HTTP Authentication with SASL
draft-vanrein-httpauth-sasl-03

Abstract

Most application-level protocols standardise their authentication exchanges under the SASL framework. HTTP has taken another course, and often ends up replicating the work to allow individual mechanisms. This specification adopts full SASL authentication into HTTP.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 23, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

HTTP has historically followed its own path for client authentication, while many other end-user protocols standardised on SASL; examples of SASL protocols include SMTP, IMAP, POP, XMPP, LDAP and AMQP. This specification introduces SASL to HTTP, so it may share in past and future work done for SASL in general.

Among the work that could be shared is backend authentication integration, which is possible due to protocol-independent SASL exchanges for any given method, making it easy to take them out of one protocol and inserting them into another. Although HTTP has adopted several SASL-compatible authentication methods, it uses various notations and so it still needs method-specific support at the HTTP level to translate them to a SASL backend.

In front-ends, a similar situation has arisen. The varying syntaxes for authentication methods have made it difficult to rely on support in most or all HTTP clients. When such clients could externalise their SASL handling to generic software such as a SASL library, then any extension to a library automatically spills over into the HTTP sphere. It is common for developers of web clients to also produce email clients, so a shared code base (and credential store) is not difficult to imagine.

Sharing is beneficial in both directions. HTTP benefits by being able to use anything from strong password mechanisms [RFC5802] without explicit support [RFC7804] in applications, up to GS2 mechanisms [RFC5801] with channel binding [RFC5056] [RFC5554] to TLS [RFC5929] based on pinning either the certificate for the TLS server or even a unique part of the individual TLS connection; for instance
Kerberos5 [RFC4120] currently uses Negotiate authentication [RFC4559] which is not as secure as GS2-KRB5-PLUS over SASL.

SASL also benefits; had it been the norm for HTTP, then the work to pass SAML over it [RFC6595] would probably have been done immediately. In fact, HTTP can still benefit from receiving standardised SAML20 inquiries over SASL, because it resolves the need for configuration of initiation paths and practices. Also, it removes authentication data from URIs, where they are not ideally placed.

In terms of security for HTTP applications, it appears beneficial to have very good authentication capabilities in the layers below the application; this is specifically true for applications developed in HTML and JavaScript, which tend to load code from various places, including code that is not always in the end user’s interest; since it already is a concern what identity information passes through these applications, it is certainly not advisable to use credentials in those places. Browsers are in a better position to take control over these assets, at the protocol levels of HTTP and TLS, and conceal credentials and possibly also identity from applications running on top. Insofar as tokens are needed, they can be derived from session keys using generally accepted key derivation schemes, but the session keys can be isolated from dynamic layers above HTTP.

### 2. Embedding SASL in HTTP

This specification integrates the SASL framework [RFC4422] into mainstream HTTP [RFC2616]. The SASL Authentication scheme follows the general structure for HTTP Authentication [RFC7235]. It uses the WWW-Authenticate and Proxy-Authenticate headers in responses from web servers and web proxies, respectively, and correspondingly the Authorization and Proxy-Authorization request header to answer to requests.

The SASL service name for the following embedding of SASL is HTTP; contrary to most other service names, it is spelled in uppercase, in line with what has become general practice in Kerberos and GSSAPI.

Since SASL prescribes channel binding to occur relative to TLS instead of to the application protocol, we can add that when the HTTPS transport is used.

Whether channel binding is used SHOULD remain a configuration choice in HTTP software, as it might interfere with intentional HTTPS proxying. Unintended proxying on the other hand, might lead to tapping of credentials under certain SASL mechanisms, and it may be
considered helpful to prevent such situations by relying on channel binding for at least those mechanisms.

2.1. HTTP Request and Response Messages

This section defines a few names for HTTP request and response messages, to be used in the remainder of this specification.

Initial Responses are those HTTP responses that set a status code 401 or 407, and that are sent when the HTTP server decides to initiate an authentication exchange.

Initial Requests are those HTTP requests that a client sends to initiate a fresh SASL authentication. User-Aware Requests are a variation defined further below, intended for attempts to address public resources under a given user name.

Intermediate Responses are HTTP responses to SASL authentication, with a status code set to 401 or 407. Intermediate Requests are those HTTP requests that a client sends to continue a SASL authentication after an Intermediate Response.

Final Responses either set a 200 or 403 status code, the first depicting success and the second depicting failure. Information in a Final 200 Response is provided in an Authentication-Info or Proxy-Authorization-Info header [RFC7615] instead of the headers used in Initial Responses and Intermediate Responses [RFC7235]. Note that proper interpretation of the Final 200 Response requires client state indicating that SASL authentication was used, or else the optional fields are not completely reliable information sources; cryptographic markers in the c2c field MAY be used to overcome this in a manner that defies abuse by rogue servers. The Final 403 Response never contains authentication-related headers.

