorized (407 - Proxy Authentication Required) response, the client should behave as described in Section 4.3.4.

#### 4.3.3 Server behavior upon receiving a "SASL" <auth-scheme> token

If the <auth-scheme> token contains a <sasl-sid> directive, then the server MUST check if the SASL authentication exchange context identified by <sasl-sid> is valid. If it is not, the server SHALL reply with a 401 - Unauthorized (407 - Proxy Authentication Required) response, that contains a new <sasl-sid> value and the session continues as described in Section 4.3.1.1, i.e. the server MUST list all supported and acceptable SASL mechanisms in the <sasl-mechanisms> directive.

If the <auth-scheme> token contains a <sasl-mechanism> directive, the server MUST check if it mechanism is acceptable. If it is not, the server MUST reply with a 450 - "Authentication mechanism not accepted" response and, if the request included a <sasl-sid> directive, delete the SASL authentication context identified by the <sasl-sid>.

If the <auth-scheme> token contains a <sasl-credentials> directive, the server MUST check if the supplied credentials authenticates the client. If the <sasl-credentials> directive contains a <cancel-

token> then the server MUST reject the exchange with a 401 - Unauthorized reply.

Otherwise, the server uses the value of <sasl-credentials> directive to check if the client is authenticated. If the client is not (yet) authenticated, the server uses the supplied credential value to calculate a new <sasl-challenge> as per the currently selected SASL mechanism. If the new <sasl-challenge> is successfully calculated, it is returned in the WWW-Authenticate (or Proxy-Authenticate) header of a new 401 - Unauthorized (407 - Proxy Authentication Required) response.

If the client authentication failed, the server SHALL reply with a 401 - Unauthorized (407 - Proxy Authentication Required) response containing a WWW-Authenticate (or Proxy-Authenticate) header containing a "SASL" <auth-scheme> authentication token with exactly two <auth-params> directives: <sasl-sid> and <sasl-status>. The value for the <sasl-status> directive shall be "failed". When receiving a message with a WWW-Authenticate (Proxy-Authenticate) header of this type, the client shall interpret the response in accordance with Section 10.4.2 of [RFC2616]. For an example of this, see Section 4.7.9.

Note: This method of conveying information about a failed authentication differs slightly from that defined in [RFC2616]. The reason for this discrepancy is twofold: There may be SASL methods for which two consecutive challenges are identical, and the method defined in 10.6.2 of [RFC2616] was not designed for multiple-step authentication exchanges.

Whenever an authentication exchange fails, both the client and the server MUST return to their previous authentication state, i.e. as if the authentication attempt never took place.

The server MAY also choose to reply with a 432 - Transition Needed response, which indicates that the user name is valid, but the entry in the authentication database needs to be updated in order to permit authentication with the specified SASL mechanism. A client, which receives a 432 - Transition Needed response, MAY retry authentication using the SASL PLAIN mechanism. This SHOULD NOT be done unless an appropriate TLS protection is in place. An interactive client MUST NOT perform PLAIN authentication automatically and MUST warn the user before proceeding.

If the client is authenticated, the server MUST at least include the <sasl-sid> directive with its "SASL" <auth-scheme> authentication response token. If the chosen SASL mechanism requires that further challenge/response data (i.e. "server returns success with additional

data" in [RFC2222]) be sent by the server, the server MUST respond with a 401 - Unauthorized (407 - Proxy Authentication Required) response containing a <sasl-challenge> directive with its "SASL" <auth-scheme> authentication response token in a WWW-Authenticate (or Proxy-Authenticate) header. Unless the server fails authentication, the client MUST reply to this with a new SASL request containing an Authorization header with a <sasl-sid> directive and an empty <sasl-credentials> directive. The server will reply to this with a 235 - Authentication Completed (236 - Proxy Authentication Completed) response and at this point authentication is complete, and a SASL security layer may take effect (see Section 4.4.2).

If the client is authenticated and the server does not need to send any further challenge information, the server replies with a 235 - Authentication Completed (236 - Proxy Authentication Completed) response.

In both cases, when the server replies with a 235 - Authentication Completed (236 - Proxy Authentication Completed) response, it MAY include an <http-authzid> directive in the "SASL" <auth-scheme> authentication response token. The <http-authzid> SHALL contain the authorization identity of the authenticated client in the form of a URI. Note that the content of a 235 - Authentication Completed (236 - Proxy Authentication Completed) response (and thus the <http-authzid> directive) is not protected by a SASL security layer. In some deployments, the value of the <http-authzid> directive may also contain confidential information which might require privacy protection.

Upon receipt of a 235/236 response the client shall consider authentication successful and may retry the original request (with the body of the request, if any), possibly protected by a negotiated security session (see Section 4.4.2).

#### 4.3.4 Client behavior upon receiving a "SASL" <auth-scheme> token

The client, upon receipt of a "SASL" <auth-scheme> authentication response token containing a <sasl-challenge> directive in a WWW-Authenticate (Proxy-Authenticate) header for a 401 - Unauthorized (407 - Proxy Authentication Required) response, calculates its credentials and responds with a new SASL request containing a (possibly empty, see previous section) <sasl-credentials> directive and a "SASL" <auth-scheme> token in an Authorization (Proxy-Authorization) header. The client repeats this until the authentication exchange is successful or the server responds with a 401 (407) message without the SASL <auth-scheme> token (see previous section).

