Workgroup: Network Working Group
Internet-Draft: draft-vanrein-httpauth-sasl-08
Published: 8 November 2022
Intended Status: Standards Track
Expires: 12 May 2023
Authors: R. Van Rein
ARPA2.net

HTTP Authentication with SASL

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 12 May 2023.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction
2. Embedding SASL in HTTP
   2.1. HTTP Request and Response Messages
   2.2. Authentication Field Definitions
   2.3. Caching Authentication Results
3. Server-Side User Name
4. Authentication Session Example
5. Security Considerations
6. IANA Considerations
7. References
   7.1. Normative References
   7.2. Informative References

Appendix A. HTTP Server Environment Variables

Appendix B. Acknowledgements

Author's Address

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, AMQP and MQTT. 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 of authentication mechanisms 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 not advisable to use credentials in those places. The HTTP layer is 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. Inasfar 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 [RFC7231], [RFC7232]. 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 requiring channel binding for those situations. HTTP in general allows a user session to hop between connections, and

browsers are likely to do this; to support this, the support of tls-server-end-point channel binding [Section 4 of [RFC5929]] is RECOMMENDED. Specific HTTP clients may exercise more control over connections to achieve stronger security; for those use cases, tls-unique channel binding [Section 3 of [RFC5929]] is RECOMMENDED. Generic web servers SHOULD support both forms of channel binding.

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 HTTP responses that normally set a status code 401 or 407, and that are sent when the HTTP server decides to initiate an authentication exchange. In addition, the server MAY send Initial Responses in other responses, to indicate to the client that it MAY try again to achieve better results [Section 4.1 of [RFC7235]].

Initial Requests are those HTTP requests that a client sends to initiate a fresh SASL authentication. The identity SHOULD be selected by the user independently from the URI; prior settings MAY however be remembered by a client for the combination of resource authority (scheme, host and possibly a separately communicated resource user name) with the server-sent realm string. The server can support a mixture of client identities for various roles or access levels through variation of realm strings. There is no current practice of server-side resource names in HTTP, but the generic URI schema presents this logic and it is easy to imagine an HTTP User header that a client could support.

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.

Positive Responses set a 200 status code to depict success. Information in this response is provided in an Authentication-Info or Proxy-Authentication-Info header [RFC7615] instead of the headers used in Initial Responses and Intermediate Responses [RFC7235]. Proper interpretation of a Positive 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.

Negative Responses also set a 401 or 407 status code and will often return the client to an earlier state that it recognises as one it has tried before. These responses should therefore offer

authentication to start again. In contrast to the Initial Response, there is now a c2c field that helps the client evaluate the request.

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

Request or Response	MUST have fields	MAY have fields
Initial Response	s2s,mech	realm
Initial Request	mech	c2s,realm,s2s
Intermediate Response	s2s	s2c
Intermediate Request	s2s	c2s
Positive Response		s2s
Negative Response	s2s,mech	realm

2.2. Authentication Field Definitions

Data for SASL is transported in the following fields:

c2s holds SASL token data from client to server. This field is transmitted with base64 encoding. The field is absent when the SASL client sends no token.

s2c holds SASL token data from server to client. This field is transmitted with base64 encoding. The field is absent when the SASL server sends no token.

s2s holds opaque server data which the client MUST reflect in Intermediate Requests and, when responding to an Initial Response, in the Initial Request. This is a necessity for a stateless HTTP Authentication framework [Section 5.1.2 of [RFC7235]]. It MAY be used in a Positive Response to pass a cacheable Section 2.3 authentication token in a future Initial Request.

The following fields support SASL within the HTTP Authentication Framework:

realm optionally names a scope of authorisation under the combination of scheme, server host name and possibly a HTTP user to implement the semantics of the generic URI username for resource selection. The realm does not necessarily match a domain name, which is used elsewhere as a realm notation.

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.

2.3. Caching Authentication Results

When an HTTP server sends a Positive Response, it MAY include an "s2s" field. If it does this, then it should be prepared 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 Positive 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 fields result in a Negative 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. Server-Side User Name

HTTP does not define a mechanism to specifically select the user as an authoritative resource name space on the server. Local syntax conventions exist, but lack universally reliable semantics. Basic authentication has been used to this effect, but this conflates the client identity with the server-side name space, which is not necessarily the same.