The following fields, defined in upcoming sections, MUST and MAY be present in HTTP authentication exchanges for SASL:

<table>
<thead>
<tr>
<th>Request or Response</th>
<th>MUST have fields</th>
<th>MAY have fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Viewing Request</td>
<td>userview</td>
<td>mech</td>
</tr>
<tr>
<td>Initial Response</td>
<td>mech,s2s,realm</td>
<td>text</td>
</tr>
<tr>
<td>Initial Request</td>
<td>mech,s2s,c2c,realm</td>
<td>c2s,userview</td>
</tr>
<tr>
<td>Intermediate Response</td>
<td>c2c,s2c,s2s</td>
<td>text</td>
</tr>
<tr>
<td>Intermediate Request</td>
<td>c2c,c2s,s2s</td>
<td>userview</td>
</tr>
<tr>
<td>Final 200 Response</td>
<td>c2c</td>
<td>s2s</td>
</tr>
<tr>
<td>Final 403 Response</td>
<td>c2c</td>
<td></td>
</tr>
</tbody>
</table>
2.2. Authentication Field Definitions

Data for SASL is transported in the following fields:

- **c2s** holds SASL mechanism-specific data from client to server.
- **s2c** holds SASL mechanism-specific data from server to client.
- **s2s** holds opaque server data which the client MUST reflect in Intermediate Requests, to implement stateless SASL handling in the server. This is a requirement for the HTTP Authentication framework [Section 5.1.2 of [RFC7235]]. It MAY also be used to pass cached Section 2.3 authentication results.
- **c2c** holds opaque client data which the server MUST reflect in Intermediate Responses and Final 200/403 Responses. This can help to also make the client stateless.

It is not generally safe for SASL mechanisms to exchange c2s and s2c messages over unprotected transports. The c2c and s2s fields MUST be protected against tampering by rogue peers, and such protection also protects against tampering by rogue intermediates when using an unprotected transport. In addition, c2c and s2s fields may also need to be concealed from peers and intermediates.

Whether s2c is supplied in a Final 200 Response depends on the SASL mechanism, which may or may not have additional data to provide in this phase. Note that SASL requires empty s2c messages to be distinguished from absence thereof. When the server provides c2s and/or s2s data in a Final 200 Response, then it indicates that the supplied fields MAY provide one-step re-authentication with an empty s2c string, but the server MAY revoke this privilege at any time and for any reason; it would respond with an Initial Response in case of such revocation, but with a quick Final 200 Response if the one-step re-authentication is still acceptable.

The following fields support SASL within the HTTP Authentication Framework:

- **userview** selects a user view on the resources accessible over HTTP. This data selection purpose is unrelated to authentication. As a general principle, the userview MAY lead to changes in authentication triggers for URI paths, but MUST NOT change authentication triggers for the underlying resources.

- **realm** optionally names a scope of authorisation under the combination of scheme, server host name and userview. This does not signify a realm in the sense of realm crossover.
mech  In an Initial Response, the field is filled with a space-separated list of SASL mechanism names; In an Initial Request, the client chooses one SASL mechanism name; In a User Viewing Request, the field is fixated to ANONYMOUS [RFC4505] if it is provided.

text  is a user-oriented text explaining what information is needed.

2.3. Caching Authentication Results

When an HTTP server sends a Final 200 Response, it MAY include an "s2s" field. If it does this, then it SHOULD prepare to accept the field value for authentication in an Initial Request. However, credentials can expire or fall in disgrace for other reasons, so the server MAY still choose to reject the provided field.

When an HTTP client receives a Final 200 Response with an "s2s" field, it MAY memorise the fields for future reuse in an Initial Request, either with or without preceding Initial Response from the server. The HTTP client MUST use the realm as part of the decision which cached result to use, but it MAY extrapolate the results from one resource retrieval in an attempt to authenticate another.

When cached field result in a Final 407 Response then the HTTP client SHOULD remove the failing cache entry, and it SHOULD try again by going through a full SASL authentication cycle. The stateless nature of HTTP authentication is helpful in the sense that a new Initial Request can be sent to an older Initial Response.

3. Viewing Users on HTTP Services

The initial authentication mechanisms for HTTP were Basic and Digest [RFC2617] but these are no longer considered secure. These forms used the username in the URI together with a password, a combination that is now officially deprecated [Section 3.2.1 of [RFC3986]].