#### 4.3.5 Subsequent requests

The same HTTP server (host identifier) may serve data governed by multiple realms that may have separate associated authentication databases. If the client leaves the authentication realm it is currently authenticated in, e.g. by issuing a request for a resource in a different realm, the server MAY force the client to re-authenticate in the new realm. In this case a new authentication exchange is started as described in 4.3.1. However there is a change in how the security layer is established (see Section 4.4.2). If a security layer is currently active and the new authentication exchange negotiates a new security layer, it MUST replace the existing one. This includes the case when the new security layer is the NULL layer, i.e. the connection reverts to a state where no SASL security layer is present). See Section 4.4.2 for a description of when the security layer is being replaced/dropped.

#### 4.3.6 Client aborting a handshake

A client may abort a handshake by letting the value of the <sasl-credentials> field consist of the <cancel-token>, "*". For an example of this, see Section 4.7.5.

#### 4.3.7 Pipelining considerations

When pipelining multiple authentication requests (or authentication requests together with other requests), the client MUST observe the rules established in Section 4.4.2. This means that an authentication request that completes a SASL authentication exchange and activates a SASL security layer, MUST be the last request in a group of requests. If this rule is not followed, the client will start sending cleartext data that may be interpreted by the server as encrypted. This can lead to a packet decode error on the server side and dropped connections.

When a SASL security layer has been negotiated clients MAY put multiple HTTP requests (and server may put multiple HTTP responses) inside a single SASL buffer of protected data. See also Section 4.4.2.

#### 4.3.8 Caching considerations

As described in [RFC2616] Section 14.8, a shared cache MUST NOT return a response to a request containing an Authorization header to any other requests unless special circumstances apply. To ensure that these circumstances do not apply here, the server MUST send a "Cache-Control: no-store" header together with the "WWW-Authenticate" header in all handshake responses.

#### 4.3.9 "Web farm" considerations

Implementation and configuration of the SASL negotiation mechanism described in this memo requires special considerations in the case of "web farm" environments where several servers may serve client requests since authentication state information otherwise may be lost. In particular, means for sharing of authentication negotiation state must be available.

#### 4.3.10 HTTP header and state management

There MUST NOT be more than one WWW-Authenticate or Proxy-Authenticate header field containing a SASL authentication response in any HTTP response. The WWW-Authenticate or Proxy-Authenticate header MUST NOT contain more than one SASL authentication response.

The only exception to these rules is when the server lists available SASL mechanisms and the access to the requested resource is controlled by more than one realm (see section 4.3.1).

There MUST NOT be more than one Authorization or Proxy-Authorization header field containing a SASL authorization request in any HTTP request.

Since support for persistent connections is optional in HTTP/1.1, all servers MUST implement some method for state management of SASL authentication exchanges. This may include (but is not limited to) session caching, session expiration, dealing with duplicated authentication requests.

This document does not specify methods for servers to manage session state once the client has been authenticated. For an example of such methods, see [RFC2965].

### 4.4 Request/response encoding

#### 4.4.1 SASL challenge/response Encoding

The <sasl-challenge> directive and the <sasl-credentials> directive contain SASL challenges and responses respectively. The challenges and responses MUST be base64 ([RFC3548], section 3) encoded before being placed in these fields. The base64 string may in general be arbitrarily long. Clients and servers MUST be able to support challenges and responses that are as long as are generated by the authentication mechanisms they support, independent of any line length limitations the client or server may have in other parts of its protocol implementation.

Note that, as described in Section 4.3.6, instead of containing a base64-encoded string, a <sasl-credentials> value may consist of the single "*" character, indicating to the server that the client aborts the handshake.

### 4.4.2 Security layer

If a protection mechanism is negotiated as part of the SASL security session, then it MUST be applied to all subsequent requests and responses sent between the server and the client for the given realm. Any negotiated security layer takes effect immediately following the <message-body> that concludes the authentication exchange for the client, and the <message-body> of 235 (236) response for the server. I.e., for later requests (and responses) all data - including the status line and headers - will be protected by the new security layer.

The same rules apply in a case of reauthentication. Whenever a new security layer (including the empty one) is negotiated due to reauthentication, the current layer gets replaced (dropped) immediately after transmission (receipt) of the 235 (236) response.

A client that requires a security layer MUST check, after successful authentication, that such a layer indeed was negotiated.

Note that a security layer requires HTTP/1.1 persistent connection.

### 4.4.3 Interaction with TLS

A client may not perform an HTTP/1.1 "Upgrade" to TLS [RFC2817] while conducting a SASL negotiation, but is free to do so after, or before, the SASL negotiation takes place.

This document allows for both a TLS and a SASL security layer to be active at the same time. No matter in which order they were negotiated, any data will be transformed by the SASL security layer first and then by TLS, i.e. the relevant protocol stack will be as follows:

#### 4.4.4 Mandatory to implement SASL mechanism

In order to guarantee interoperability, all client and server implementations conformant to this document MUST support the DIGEST-MD5 [RFC2831] SASL mechanism. Since support for persistent connections is optional in HTTP/1.1, this implies that all clients and servers MUST support DIGEST-MD5 in non-persistent mode.

### 4.5 Status codes and error handling

#### 4.5.1 HTTP/1.1 status codes

HTTP/1.1 status codes which apply to SASL-based mechanisms are:

-235 - Authentication Completed
This status code indicates that SASL authentication with the server is complete and the client may retry sending the original request.
-236 - Proxy Authentication Completed
This status code indicates that SASL authentication with the proxy is complete and the client may retry sending the original request.
-401 - Unauthorized
An HTTP/1.1 server will use this status code when credentials supplied by a client could not be validated, in addition to the use described in Section 4.3 above.
-407 - Proxy Authentication Required
An HTTP/1.1 proxy will use this status code when credentials supplied by a client could not be validated, in addition to the use described in Section 4.3 above.
-432 - Transition Needed
This status codes indicates that the user name is valid, but the entry in the authentication database needs to be updated in order to permit authentication with the specified SASL mechanism. This typically is done by authenticating once using the PLAIN authentication mechanism. See Section 4.3.4.