To allow HTTP servers to zoom in on user-specific information, the User header is hereby introduced. Its syntax matches the userinfo part of a URI, up to but excluding any colons in it:

```
User = *( unreserved / pct-encoded / sub-delims )
```

The value of the header MUST be percent-decoded before the server can use it to identify a local user.

The User header MAY be sent by clients, and HTTP servers MAY ignore it for any reason, including local user identities that do not comply to a more restrictive local user name syntax.

When an HTTP server makes use of the User header, it MUST include a Vary header in its response, with either a single "*" in it or the name "User". This informs caches that the response must be considered specific to the User header value in the matching request.

The User header may be used with or without any form of authentication. When used with authentication, the value of the percent-decoded header is considered part of the authority component of the resource, and therefore of the naming scope for the realm. Clients can use this refined notion of realm to select an authentication identity; when the value is known early enough, this may even help to select an X.509 client certificate. Note that the User header might be used together with the aforementioned practice of Basic authentication, but it can also replace it with an even simpler mechanism to free up the authentication exchange for HTTP SASL.

The distinction of a client-side user from a server-side user can benefit the use of credential schemes that are not tied to the HTTP server. A specific example of this is the current work on realm crossover with GS2-SXOVER-PLUS. The use of such a mechanism may offload security concerns from the application layer.

4. Authentication Session Example

This section is non-normative.

When an HTTP server receives a request for a protected page, it will send an Initial Response to ask for authentication with a special status code 401; for proxy access that would be 407, and header names change accordingly. Stripped down to the bare essentials, the server sends (this section adds whitespace for clarity)

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SASL
    realm="members only",
    mech="SCRAM-SHA-256 SCRAM-SHA-256-PLUS
          SCRAM-SHA-1 SCRAM-SHA-1-PLUS
          GS2-KRB5-PLUS GS2-KRB5",
    s2s="[xxxxx]"
```

The server offers SCRAM-* and GS2-KRB5 mechanisms. The variants with -PLUS provide additional channel binding, to ensure that authentication is specific to the current HTTPS connection, thus avoiding replay of the session across connections. Clients aware of HTTP connections may use connection-specific channel binding (tls-unique) while those that abstract from the connections must resort to weaker name-based channel binding (tls-server-end-point).

The server might have additionally offered the ANONYMOUS mechanism to allow the client to select "guest mode" access; the interaction would continue as authenticated, but presumably with limited access to HTTP resources and continued WWW-Authenticate headers to continue to offer authentication to improve resource information content. The

server might have offered EXTERNAL to allow the client to incorporate a TLS credential for authentication and possibly change to an authorization identity. The server might have offered GS2-SXOVER-PLUS if it is willing to connect to the client's home realm over Diameter, and thereby support realm crossover of SASL credentials.

The client initiates the SCRAM-SHA-256-PLUS mechanism, and to that end sends an Initial Request (this section shows square brackets around text that is transmitted with base64-encoding)

```
Authorization: SASL
    realm="members only",
    mech="SCRAM-SHA-256-PLUS",
    c2s="[n,,n=user,r=rOprNGfwEbeRWgbNEkqO]",
    s2s="[xxxxx]"
```

This mechanism is initiated by the client, hence the inclusion of the c2s token in the Initial Request. The contents of this field are specific to the selected mechanism, so SCRAM-SHA-256-PLUS in this case.

The SCRAM mechanism implementation is now initiated with the c2s token, and the server produces a followup challenge in a s2c token. To be able to validate future client messages against server-side state, it includes such state in an s2s token. This token is presumably protected from abuse with a signature and/or encryption, and it would likely identify the selected mechanism to validate during later rounds. The server packs all this in an Intermediate Response

```
HTTP/1.1 401 Unauthorized
WWW-Authentication: SASL
    s2c="[r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxF
          Ilj)hNlF$k0,s=W22ZaJ0SNY7soEsUEjb6gQ==,
          i=4096]",
    s2s="[yyyyy]"
```

Given that all server state is contained in this message, the client is free at any time to give up authentication and perhaps try another method. Normally however, it would proceed with the ongoing transaction.

The SCRAM mechanism continues with another round. The client engages in the prescribed cryptographic computations and packs an Intermediate Request along with updated state in the new c2s token