The use of a user name in a URI has not been deprecated, but has been withdrawn from HTTP because it was not included in the core HTTP specification [RFC2616]. With client identities that support realm crossover, it is possible to add this to SASL Authorization headers, through a new userview field, to be used like this:

```
GET / HTTP/1.1
Host: www.example.net
Authorization: SASL userview=snowboarders
```

There is a clear need for users in HTTP URIs, as can be seen from a wild variation of mappings into paths and sub-domains. The actual
use of the username part of the URI is however preferrable, and the URI format defines it in the authority part [Section 3.2 of [RFC3986]] which seems to match with the intended uses. A URI-based user name will probaly still be mapped to a path on a server, but need not be shown in the path in the URI. Better standardisation of user names is supportive of better integration with tools on both the client and server side.

We emphasise that the userview and client identity are orthogonal concepts, except for the special case where the client is viewing his own resources. This implies that it can be used with or without authentication. The userview field indicates the name space of resources beig addressed, while the client identity is involved in access rights.

The reason to integrate the userview with SASL is one of identity control; when the userview changes, one is visiting another part of a HTTP resource space, and it may then be proper to switch to another identity. This is in line with the scoping of protection spaces [Section 2.2 of [RFC7235]] to combinations of URI scheme, URI authority section and a server-defined realm string, where the URI authority section includes the user name in the URI.

Browsers currently show varying behaviours when supplied with a user name. This is mostly due to the deprecation [Section 3.2.1 of [RFC3986]] of userparts with a password embedded. This has created uncertainty and variation in handling of HTTP URIs with user names but no password. The userview field is intended to present a constructive alternative, where user names may once more be used to scope the resource name space to that of a user, without saying anything about authentication.

4. Security Considerations

The SASL exchange may be at risk of tampering when the sequence of HTTP messages is not secured to form one stream. The termination of such a secure layer MUST also terminate an ongoing SASL handshake.

SASL EXTERNAL can be a very efficient mechanism to combine with a secure transport layer if that includes authentication. This may be the case for TLS, especially when client-side authentication is deployed. Mechanisms other than EXTERNAL should take into account that a relation may exist between identities negotiated in the protective layer and the SASL exchange over HTTP. For example, a login account may be exchanged for an alias or group identity.

Channel binding is available in some SASL mechanisms. When used with HTTP SASL over TLS, it binds to the TLS channel, by default using the
type tls-unique [Section 3 of [RFC5929]]. When doing so, it is vital that either there be no renegotiation of the TLS handshake, or both secure renegotiation [RFC5746] and the extended master secret [RFC7627] are used.

5. IANA Considerations

This specification extends the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" with an "Authentication Scheme Name" SASL, referencing this specification.

This specification defines an additional entry in the registry "Generic Security Service Application Program Interface (GSSAPI)/Kerberos/Simple Authentication and Security Layer (SASL) Service Names" namely:

Service Name: HTTP
Usage: Web authentication using the SASL framework
Reference: TBD: this specification

6. References

6.1. Normative References


Appendix A. HTTP Server Environment Variables

We define a number of variables that SHOULD be passed from an HTTP SASL stack to applications run on top of it. The intention of defining these is to obtain maximum interoperability between these layers of software.

A common practice is to set environment variables with a given name to values that may be meaningful to applications. Those applications should be mindful about the possible meaning of absent variables.

The following variable MAY be available in both the SASL authenticated and unauthenticated state:

SASL_USERVIEW refers to the user name in the URI

SASL_USERVIEW that MUST NOT be accompanied by a password. It refines the view on resources held by the web server, usually from a general site to one that is user-specific. The URI user
is considered local to the web server (and, as a result of that, often its domain or security realm). This variable is only set when it is provided through the siteview field in the SASL exchange.

The following variables MUST NOT be available until SASL authentication is successful; it would be available when the server could send a 200 OK response:

SASL_SECURE is only "yes" (without the quotes) when a client is authenticated to the current resource. It never has another value; it is simply undefined when not secured by SASL.

SASL_REALM is the realm for which the secure exchange succeeded. A realm is not always used, because sites only need it when there are more than one in the same name space. When undefined in the SASL flow, this variable will not be set.

SASL_CLIENTID is the identity as confirmed through SASL authentication. Its content is formatted like an email address, and includes a domain name. That domain need not be related to the web server; it is possible for a web server to welcome foreign clients.

SASL_MECH indicates the mechanism used, and is one of the standardised SASL mechanism names. It may be used to detect the level of security.

SASL_S2S holds the accepted s2s field, and could be used as a random session identifier. It would normally be encrypted information.

SASL_S2S_ is a prefix for extra information that the server may extract from the s2s field in the HTTP SASL protocol flow. This depends on the authentication stack used in the web server.

Appendix B. Acknowledgements

Thanks to Henri Manson for making the first implementation of this specification and for feedback on the header formats.

Author’s Address
Rick van Rein
ARPA2.net
Haarlebrink 5
Enschede, Overijssel 7544 WP
The Netherlands

Email: rick@openfortress.nl