This status code can be sent, for example, if a user has an entry in a system authentication database such as Unix /etc/passwd, but does not have credentials suitable for use by the specified mechanism.
-450 - Authentication mechanism not accepted
An HTTP/1.1 server will use this status code when a client suggests an authentication mechanism which is not supported or accepted by the server.

#### 4.5.2 Client error handling

When a client does not support any of the security mechanisms suggested by a server, or is otherwise unable to complete a SASL mechanism handshake with a server, it shall close the connection.

(instead of closing the connection the client MAY also cancel the SASL exchange by specifying a "*" in a <sasl-credentials> directive as described in Section 4.3.6). User-oriented clients SHOULD provide the user with information about the failed handshake, and MUST fail in a controlled, predictable manner.

### 4.6 Authorization identity

This document defines an authorization identity in the HTTP profile of SASL to be a sequence of Unicode characters (excluding NUL), encoded in UTF-8. This sequence is further prepared using the "SASLPrep" profile [SASLPrep] of the "stringprep" algorithm [RFC3454]. The latter restriction is required in order to have a predictable result when comparing two authorization identities entered by two different individuals, potentially using different input mechanisms. This is also required as many SASL mechanisms use authorization identities to produce hash values.

Clients MUST use the algorithm described above on authorization identities entered by a user (for interactive clients) or read from a configuration file. Servers MUST verify that a received authorization identity is in the correct form. If the preparation of the authorization identity fails or results in an empty string, the server MUST fail the authentication exchange. The only exception to this rule is when the received authorization identity is already the empty string.

### 4.7 Examples

Note: In the examples, some lines are wrapped for readability reasons.

#### 4.7.1 Example 1 - Server requires authentication

This example illustrates a client requesting a URL and a server responding with a list of supported SASL mechanisms. The client selects one of these and responds with a new request containing an initial-response type <sasl-credentials> directive. The server then issues a <sasl-challenge> directive back to the client which once again responds with a <sasl-credentials> directive in the Authorization header field.

```
C: GET http://classified.example.com/classified.html HTTP/1.1
   Host: classified.example.com

S: HTTP/1.1 401 Unauthorized
   Cache-Control: no-store
   WWW-Authenticate: SASL
```

```
                mechanisms="DIGEST-MD5,GSSAPI,CRAM-MD5",
                realm="testrealm@example.com",
                id="jfkasdgru42705"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
                mechanism="CRAM-MD5",
                id="jfkasdgru42705"

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
                id="jfkasdgru42705",
                challenge="PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9u
                          Lm1jaS5uZXQ+"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
                id="jfkasdgru42705",
                credentials="dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNG
                           QzODkw"

   S: HTTP/1.1 235 OK
      Cache-Control: no-store
      WWW-Authenticate: SASL
                id="jfkasdgru42705"
```

Client now retries the original request:

```
   C: GET http://classified.example.com/classified.html HTTP/1.1
      Host: classified.example.com

   S: HTTP/1.1 200 OK
      Cache-Control: no-store
      ...Requested Document follows...
```

#### 4.7.2 Example 2 - Initial response

In this example a client knows in advance that a certain SASL mechanism is required. The mechanism allows for an initial-response type message and the client therefore includes a <sasl-credentials> directive in its Authorization header. The server accepts the

credentials and responds with the requested information.

```
  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              mechanism="SECURID",
              credentials="AG1hZ251cwAxMjM0NTY3OAA="
```

(the client doesn't know if authentication is complete at this point, as certain SASL mechanisms have a variable number of steps.)

```
  S: HTTP/1.1 235 OK
     Cache-Control: no-store
     WWW-Authenticate: SASL
              id="jfkasdgru42705"

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Host: classified.example.com

  S: HTTP/1.1 200 OK
     Cache-Control: no-store
     ...Requested Document follows...
```

#### 4.7.3 Example 3 - One mechanism only

In this example a server supports only one SASL mechanism, which allows for sending of an initial challenge to a client.

```
  C: GET http://classified.example.com/classified.html HTTP/1.1
     Host: classified.example.com

  S: HTTP/1.1 401 Unauthorized
     Cache-Control: no-store
     WWW-Authenticate: SASL
              mechanisms="CRAM-MD5",
              realm="testrealm@example.com",
              id="jfkasdgru42705",
              challenge="PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9u
                        Lm1jaS5uZXQ+"

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              id="jfkasdgru42705",
```

```
                credentials="dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNG
                          QzODkw"

   S: HTTP/1.1 235 OK
      Cache-Control: no-store
      WWW-Authenticate: SASL
               id="jfkasdgru42705"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Host: classified.example.com

   S: HTTP/1.1 200 OK
      Cache-Control: no-store
      ...Requested Document follows...
```

#### 4.7.4 Example 4 - Server sends additional data

This example demonstrates the use of an integrity/privacy layer. Note that the client is using the CONNECT method, as it is willing to negotiate integrity/privacy protection provided by the DIGEST-MD5 SASL mechanism.

