HTTP
Internet-Draft
Intended status: Standards Track
Expires: 6 June 2026

N. Williams
3 December 2025

HTTP Bearer Auth Method Extensions
draft-williams-http-bearer-extension-00

Abstract

This document specifies an improved HTTP 401 and 407 flow for Bearer authentication where user-agents (or client applications) can automatically fetch requested tokens from a Security Token Service (STS). A fallback to an OpenID Connect (OIDC) redirect flow is included.

This improved 401/407 Bearer flow, when used, elides the need for Proof Key for Code Exchange (PKCE) and does not impose on application Universal Resource Identifier (URI) query parameter design. As well this extension allows for user-agent caching of tokens.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-williams-http-bearer-extension/.

Discussion of this document takes place on the NETWORK Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/.

Source for this draft and an issue tracker can be found at https://github.com/nicowilliams/http-bearer-ext.

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/.

This Internet-Draft will expire on 6 June 2026.

Copyright Notice

Table of Contents

1. Introduction

HTTP [HTTP] has a number of authentication methods available, including Bearer [BEARER] that uses OAuth 2.0 [OAUTH2], which is the subject of this document. Some HTTP authentication methods, like Negotiate [NEGOTIATE], allow the user-agent to automatically respond to a 401 or 407 status response when possible by creating a token locally using the GSS-API [GSSAPI], typically using Kerberos [KERBEROS]. However, the Bearer method effectively requires that the client application respond to the 401/407 authentication request rather than the HTTP user-agent, and it provides little metadata that either can use to figure out how to obtain a Bearer token. Today an OAuth application just has to know how to obtain the Bearer token, and has to be configured appropriately, and this greatly limits the applicability of the 401/407 flows.

Note that we will mostly refer to 401 flows here, but everything that applies to 401 flows also applies to 407 flows.

Here we specify extension auth-params to the 401 WWW-Authenticate: Bearer HTTP response so that either the client application or the user-agent can figure out how to obtain the requested token after applying local policy.

As well we provide a way to improve the OIDC redirect flow to make use of this extension where possible, and this provides a migration path to a world where URI q-param camping and PKCE [PKCE] and code-to-token conversions are no-longer needed just to use OAuth 2.x.

The extended Bearer 401 flow presented here is a glorified redirect that avoids the need for camping on the application's URI q-params.

1.1. Motivation

1.1.1. Automation of Bearer

In general it would be better for user-agents to know how to execute HTTP auth methods, and generally they do, except for Bearer, where they often only know when the application has provided the user-agent with a token ahead of it being needed.

Some user-agents, such as for example (at the time of this writing) curl, do not support redirect flows quite well enough. This causes users to work around these limitations by, for example, scripting around these user-agents' limitations.

In order to make such user-agents automatically fetch tokens we want improve the 401 Bearer flow so that the user-agent can learn where and how to acquire a token. This necessitates OPTIONAL local policy, but only because some such user-agents do not enable redirect following by default, whereas browser-type user-agents chase redirects by default.

1.1.2. Making OAuth, OIDC, and SAML not Camp on URI Design

Besides automatic Bearer token fetching in tools such as curl there is also the desire to not impose on application URI design.

1.1.3. Reducing Load on Token Issuers by Facilitating Token Caching

By making auth requests a first-class operation (by using 401 where possible instead of a redirect) and the user-agent responsible for executing auth request we make it possible for the user-agent to cache tokens during their remaining lifetime. This allows for a significant reduction in issuer load in some cases.

The token cache can be in-memory or stored in local storage (e.g., a local filesystem) for sharing among the user's non-browser applications, further reducing issuer load.

1.1.4. Enabling the Use of Bearer for Proxy Auth

Currently Bearer cannot be used for proxy auth (407) in browsers as browsers do not know how to fetch the required token.

For many users this means being stuck with the New Technology LAN Manager (NTLM) mechanism via Negotiate [NEGOTIATE], or Kerberos [KERBEROS] via Negotiate, or SCRAM [SCRAM]. But NTLM is weak and dangerous, Kerberos is often seen as legacy that one should be able to avoid, and SCRAM requires passwords and thus is not a single-sign-on solution.

Maintaining infrastructure for many auth methods is a burden on operators. Operators would like to reduce that burden. And many operators want to be able to use OAuth as the primary or only auth method. But there is a long tail of work to do to make OAuth a replacement for Kerberos. Proxy auth is part of that long tail, thus by eliminating that part of the long tail we make the tail shorter.