```
Authorization: SASL
    c2s="[c=biws,r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFIlj)hN
         lF$k0,p=dHzbZapWIk4jUhN+Ute9ytag9zjfMHgsqmmiz7AndVQ=]",
    s2s="[yyyyy]"
```

When the client has performed authentication properly, as determined by a server-side check of the c2s response token with the prior state in the s2s token, it can send a Positive Response along with the requested resource

```
HTTP/1.1 200 OK
WWW-Authentication: SASL
    s2c="[v=6rriTRBi23WpRR/wtup+mMhUZUn/dB5nLTJRsjl95G4=]",
    s2s="[zzzzz]"
```

The s2s token in a Positive Response is an optional extension. It is presented by the server to allow the client to speed up authentication in future requests. The client may send it whenever the server asks for the same realm string under the same scheme and authority; the client may make proactive assumpions about the realm string for new requests. Authentication must never be reused in another context than bound by channel binding. When used, the client immediate sends an Intermediate Response holding

```
Authorization: SASL
    realm="members only",
    s2s="[zzzzz]"
```

The server always has an option to refuse repeated authentication and forcing the client into a new authentication round. One reason for this could be that a session timed out. Another might be that the client is trying to use a credential outside a scope set by channel binding.

5. Security Considerations

It is not generally safe for SASL mechanisms to exchange c2s and s2c messages over unprotected transports. Furthermore, the SASL exchange may be at risk of tampering when the sequence of HTTP messages is not secured to form one stream. This means that a secure transport layer must be used, like TLS. The termination of such a secure layer MUST also terminate any ongoing SASL handshakes.

The s2s field 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, but dependent on the mechanism used, the s2s field may also need encryption to conceal their data from peers and intermediates.

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. 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.

The User header field as defined herein is orthogonal to issues of authentication and authorisation, and adds no security concerns.

6. 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
```

The capitalisation of the service name has historic origins and is now the preferred spelling for reasons of compatibility.

Please add the following entry to the Message Headers registry:

Header Field Name	Template	Protocol	Status	Reference
User		http	TBD	TBD:THIS_SPEC

7. References

7.1. Normative References

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <https://www.rfc-editor.org/info/rfc4559>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/info/rfc4422>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, DOI 10.17487/RFC5554, May 2009, <https://www.rfc-editor.org/info/rfc5554>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <https://www.rfc-editor.org/info/rfc5746>.

[RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI 10.17487/RFC5801, July 2010, <https://www.rfc-editor.org/info/rfc5801>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/info/rfc5929>.

[RFC6595] Wierenga, K., Lear, E., and S. Josefsson, "A Simple Authentication and Security Layer (SASL) and GSS-API Mechanism for the Security Assertion Markup Language (SAML)", RFC 6595, DOI 10.17487/RFC6595, April 2012, <https://www.rfc-editor.org/info/rfc6595>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC

7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/info/rfc7232>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <https://www.rfc-editor.org/info/rfc7615>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/info/rfc7627>.

7.2. Informative References

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <https://www.rfc-editor.org/info/rfc5802>.

[RFC7804] Melnikov, A., "Salted Challenge Response HTTP Authentication Mechanism", RFC 7804, DOI 10.17487/RFC7804, March 2016, <https://www.rfc-editor.org/info/rfc7804>.

[I-D.vanrein-dnstxt-krb1]
van Rein, R., "Declaring Kerberos Realm Names in DNS (_kerberos TXT)", Work in Progress, Internet-Draft, draft-vanrein-dnstxt-krb1-10, 21 October 2022, <https://www.ietf.org/archive/id/draft-vanrein-dnstxt-krb1-10.txt>.

Appendix A. HTTP Server Environment Variables

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

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.

REMOTE_USER is the client 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.

The following variable SHOULD be available while processing a request with a User header with locally acceptable syntax:

LOCAL_USER gives the HTTP User header value after syntax checking and percent-decoding. If used at all, it MUST be treated as a resource name space selector. This header does not describe the authenticated client identity, which is usually passed in a variable REMOTE_USER.

Appendix B. Acknowledgements

Thanks to Henri Manson for making the first implementation of this specification and for feedback on the header formats. The specification also benefited from input by Daniel Stenberg.

This work was supported with an open source development fund from NLNet.

Author's Address

Rick van Rein
ARPA2.net

Haarlebrink 5
Enschede

Email: rick@openfortress.nl