In its third message, the client specifies options="http-authzid", which instructs the server to return an <http-authzid> directive upon successful authentication.

```
   C: GET http://classified.example.com/classified.html HTTP/1.1
      Host: classified.example.com

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
               mechanisms="DIGEST-MD5,GSSAPI,CRAM-MD5",
               realm="testrealm@example.com",
               id="0001"

   C: CONNECT classified.example.com:80 HTTP/1.1
      Host: classified.example.com

   S: HTTP/1.1 200 OK

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
               mechanism="DIGEST-MD5",
               id="0001", options="http-authzid"
```

```
 S: HTTP/1.1 401 Unauthorized
    Cache-Control: no-store
    WWW-Authenticate: SASL
             id="0001",
             challenge="cmVhbG09ImVsd29vZC5pbm5vc29mdC5jb20iLG5vbmNl
                       PSJPQTZNRzl0RVFHbTJoaCIscW9wPSJhdXRoIixhbGdv
                       cml0aG09bWQ1LXNlc3MsY2hhcnNldD11dGYtOA=="

 C: GET http://classified.example.com/classified.html HTTP/1.1
    Cache-Control: no-store
    Pragma: no-cache
    Host: classified.example.com
    Authorization: SASL
             id="0001",
             credentials="Y2hhcnNldD11dGYtOCx1c2VybmFtZT0iY2hyaXMiLH
                        JlYWxtPSJlbHdvb2QuaW5ub3NvZnQuY29tIixub25jZT
                        0iT0E2TUc5dEVRR20yaGgiLG5jPTAwMDAwMDAxLGNub
                        25jZT0iT0E2TUhYaDZWcVRyUmsiLGRpZ2VzdC11cmk9
                        ImltYXAvZWx3b29kLmlubm9zb2Z0LmNvbSIscmVzcG9
                        uc2U9ZDM4OGRhZDkwZDRiYmQ3NjBhMTUyMzIxZjIxND
                        NhZjcscW9wPWF1dGg="

 S: HTTP/1.1 401 Unauthorized
    Cache-Control: no-store
    WWW-Authenticate: SASL
             id="0001",
             challenge="cnNwYXV0aD00YjJiYjM3ZjA0OTEwNTA1Nzc3YzJmNjM
                       4YzkyMjcyNQ=="

 C: GET http://classified.example.com/classified.html HTTP/1.1
    Cache-Control: no-store
    Pragma: no-cache
    Host: classified.example.com
    Authorization: SASL
             id="0001"

 S: HTTP/1.1 235 OK
    Cache-Control: no-store
    WWW-Authenticate: SASL
             id="0001",
             http-authzid="http://example.com/testrealm/users/lisa"

CP: GET http://classified.example.com/classified.html HTTP/1.1
    Host: classified.example.com

SP: HTTP/1.1 200 OK
    ...Requested Document follows...
```

```
    CP: ...Any subsequent request for a data on the same server,
           unless the server requests reauthentication...
```

#### 4.7.5 Example 5 - Abort

The following example shows how a client can abort an authentication exchange.

```
  C: GET http://classified.example.com/classified.html HTTP/1.1
     Host: classified.example.com

  S: HTTP/1.1 401 Unauthorized
     Cache-Control: no-store
     WWW-Authenticate: SASL
              mechanisms="DIGEST-MD5,GSSAPI,CRAM-MD5",
              realm="testrealm@example.com",
              id="0001"

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              mechanism="DIGEST-MD5",
              id="0001"

  S: HTTP/1.1 401 Unauthorized
     Cache-Control: no-store
     WWW-Authenticate: SASL
              id="0001",
              challenge="cmVhbG09ImVsd29vZC5pbm5vc29mdC5jb20iLG5vbmNl
                        PSJPQTZNRzl0RVFHbTJoaCIscW9wPSJhdXRoIixhbGdv
                        cml0aG09bWQ1LXNlc3MsY2hhcnNldD11dGYtOA=="

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              id="0001",
              credentials="*"

  S: HTTP/1.1 401 Authentication Canceled
     ...
```

#### 4.7.6 Example 6 - Client requires authentication

The following example is almost identical to Example 1, but here the

client requires authentication to the server.

```
  C: OPTIONS http://classified.example.com/classified.html HTTP/1.1
     Authorization: SASL
     Host: classified.example.com

  S: HTTP/1.1 401 Unauthorized
     Cache-Control: no-store
     WWW-Authenticate: SASL
              mechanism="DIGEST-MD5,GSSAPI,CRAM-MD5",
              realm="testrealm@example.com",
              id="jfkasdgru42705"

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              mechanism="CRAM-MD5",
              id="jfkasdgru42705"

  S: HTTP/1.1 401 Unauthorized
     Cache-Control: no-store
     WWW-Authenticate: SASL
              id="jfkasdgru42705",
              challenge="PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9u
                        Lm1jaS5uZXQ+"

  C: GET http://classified.example.com/classified.html HTTP/1.1
     Cache-Control: no-store
     Pragma: no-cache
     Host: classified.example.com
     Authorization: SASL
              id="jfkasdgru42705",
              credentials="dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNG
                          QzODkw"

  S: HTTP/1.1 235 OK
     Cache-Control: no-store
     WWW-Authenticate: SASL
              id="jfkasdgru42705"
```

Upon receipt of a 235 response the client submits the request it originally intended to submit:

```
  C: GET http://classified.example.com/classified.html HTTP/1.1
     Host: classified.example.com
```

```
S: HTTP/1.1 200 OK
   Cache-Control: no-store
   ...Requested Document follows...
```

#### 4.7.7 Example 7 - Client requires authentication, server supports multiple realm

The following example is almost identical to Example 2, but here the server supports multiple realms.

```
C: GET http://classified.example.com/classified.html HTTP/1.1
   Cache-Control: no-store
   Pragma: no-cache
   Host: classified.example.com
   Authorization: SASL
            mechanism="SECURID",
            credentials="AG1hZ251cwAxMjM0NTY3OAA="
```