1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Extended 401 WWW-Authenticate: Bearer Flow

Currently the following auth-params are supported in a 401 response with WWW-Authenticate=Bearer:

* realm
* scope
* error
* error_description
* error_uri

We add the following additional, OPTIONAL auth-params:

* issuer with a URI as a value
* server_token_type with a token type Universal Resource Name (URN) as a value
* server_token with a token to be forwarded to the issuer by the user-agent or client application
* state with a "state" value -- this corresponds to the OIDC state q-param in a redirect to the OpenID provider

The issuer parameter allows the user-agent or client application to discover the URI of a Security Token Service (STS) to visit to get a token. Though of course a user-agent or client application could map a realm auth-param value to an STS URI, or even the realm could consist of an issuer URI, however by having an auth-param intended to carry an issuer URI we avoid ambiguities about the meaning of "realm".

The server_token auth-param allows the server to pass a cryptographic token to the issuer via the user-agent's or client application's subsequent request to the issuer. The user-agent or client application will set a new requested header Token-Request-Pref to the value in the 401 response's server_token auth-param.

Essentially this functions like a redirect response, but with a 401 status code and with the Server-Token and Server-Token-Type carrying the token and token type from the server_token and server_token_type auth-params respectively, if present.

The user-agent and/or client application MUST apply local policy, where any is set, as to whether to execute a token request against the requrested issuer.

When making the token request to the issuer the user-agent or client application MUST also set the following fields:

* Referer, as if the 401 response had been a redirect

* a new Token-Request header to indicate that the request is in response to an extended Bearer token request by the server

* Token-Request-Token, set to the token from the server_token auth-param, if present

When making the token request to the issuer the user-agent or client application SHOULD also set the following fields:

* Token-Request-Auth-Params, with all auth-params other than the issuer and server_token parameters.

When the STS supports this feature then a successful response MUST be a 200 status code response if successful rather than a 3xx redirect response, with a field Location field set to the value of the Token-Request-URI field (if present) from the request to the STS and with the issued token in the response body.

The value forms of these headers are specified below (Section 6).

2.1. Example flow

Request to https://some.origin.example/some/resource:

```
GET /some/resource HTTP/1.1
Host: some.origin.example
```

Here the user-agent or client MAY use a new field, Token-Request-Pref, to indicate a preference for 401 or 3xx flows. Not specifying a preference allows the server to honor that preference instead of having to guess it from fingerprinting the user-agent.

The server may not support this feature, in which case it will either use an unextended 401 WWW-Authenticate: bearer or a 3xx OIDC redirect flow. Here we show the server responding with an extended 401 WWW-Authenticate: Bearer:

```
HTTP/1.1 401 Authentication required
WWW-Authenticate: Negotiate
WWW-Authenticate: Bearer issuer=https://iam.foo.example/v1/token
        scope=read:some/resource
        server_token=Bearer:<some-JWT>
```

Token request to STS:

```
POST /v1/token
Host: iam.foo.example
Token-Request-Auth-Params: scope=read:some/resource
Token-Request-Token: Bearer:<some-JWT>
Token-Request-URI: https://some.origin.example/some/resource

grant_type=urn:ietf:params:oauth:grant-type:token-exchange&
audience=some.origin.example&
scope=read:some/resource&
...
```

The token request to the STS may elicit a 401 response instructing the user-agent or client application to authenticate in some manner. For example, an STS might use Bearer with a "token granting token" that the user-agent just has to have a priori (how such a token-granting-token is acquired and refreshed is out of scope for this document), and/or it might use Negotiate, Basic, or other authentication methods. The user-agent or client application might also be configured to know which authentication methods a given issuer might require and thus "pre-flight" an Authorization field in the token request to the STS. We elide the 401 and retry here for brevity.

It is also possible that an STS might issue a redirect or extended 401 flow to another STS. This too is elided here, but the user-agent MUST save the state from the first token request, process the second, return to the first STS, and retry the first request with the newly acquired credential.

Thus after any authentication exchanges, a successful response from the STS if it supports this feature is a 200 Ok response with a Location field to remind the user-agent or client its state (but the user-agent or client MUST recall this metadata anyways):

```
HTTP/1.1 200 Ok
Location: https://some.origin.example/some/resource

<token>
```

XXX Well, perhaps we should dispense with the Location field here given that we require the user-agent/client to know that anyways.

XXX It might be desirable to allow the STS to tell the user-agent to set some select fields in its return request to the origin, perhaps headers named Token-Response* or specifically Authorization.

Otherwise if the STS does not support this feature but it would issue the token then it should issue a code and respond with a redirect as in an OIDC redirect flow:

```
HTTP/1.1 302 Found
Location: https://some.origin.example/some/resource?code=..&state=..
```

The user-agent or client MUST remember the URI in case the STS responds with 200 Ok with the token as the response body and without a Location field.

Armed with the requisite token (if it was issued), the user-agent/client SHOULD retry the original request with the Authorization field set to Bearer <token> with the token returned by the STS.

