Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 12, 2019

R. Van Rein
ARPA2.net

November 8, 2018

HTTP Authentication with SASL
draft-vanrein-httpauth-sasl-01

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 May 12, 2019.

Copyright Notice

Table of Contents

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 has 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 GS2 mechanisms [RFC5801] with channel binding [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, becuase it resolves the need for configuration of initiation paths and practices. Also, it removes authentication data from URIs, where they are not ideally placed.

TODO: Does this do justice to current SAML over HTTP?

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. 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 [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-Authentication-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:

Request or Response	MUST have fields	MAY have fields
Unauth Request		userview,visitor
Initial Response	mech,s2s,realm	encalg,enckid,text
Initial Request	mech,s2s,c2c,realm	encalg,enckid,c2s, userview,visitor
Intermediate Response	mech,c2c,s2c,s2s	text
Intermediate Request	mech,c2c,c2s,s2s	userview,visitor
Final 200 Response	mech,c2c,name,realm	s2s
Final 403 Response		

2.2. Authentication Field Definitions

Data for SASL is transported in the following fields:

c2s holds SASL mechanism-specific data from client to server, usually encrypted during realm crossover.

s2c holds SASL mechanism-specific data from server to client, usually encrypted during realm crossover.

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]].

c2c holds opaque client data which the server MUST reflect in Intermediate Responses and Final 200 Responses. This can help to also make the client stateless.

encalg identifies an encryption algorithm to protect the c2s and s2c fields, especially during realm crossover. When this field is absent, the c2s and s2c fields are not encrypted but literally follow the SASL mechanism exchange.

enckey is an identity of the encryption key used under encalg. The fields enckey and encalg MUST always be paired; either both are present, or both are absent.

As in other protocols, it is not safe for all 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.

mech selects the SASL mechanism to use. It MUST be reflected from the preceding message, except: 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-Aware Request, the field is fixated to ANONYMOUS.

visitor is a hint that suggests an unauthenticated client identity. It cannot be used as a basis of security, other than to point at a SASL backend to use for authentication under realm crossover.

name is the authorised user@domain.name or domain.name identity.

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

3. Support for Realm Crossover

HTTP services tend to define users as part of their own secure realm. This culminates in competitions over user names in the most popular services. It also leads to variations in user names across services, difficult to both users and their contacts. More seriously, it leads to uncertainty about the same-ness of users at different services. Anyone can claim to be your bank, but there is no basis of trusting such names.

A more flexible model allows the client to determine their names, and locate them under their own domain. Provided that services use domain-specific methods for identity assurance, the client can be represented by their complete form, and no concern of same-ness of users names across such services need exist. As a pleasant side-effect, the competition over user names has become a local matter, to be resolved by the client.

HTTP SASL supports this model with a number of choices. First, the format of a user identity is formalised to be a Network Access Identifier [RFC7542] of the form

nai_clientid ::= utf8-username "@" utf8-realm

Second, there is an option to claim a client identity at all times, without authentication, in any of the forms of a Network Access Identifier,

nai_visitor ::= nai

In both cases, the right-hand-side symbols refer to the quoted specification for a NAI.

The nai_clientid is reported back from SASL to the HTTP service. When it is provided, it has been validated. See Appendix A for the mechanism of passing such information from an HTTP server to an application service.

The nai_visitor is used as a field to the SASL Authorization header, and can be provided at any point. The form might be sent on its own, as in

Authorization: SASL visitor=john@example.com

This identity is also passed from HTTP server to application server, but it comes straight from this header. This means that it is just a hint at an identity, without any formal status. Services seem to enjoy such hints, and receiving it in a standardised manner can be beneficial for support in development environments, and it allows more client control.

It is anticipated that local information is subject to access control rules that determine specific client identities, probably a mixture of individuals and entire domains, who can exercise a variety of rights to the resource offering over HTTP. This model can be supportive of local users as well as foreign ones.

The nai_visitor MAY be used as a hint towards a SASL realm to address, especially during the Initial Request generation. When no nai_visitor is available at that time, local policy dictates what SASL realm to use. This is important because the SASL realm determines which mechanisms can be offered.

It could happen that an Intial Response comes back with a different SASL realm in the nai_visitor than assumed during the generation of the Initial Request. In such cases, the requested SASL realm is

approached. When its offering of mechanisms does not contain the choice made by the client, a new Initial Request SHOULD be generated, to support the client in their change of mind. There does not seem to be a reason to grant such changes to occur more than once.

3.1. Encrypting SASL for Realm Crossover

Most SASL methods are unsafe to use over plaintext channels; it is a common and good practice to run it over a secure channel such as TLS. This is precisely the condition of HTTP SASL with server-based accounts. But TLS only protects the direct link between a client and a server; during realm crossover the server might effectively be a man-in-the-middle. This is especially true when a SASL backend in the client's domain is asked to perform the authentication for the HTTP server.

The solution is to encrypt the SASL traffic while in transit. Since the client and backend authentication service know each other, they can resort to simple mechanisms such as symmetric encryption and integrity checking. This means that the client and backend need to share a random key and an algorithm. Random keys can for example be bootstrapped through an ECDHE exchange, or as by-product [RFC5705] of a TLS session.

Encryption is determined by an algorithm name, a key identifier and the HTTP server name as it occurs in the Server Name Indication in TLS. The algorithm name is captured in an enc field and the key identifier in a enckid field to the SASL Authorization header:

Authorization: SASL encalg="aes128-cts-hmac-sha256-128" enckid="1783"

Key and algorithm naming can be defined locally. HTTP SASL passes them through verbatim. SASL end points interpret them, and to improve interoperability the algorithm names SHOULD be the names or numbers of Kerberos encryption types. This is a practical choice, because most SASL end points have access to suitable code for handling such forms.

Encryption MAY be requested during the Initial Request, which itself is not yet a sensitive message. When its use is desired by a client, it MUST be included in the Initial Response, and will then be used for the c2s and s2c fields in that message and any following. Encryption SHOULD be used if an AT character U+0040 is sent as part of a visitor field.

A separate specification will introduce a mapping of SASL into Diameter messages, including support for the enc and key fields.

3.2. 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 siteuser 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 are 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. Unfortunately, this also means that user names in HTTP URIs are sometimes suppressed or warned about. 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 at all.

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.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <https://www.rfc-editor.org/info/rfc7542>.

[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>.

6.2. Informative References

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <https://www.rfc-editor.org/info/rfc2617>.

[RFC4505] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism", RFC 4505, DOI 10.17487/RFC4505, June 2006, <https://www.rfc-editor.org/info/rfc4505>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[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>.

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 variables MAY be available in both the SASL authenticated and unauthenticated state:

SASL_SITEVIEW refers to the user name in the URI and SHOULD NOT be used with 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.

SASL_VISITOR refers to self-identification of the client independent of authentication. Its general form is that of an email address, but its local part MAY be empty. This variable may be used to determine the domain against which the client intends to authenticate and, from that, the SASL mechanisms that can be offered.

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