As the server supports multiple realms for the requested resource, it forces the client to select the proper realm

```
S: HTTP/1.1 401 Unauthorized
   Cache-Control: no-store
   WWW-Authenticate: SASL
            mechanisms="DIGEST-MD5,SECURID",
            realm="testrealm@sales.example.com",
            id="jfkasdgru42705"
   WWW-Authenticate: SASL
            mechanisms="SECURID",
            realm="testrealm@example.com",
            id="jfkasdgru42705"
```

(Note that the server may choose to return multiple SASL challenges in a single WWW-Authenticate response, in this case the last server response may also look like:

```
S: HTTP/1.1 401 Unauthorized
   Cache-Control: no-store
   WWW-Authenticate: SASL
            mechanisms="DIGEST-MD5,SECURID",
            realm="testrealm@sales.example.com",
            id="jfkasdgru42705", SASL
            mechanisms="SECURID",
            realm="testrealm@example.com",
            id="jfkasdgru42705"
```

Also note, that different SASL challenges may use the same or different "id".)

```
   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
               mechanism="SECURID", id="jfkasdgru42705",
               realm="testrealm@sales.example.com",
               credentials="AG1hZ251cwAxMjM0NTY3OAA="

   S: HTTP/1.1 235 OK
      Cache-Control: no-store
      WWW-Authenticate: SASL
               id="jfkasdgru42705"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Host: classified.example.com

   S: HTTP/1.1 200 OK
      Cache-Control: no-store
      ...Requested Document follows...
```

#### 4.7.8 Example 8 - Client uses POST request

In this example the client is willing to perform a POST request but the server requires authentication and the establishment of a security layer.

Note that since the client sends its information unprotected in the initial POST message, in effect only the server's response (and any later messages) will benefit from this security layer.

```
   C: POST http://classified.example.com/update_classified.php
HTTP/1.1
      Host: classified.example.com
      Content-Type: ...
      Content-Length: ...

      ...request body...

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
               mechanisms="DIGEST-MD5,GSSAPI,OTP",
               realm="testrealm@example.com",
               id="0001"

   C: CONNECT classified.example.com:80 HTTP/1.1
      Host: classified.example.com
```

```
     S: HTTP/1.1 200 OK

     C: POST http://classified.example.com/update_classified.php
   HTTP/1.1
        Cache-Control: no-store
        Pragma: no-cache
        Host: classified.example.com
        Authorization: SASL
                 mechanism="OTP",id="0001",credentials="AHRpbQ=="

     S: HTTP/1.1 401 Unauthorized
        Cache-Control: no-store
        WWW-Authenticate: SASL
                 id="0001",challenge="b3RwLW1kNSAxMjMga2UxMjM0IGV4dA=="

     C: POST http://classified.example.com/update_classified.php
   HTTP/1.1
        Cache-Control: no-store
        Pragma: no-cache
        Host: classified.example.com
        Authorization: SASL
                 id="0001",credentials="aGV4OjExZDRjMTQ3ZTIyN2MxZjE="

     S: HTTP/1.1 235 OK
        Cache-Control: no-store
        WWW-Authenticate: SASL id="0001"

    CP: POST http://classified.example.com/update_classified.php
   HTTP/1.1
        Host: classified.example.com
        Content-Type: ...
        Content-Length: ...

        ...request body...

    SP: HTTP/1.1 200 OK
        ...Response to POST, if any...

    CP: ...Any subsequent request for a data on the same server,
           unless the server requests reauthentication...
```

#### 4.7.9 Example 9 - Client authentication fails.

In this example the client authentication fails and the server indicates this in its final message using the <sasl-status> directive.

```
     C: GET http://classified.example.com/classified.html HTTP/1.1
```

```
      Host: classified.example.com

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
               mechanisms="DIGEST-MD5,GSSAPI,CRAM-MD5",
               realm="testrealm@example.com",
               id="jfkasdgru42705"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
               mechanism="CRAM-MD5",
               id="jfkasdgru42705"

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
               id="jfkasdgru42705",
               challenge="PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9u
                         Lm1jaS5uZXQ+"

   C: GET http://classified.example.com/classified.html HTTP/1.1
      Cache-Control: no-store
      Pragma: no-cache
      Host: classified.example.com
      Authorization: SASL
               id="jfkasdgru42705",
               credentials="dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNG
                           QzODkw"

   S: HTTP/1.1 401 Unauthorized
      Cache-Control: no-store
      WWW-Authenticate: SASL
               id="jfkasdgru42705"
               status="failed"
```

At this point, both the server and the client shall return to their initial state wrt. the SASL authentication.

### 4.8 Interoperability with existing HTTP/1.1 clients and servers

A client supporting a certain SASL-based authentication mechanism allowing for initial responses MUST NOT include a <sasl-credentials> directive in a "SASL" <auth-scheme> authorization request in an Authorization or Proxy-Authorization header unless it knows that the

server supports the SASL mechanism in question. The client SHOULD use an OPTIONS request to discover the server's SASL capabilities (see Section 4.3.1.2 for more details).

A server supporting SASL-based authentication SHOULD include a "Basic" and a "Digest Access" <auth-scheme> token in a WWW-Authenticate or Proxy-Authenticate header field, if these authentication methods are acceptable to the server. This ensures proper interworking with clients only capable of performing a "Basic" or "Digest Access" authentication. Since these authentication mechanisms does not offer strong security, the risk of downgrading attacks should be carefully considered (see also the "Security Considerations" section in this memo and Section 4.1 and 4.2 in [RFC2617]).

### 4.9 Preferences

Servers MUST list authentication mechanisms in the WWW-Authenticate (Proxy-Authenticate) header field in preferred order.