Note that the extended Bearer 401 flow MUST NOT change the browser's window location when the user-agent is a browser. Instead the user-agent MUST execute the STS request and then retry the original request (if the STS response is successful) in the same sort of way that it would perform any Kerberos KDC exchanges had the selected HTTP authentication method been Negotiate [NEGOTIATE], that is: the STS interaction MUST NOT be visible to any scripts on the web page that produced the 401 response.

2.2. Server Response Type Selection

When a server needs the client to authenticate or obtain a new access token (with different scopes), it can respond with a 3xx redirect or a 401. The server needs to be able to decide which to use.

When the client includes a Token-Request-Pref field in its request, the server that supports extended 401 flows SHOULD use the client's preference.

Otherwise the server should use local configuration, possibly including user-agent fingerprinting (such as by user-agent substring matching), to pick a response type. When the user-agent is interactive the server can return a 401 with a body with a script that will load a page that will trigger an OIDC redirect flow, possibly after a few seconds, or with a link that the user can click on. Or the server can expose a way for the user to choose the flow to use, though this presumably will not usually be done except for debugging or demonstration purposes.

2.3. User-Agent Use

Ideally user-agents will implement the 401 flow, but typically they will not include the Token-Request-Pref field in their requests, either at all or unless configured to do so (possibly only for some sites).

Reasons for not including the Token-Request-Pref field include:

* avoiding request bloat

* avoiding fingerprinting

User-agents SHOULD be configurable, possibly using a list of sites and domainname stems for selecting when to include this header.

2.4. Client Application Use

Client applications can also implement the extended 401 flow as they would a regular unextended 401 flow.

As with user-agents, when it is the client application that implements the extended 401 flow the application SHOULD have local policy and configuration.

2.5. Mapping realm to issuer

For the non-extended 401 case, user-agents and clients can be configured to map the realm auth-param to an issuer, and engage in the extended 401 flow anyways.

2.6. Token Caching

Where a user-agent or client application implements this extended Bearer 401 flow it can and SHOULD also implement some sort of token cache to alleviate load on STSes and to improve availability. Proxies, however, should never cache token request responses, and the STS SHOULD set the Cache-Control field accordingly to include the the following directives:

* no-cache, unqualified

* max-age set to the number of seconds after which the token should not be used due to impending expiration

* s-maxage=0 so as to reinforce that shared caches must not cache this token

If the Cache-Control is absent then the user-agent or client MAY infer:

* no-cache, unqualified

* max-age with any lifetime it can glean from the token if that metadata is unencrypted in the token

Similarly, if Cache-Control but no-cache is absent then the user-agent or client MAY infer it, and if max-age is absent then the user-agent or client MAY infer it as well.

3. Extended Redirect Flow

The extended redirect flow has a normal 3xx response from the server but with the WWW-Authenticate field present, possibly including new auth-params.

Note well that some user-agents terminate 3xx response processing as soon as they see the expected Location field, and thus may not see a WWW-Authenticate field if it comes after, and note that proxies are allowed to reorder fields (but not field values for multi-valued fields), therefore it is not possible for a server to reliably count on the WWW-Authenticate field being seen by the client.

Still, when a client sees the WWW-Authenticate field then it SHOULD act as though the response was a 401 response engaging in an extended OAuth 401 flow.

4. Negotiation of 401 vs Redirect Flows

A server may only support redirect flows, only 401 flows, or it may support negotiation as described here. Ideally all servers will eventually support negotiation, and/or all user-agents will support the extended Bearer 401 flow.

The Token-Request-Pref field has two values: 401 and redirect. When present the server SHOULD use the expressed preference if it needs a new token.

When this field is absent the server must choose a response type using configuration and user-agent fingerprinting.

5. Local Policy

Local policy is OPTIONAL. When present, local policy expresses:

* whether to enable automatic token request handling when using the extended Bearer 401 flow
* allow lists for issuers and/or realms
* mappings of realms to issuers
* authentication methods to allow at each issuer, if any
* global and site-specific flow type preferences, and whether to express those flow type preferences via the Token-Request-Pref field
* for each allowed issuer optionally record its authentication method preferences so that the user-agent or client application may pre-flight an Authorization field in the token requests to that issuer

6. New fields

* Token-Request-Pref is a structured field of kind Item with a value of "401" or "redirect"
* Token-Request is a structured field of kind Item with a value of "true" if present
* Token-Request-Token is a structured field of kind Item with a token as its value

* Token-Request-Auth-Params is a structured field of kind List with the 401 response's auth-params from the WWW-Authenticate: Bearer field

* Token-Request-URI is a field of kind Item containing the URI that requested the token

7. New auth-params

* issuer

* server_token_type

* server_token

* state

See section 2 (Section 2) for details.

8. Wider Applicability: Negotiate Auth w/o GSS or Kerberos Implementations!

This section is INFORMATIONAL.