### 4.10 SASL mechanism recommendations

It is RECOMMENDED that an SASL mechanism that supports the negotiation of a security layer with integrity protection be used, and that this protection be enabled to avoid the connection being hijacked after authentication has taken place. [RFC2222] discusses some of the security issues related to SASL mechanisms.

## 5 IANA considerations

### 5.1 GSSAPI/SASL service name

For use with SASL [RC2222], a protocol must specify a service name to be used with various SASL mechanisms, such as GSSAPI. For HTTP, the service name shall be "http".

### 5.2 HTTP/1.1 Status codes

This memo defines the following HTTP/1.1 status codes:

-235 "Authentication Completed"
-236 "Proxy Authentication Completed"
-432 "Transition Needed"
-450 "Authentication mechanism not accepted"

# 6 Security considerations

## 6.1 Introduction

This memo describes a method to integrate the SASL framework in HTTP/1.1. SASL as such allows a wide variety of mechanism, each with their own security characteristics. The following sections represent an attempt to discuss threats that can be regarded to be generic in the sense that they apply to the integration itself rather than specific SASL mechanisms. Security services offered by, and security considerations applying to, particular SASL mechanisms can be found through the IANA SASL mechanism registry.

## 6.2 Active attacks

### 6.2.1 Man-in-the-middle

Users of SASL in HTTP/1.1 SHOULD recognize that certain man-in-the-middle attacks are possible since the negotiation of the particular SASL security mechanism to be used is not necessarily protected. For example, if the server suggests SASL mechanisms A, B and C in a "SASL" <auth-scheme> token where A is a "strong" mechanism (for some definition of "strong") but B and C are "weak" or provide fewer security attributes than A, then an attacker could simply remove A from the list. This forces the client to choose a "weaker" mechanism and neither side will necessarily detect the changes made by the attacker.

To mitigate these attacks, servers SHOULD only suggest SASL mechanisms that will provide adequate security for the task at hand.

Similarly, the SASL <auth-scheme> token may be removed from the WWW-Authenticate (Proxy-Authenticate) header, thus forcing use of either the Basic or Digest Access method. For this reason, and unless other precautions (such as only accepting certain SASL mechanisms) are taken, it is RECOMMENDED that this authentication mechanism be used only in conjunction with a transport, e.g. TLS, providing protection against these attacks (server authentication and integrity protection of messages). Note however that when using client authentication mechanisms within a server-authenticated TLS tunnel, care must be taken to avoid the attack described in [MITM].

### 6.2.2 Denial of service

Since HTTP/1.1 requests and responses are not protected against modification per se, an attacker may, by removing SASL elements from HTTP/1.1 headers hinder a client from accessing a certain service. This is however a generic threat and not specific to the mechanism

described herein.

#### 6.2.3 Replay

Use of the "Cache-Control: no-store" and "Pragma: no-cache" headers when indicated in requests and responses ensures that proxies do not inadvertently store and/or deliver SASL handshake messages that otherwise could be used in replay attacks.

### 6.3 Passive attacks

Unless a transport security providing confidentiality is employed, the method described in this memo is susceptible to passive attacks where an attacker wants to find out about the mechanisms that are supported by a particular client.

### 6.4 Protecting the body of POST/PUT requests

When the client performs a POST/PUT request in the clear and gets Unauthorized response back from the server it is already too late to protect the body of the POST/PUT request, as it was already sent in the clear. Arguably, if the client sent some data in the clear with the user's permission, the user doesn't find the information being sent worth protecting. However, existing web clients are able to warn users about sending data in the clear, but don't have an option to establish a secure connection first.

The described problem is not specific to this document. HTTP over TLS uses a different URL schema to notify the client that it has to establish a secure connection first with TLS.

So, one way to mitigate the problem would be to define a new URL schema (or an extension to the existing URL schema) for SASL in HTTP. This is however outside of the scope for this document.

A client wishing to protect body of a POST/PUT request from modification and/or disclosure should first establish a channel protection using TLS and/or SASL. In general, an interactive client SHOULD ask a user (or be configurable) to establish channel protection before performing any POST/PUT.

### 6.5 Other considerations

Section 8.2 of [RFC2817] contains relevant security considerations for the CONNECT method.

Note that SASL mechanisms offering confidentiality and integrity protection of messages are only usable in conjunction with the

CONNECT method as described, since a proxy otherwise would be unable to handle the messages properly.

Section 6.3 ("Multiple authentications") of [RFC2222bis] contains security considerations regarding replacing a SASL security layer with no layer on reauthentication.

# 7 Implementation considerations

This section is informative.

## 7.1 The SASL authentication exchange context

This memo assumes the existence of a SASL authentication exchange context during the lifetime of a SASL handshake. The SASL authentication exchange context is a SASL structure that represents all SASL state associated with the authentication exchange identified by sasl-sid. It may include (but is not limited to): the current step in a multiple-step authentication exchange, an authentication id, any material derived from password, private key, etc.

The context should be kept for some period of time after the connection goes away. This period is implementation defined. The SASL context should be deleted once the session expires, and must be deleted once the authentication exchange completes with success or failure, or the session otherwise becomes invalid (e.g. when a duplicated authentication exchange was received for the same session).

Although, a particular implementation may choose to store any SASL security layer state (e.g. encryption/decryption keys) as a part of the SASL context, this document considers a SASL security layer state to be a separate entity from the corresponding SASL context. The SASL security layer state is deleted when the connection it is protecting is closed or the corresponding authentication exchange fails. In the latter case we are talking about partially created SASL security layer states. However, as opposed to the SASL context, the SASL security layer state is not deleted when the authentication exchange completes successfully.

## 7.2 SASL security layer handling

This section attempts to summarize client and server behaviour with regards to SASL security layer negotiation.

A client willing to negotiate a SASL security layer must perform all of the following steps:

a) Use persistent connection to perform a SASL authentication exchange (Section 4.4.2). A SASL security layer (if supported by the server and negotiated) can only be used on the TCP connection that was used for the final "round" (i.e. C->S: client response, S->C: server confirms that authentication was successful) of the authentication exchange. Note that some SASL mechanisms use IP addresses in authentication exchange, which effectively requires the use of a persistent connection during the whole authentication exchange.

b) Use CONNECT to establish an end to end tunnel through proxies, unless the client has a prior knowledge that it talks directly to the target server (Section 4.3.2).

c) Notify the SASL layer/library being used that it supports channel integrity and/or confidentiality.

As the SASL security layer is an optional feature of SASL, the rules a)-c) do not guarantee that a security layer will be negotiated. A client that requires a security layer must check, after successful authentication, that such a layer indeed was negotiated.

Regarding c) above, if a client is not able and/or not willing to negotiate a SASL security layer it must notify the SASL layer/library being used that it doesn't support channel integrity or confidentiality. Failure to do so may result in a situation when both parties negotiate a SASL security layer, but the client is unable to use it. The client doesn't have to do step b) and may not do step a).

Similarly, a server willing to negotiate a SASL security layer must perform all of the following steps:

a) Use a persistent connection to perform a SASL authentication exchange (Section 4.4.2). A SASL security layer (if supported by the client and negotiated) can only be used on the TCP connection that was used for the final "round" of the authentication exchahge.

b) Support the CONNECT method (Section 4.3.2).

c) Notify the SASL layer/library being used that it supports channel integrity and/or confidentiality.

As for clients above, rules a)-c) do not guarantee that a security layer will be negotiated. A server, which requires a security layer, must check, after successful authentication, that such a layer indeed was negotiated.

If a server is not able and/or not willing to negotiate a SASL security layer it must notify the SASL layer/library being used that it doesn't support channel integrity or confidentiality. Failure to do so may result in a situation when both ends negotiate a SASL security layer, but the server is unable to use it.

### 7.3 SASL Profile Checklist

The profiling requirements of [SASL] require that the following information be supplied by a protocol definition:

service name: "http" (section 5.1)

authentication protocol exchange initiation: section 4.3.1

listing supported SASL mechanisms:
 a) if server requires authentication: section 4.3.1.1
 b) client request the list: section 4.3.1.2

Initial client response: sections 4.3.1.2, 4.3.2

Initial server challenge: section 4.3.1.1

exchange sequence: client -> server: section 4.3.3
server -> client : section 4.3.2, 4.3.4
server sends failure: sections 4.3.3, 4.4.1
server sends success: section 4.3.3

client aborts exchange: section 4.3.6, also sections 4.4.1, 4.2.3

optional data with success: not supported, see section 4.3.3

security layer negotiation: section 4.4.2

order of SASL security layer and TLS,
if both are negotiated: section 4.4.3

use of the authorization identity: section 4.6

multiple authentications: yes, see section 4.3.5, also section 4.4.2

Interaction of SASL exchange with line lenght limits: section 4.4.1

Specific Issues:
 multiple realms: sections 4.3.1.1 and 4.3.1.2
 persistent connection: sections 3 and 4.3.10
 mixing multiple authentications on the same connection: section 4.3.2
 OPTIONS method: 4.3.1.2

## 8 Acknowledgements

Text for Section 4.6 was borrowed from [RFC2829]. Thanks to Keith Burdis, Raif S. Naffah, Mark Nottingham, Joe Orton, John P. Speno, Lisa Dusseault and Eric Rescorla for providing useful feedback and suggestions.

Robert Zuccherato, Entrust Inc., made significant contributions to earlier drafts of this work.

A large part of this document was written while Alexey was working for MessagingDirect.

## 9 References

### 9.1 Normative references

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3," IETF RFC 2026, October 1996.

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

[RFC2222] Myers, J., "Simple Authentication and Security Layer," IETF RFC 2222, October 1997.

[RFC2234] Crocker, D., Overell, P., "Augmented BNF for Syntax Specifications: ABNF," IETF RFC 2234, November 1997.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," IETF RFC 2396, August 1998.

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

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

[RFC2817] Khare, R., Lawrence, S., "Upgrading to TLS Within HTTP/1.1," IETF RFC 2817, May 2000.

[RFC2831] Leach, P. C. Newman, "Using Digest Authentication as a SASL Mechanism," IETF RFC 2831, May 2000.

[RFC3454] P. Hoffman, M. Blanchet, "Preparation of Internationalized

Strings ("stringprep")," IETF RFC 3454, December 2002.

[RFC3548] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," IETF RFC 3548, July 2003

[SASLPrep] Zeilenga, K., "SASLprep: Stringprep profile for user names and passwords," Work in progress, draft-ietf-sasl-saslprep-XX.txt.

### 9.2 Informative references

[MITM] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunnelled Authentication Protocols." Available from http://eprint.iacr.org/2002/163/

[RFC2222bis] Melnikov, A., "Simple Authentication and Security Layer (SASL)", Work in progress, draft-ietf-sasl-rfc2222bis-XX.txt.

[RFC2246] Dierks, T., and C. Allen, "The TLS Protocol Version 1.0," IETF RFC 2246, January 1999.

[RFC2829] Wahl, M., Alvestrand, H., Hodges, J., and R. Morgan, "Authentication Methods for LDAP," IETF RFC 2829, May 2000.