A similar extension design can be applied to the Negotiate HTTP auth method that would allow user-agents to support Negotiate without having a local implementation of the GSS-API [GSSAPI], Kerberos [KERBEROS], or other GSS mechanisms. All that would be needed is an STS-like service that itself can issue Kerberos tickets or can impersonate users to a Kerberos Key Distribution Center (KDC) and thus obtain tickets on their behalf, then the STS can mint GSS-API tokens for the Kerberos mechanism [GSSKRB5]. The rest of the flow would be remarkably similar to the extended Bearer 401 flow.

An implementation of such an Negotiate token issuer could have a certification authority credential to mint user certificates for use with PKINIT [PKINIT] to impersonate users to a Kerberos KDC. Thus such a service would not need intimate access to the KDC's key database, and can be interoperate with any KDC implementation that supports PKINIT. In all cases such a service would be as security-sensitive as a KDC, and would best be seen as part of the KDC, and for discovery purposes should probably be co-located with the KDC.

As in the case of Bearer an issuer URI auth-param for Negotiate might be useful, though perhaps the user will have to have selected an issuer from a server-provided list of choices. In the case of Kerberos, for example, the choice of token issuer will be dependent on the user's credentials' issuer, thus local configuration at the

user-agent will be the best source of an issuer value for a Negotiate 401 flow that resembles the extended Bearer 401 flow. But even in the case of Kerberos if the user-agent can help the server include the correct issuer auth-param value then the flow can work in spite of the lack of local configuration.

9. Single Sign-On

This section is INFORMATIVE.

When engaging in an OIDC or SAML redirect flow, or an extended 401/407 flow as in this document, ultimately the user-agent, the client, and/or the user will have to authenticate the user to an issuer. How that is done initially (e.g., with passwords, smartcards and PINs, etc.) is out of scope here, but for Single Sign-On (SSO) one will want to distinguish between "initial authentication" and non-initial authentication (i.e., _not_ passwords, smartcards, PINs, etc.), much like Kerberos [KERBEROS] does. A user-agent could use cookies, or a "token-granting token" (similar to a Kerbero ticket-granting ticket, both sharing the same acronym of 'TGT') to repeatedly authenticate to an issuer for a limited time after which the user has to once again engage in "initial authentication".

10. IANA Considerations

As there is no registry of HTTP auth method auth-params, we do not register any of the new auth-params created here. However, it might be desirable to create such a registry before this document's publication as an RFC.

This document registers several new HTTP fields listed below.

10.1. HTTP Field Registrations

* Field Name: Token-Request-Pref

  Status: permanent

  Structured Type: Item

  Reference: this document

* Field Name: Token-Request

  Status: permanent

  Structured Type: Item

Reference: this document

* Field Name: Token-Request-Token

  Status: permanent

  Structured Type: Item

  Reference: this document

* Field Name: Token-Request-Auth-Params

  Status: permanent

  Structured Type: Item

  Reference: this document

* Field Name: Token-Request-URI

  Status: permanent

  Structured Type: Item

  Reference: this document

11. Security Considerations

Existing user-agents that support redirect flows usually trust all redirects. As a result we make local policy OPTIONAL for use in the automatic the extended Bearer 401 flow. The specification of an "abstract schema" for local policy is an improvement.

User-agents such as browsers MUST NOT allow scripts embedded or injected into web pages to access to the STS responses from extended Bearer 401 flow. This is already the case, thus this requirement is not a new requirement and does not impose on other working groups or standards development organizations.

Apart from the above this document does not add new security considerations beyond those of OAuth and OIDC, but it does remove the need for "codes" and PKCE [PKCE] by not abusing URI q-params to carry tokens or token-like "codes", instead carrying tokens only in the Authorization field and in the STS response body invisible to scripts on web pages. As well by not abusing URI q-params no sensitive cryptographic material such as "codes" and tokens can end up in user-agent history nor in client- or server-side logs.

Thus this extended Bearer 401 flow is strictly an improvement in security for OAuth.

But note that when the server does not implement the extended Bearer 401 flow then a fallback to a redirect flow may happen, thus user-agents and/or clients still need to implement PKCE [PKCE].

12. References

12.1. Normative References

[BEARER] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[OAUTH2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[GSSAPI] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <https://www.rfc-editor.org/info/rfc2743>.

[GSSKRB5] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, DOI 10.17487/RFC4121, July 2005, <https://www.rfc-editor.org/info/rfc4121>.

[KERBEROS] 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>.

[NEGOTIATE] 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>.

[PKCE] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

[PKINIT] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, DOI 10.17487/RFC4556, June 2006, <https://www.rfc-editor.org/info/rfc4556>.

[SCRAM] 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. Acknowledgements

Author’s Address

Nico Williams
Austin
United States
Email: nico@cryptonector.com