[RFC2965] Kristol, D., L. Montulli, "HTTP State Management Mechanism," IETF RFC 2965, October 2000.

## 10 Authors' addresses

Magnus Nystrom Email: magnus@rsasecurity.com
RSA Security
Box 10704
121 29 Stockholm
Sweden

Alexey Melnikov Email: Alexey.Melnikov@isode.com
Isode Limited
5 Castle Business Village,
36 Station Road,
Hampton, Middlesex,
United Kingdom, TW12 2BX

## 11 IPR Disclosure Acknowledgement

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

## 12 Intellectual Property Statement



## 13 Full Copyright Statement



Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

## Appendix A. Changes since previous revisions

Changes since -11

Editorial changes: Made "Conventioned used in this document" the first section. Moved 4.1.1 into new section 1. Moved 4.3.7 at the beginning of Section 4 (now section 4.1.2).

Added clarification note that a SASL security layer is an optional to negotiate feature of SASL.

Added Introduction text as suggested by Eric Rescorla.

Clarified significance and handling of the order of SASL mechanisms in the sasl-mechanisms directive as per comment by Eric.

Clarified server behaviour when receiving and OPTIONS request as per comments by Lisa (Sections 4.3.1.2 and 4.8.1)

Clarified the meaning of "a client ... MUST choose one of the available mechanisms" in Section 4.3.2. (List/Eric)

Other minor editorial changes as suggested by Lisa/Eric.

Updated Copyright/IPR as per new IETF policy.

Changes since -10

Added text on client prioritization when receiving both a "SASL" auth-scheme and a "Basic" or "Digest" auth-scheme in a 401 or 407 response.

Replaced "SASL block" with "SASL buffer of protected data". The latter is defined in RFC 2222.

Other editorial changes based on feedback by Lisa Dusseault.

Changed ABNF and updated examples in order to allow for an empty challenge/response.

Added http-authzid directive as suggested by Lisa Dusseault. Added sasl-options directive.

Changes since -09

Added empty initial credentials

New method for specifying failed authentication, including an

example.

Rewording of 4.3.8.

Added mandatory to support SASL mechanism.

Added explanatory text for multiple SASL challenges and for client abort of handshakes.

Added reference to [RFC2222bis] and [MITM].

Added an example when server supports multiple realms.

Added "SASL Profile Checklist" section.

Editorial clarifications and corrections.

Changes since -08

Editorial clarifications and corrections.

Changes since -07

Added "Implementation consideration" section with big discussion on how to correctly implement a SASL security layer. (Comment by Keith Burdis)

Moved the biggest part of "SASL Context" definition to the "Implementation consideration".

Added text describing that SASLPrep should be used on authorization identities.

Added section describing ways to protect/help protect body of a POST/PUT request. (Comment by Keith Burdis)

Several minor fixes.

Changes since -06

Changed 102 status code back to 401.

"credentials" directive is no longer returned by the server, only "challenge" is used.

Added text about SASL context.

Split "SASL handshake initiation" section into Client and Server

initiated.

Added text about performing multiple authentications in parallel.

Clarified the use of persistent connection with SASL. Added warnings about session caching and expiration. Updated text to tell when SASL context is destroyed.

Added new status codes: 450 "Authentication mechanism not accepted".

Expired session is denoted by a 401 (407) response with a new <sasl-sid> value.

Clarified when security layer is replaced/dropped on reauthentication.

Added warning that the server is required to keep track of authenticated clients. Removed the text that was saying that the server must return sasl-sid in 200 responses when authentication is complete.

Updated examples as a result of the changes mentioned above.

Other minor clarifications.

Changes since -05

Replaced "Cache-Control: no-cache" with "Cache-Control: no-store" as per Mark Nottingham comment.

ABNF corrections from Joe Orton and John P Speno.

More corrections from Joe Orton.

Changed 401 to a new status code 102 used solely for authentication.

Added Transition Needed status code (432). Should check if this code conflicts with anything.

Added new "Expect: 102-continue" header.

Reworked Section 4.3 to describe more error cases and more detailed implementation instructions.

Disallow TLS Upgrade during SASL authentication (it is fine before or after). Clarified order of security layers.

Clarified that Authorization header with SASL response MUST NOT be

used with CONNECT.

Relaxed restriction for mixing SASL session ids on the same connection in certain cases.

Added new 235/236 status codes for successfully completed authentication.

Clarified that the body of the original request MUST NOT be sent until authentication is complete. Updated examples to reflect that.

Added an example with a POST request.

Changes since -04

Reworked the Introduction section.

Updated example 4.7.4 to include Authorization header in CONNECT request. This saves a round trip.

Added text that the client must use OPTIONS to find out which SASL mechanisms are supported by the server. Added an example.

Added text regarding the server requiring reauthentication when the client leaves the realm it authenticated in.

Some clarification about the CONNECT method. Added text that a CONNECT request should start the authentication exchange.

Incorporated comments from Raif S. Naffah and Keith Burdis.

Changes since -03

Fixed several errors in examples due to change from "sasl-mechanism" to "sasl-mechanisms".

More comments from Keith Burdis.

Changes since -02

Added discussions about CONNECT and session protection.

Added "Proxy servers considerations" Section. Updated examples to include headers that prevent caching.

Added Web farm considerations section that talks about a next response going to a different backend web-server.

Incorporated many suggestions/corrections from Keith Burdis.

Editorial changes. Cleanup some SHOULDs and MUSTs.

Changes since -01

Added examples

Split ABNF into client and server side. ABNF cleanup.

Many editorial changes.