Internet Engineering Task Force (IETF)
Request for Comments: 7235
Obsoletes: 2616
Updates: 2617
Category: Standards Track
ISSN: 2070-1721

R. Fielding, Editor
Adobe
J. Reschke, Editor
greenbytes

June 2014

# Hypertext Transfer Protocol (HTTP/1.1): Authentication

## Abstract

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypermedia information systems. This document defines the HTTP Authentication framework.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741[1].

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7235[2].

## Copyright Notice



[1] https://www.rfc-editor.org/rfc/rfc5741.html#section-2
[2] http://www.rfc-editor.org/info/rfc7235
[3] http://trustee.ietf.org/license-info

# Table of Contents

# 1. Introduction

HTTP provides a general framework for access control and authentication, via an extensible set of challenge-response authentication schemes, which can be used by a server to challenge a client request and by a client to provide authentication information. This document defines HTTP/1.1 authentication in terms of the architecture defined in "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing" [RFC7230], including the general framework previously described in "HTTP Authentication: Basic and Digest Access Authentication" [RFC2617] and the related fields and status codes previously defined in "Hypertext Transfer Protocol -- HTTP/1.1" [RFC2616].

The IANA Authentication Scheme Registry (Section 5.1) lists registered authentication schemes and their corresponding specifications, including the "basic" and "digest" authentication schemes previously defined by RFC 2617.

## 1.1. Conformance and Error Handling

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Conformance criteria and considerations regarding error handling are defined in Section 2.5 of [RFC7230].

## 1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with a list extension, defined in Section 7 of [RFC7230], that allows for compact definition of comma-separated lists using a '#' operator (similar to how the '*' operator indicates repetition). Appendix B describes rules imported from other documents. Appendix C shows the collected grammar with all list operators expanded to standard ABNF notation.

# 2. Access Authentication Framework

## 2.1. Challenge and Response

HTTP provides a simple challenge-response authentication framework that can be used by a server to challenge a client request and by a client to provide authentication information. It uses a case-insensitive token as a means to identify the authentication scheme, followed by additional information necessary for achieving authentication via that scheme. The latter can be either a comma-separated list of parameters or a single sequence of characters capable of holding base64-encoded information.

Authentication parameters are name=value pairs, where the name token is matched case-insensitively, and each parameter name MUST only occur once per challenge.

```
auth-scheme    = token

auth-param     = token BWS "=" BWS ( token / quoted-string )

token68        = 1*( ALPHA / DIGIT /
                     "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

The token68 syntax allows the 66 unreserved URI characters ([RFC3986]), plus a few others, so that it can hold a base64, base64url (URL and filename safe alphabet), base32, or base16 (hex) encoding, with or without padding, but excluding whitespace ([RFC4648]).

A 401 (Unauthorized) response message is used by an origin server to challenge the authorization of a user agent, including a WWW-Authenticate header field containing at least one challenge applicable to the requested resource.

A 407 (Proxy Authentication Required) response message is used by a proxy to challenge the authorization of a client, including a Proxy-Authenticate header field containing at least one challenge applicable to the proxy for the requested resource.

```
challenge   = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

> **Note:** Many clients fail to parse a challenge that contains an unknown scheme. A workaround for this problem is to list well-supported schemes (such as "basic") first.

A user agent that wishes to authenticate itself with an origin server — usually, but not necessarily, after receiving a 401 (Unauthorized) — can do so by including an Authorization header field with the request.

A client that wishes to authenticate itself with a proxy — usually, but not necessarily, after receiving a 407 (Proxy Authentication Required) — can do so by including a Proxy-Authorization header field with the request.

Both the Authorization field value and the Proxy-Authorization field value contain the client's credentials for the realm of the resource being requested, based upon a challenge received in a response (possibly at some point in the past). When creating their values, the user agent ought to do so by selecting the challenge with what it considers to be the most secure auth-scheme that it understands, obtaining credentials from the user as appropriate. Transmission of credentials within header field values implies significant security considerations regarding the confidentiality of the underlying connection, as described in Section 6.1.

```
credentials = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

Upon receipt of a request for a protected resource that omits credentials, contains invalid credentials (e.g., a bad password) or partial credentials (e.g., when the authentication scheme requires more than one round trip), an origin server SHOULD send a 401 (Unauthorized) response that contains a WWW-Authenticate header field with at least one (possibly new) challenge applicable to the requested resource.

Likewise, upon receipt of a request that omits proxy credentials or contains invalid or partial proxy credentials, a proxy that requires authentication SHOULD generate a 407 (Proxy Authentication Required) response that contains a Proxy-Authenticate header field with at least one (possibly new) challenge applicable to the proxy.

A server that receives valid credentials that are not adequate to gain access ought to respond with the 403 (Forbidden) status code (Section 6.5.3 of [RFC7231]).

HTTP does not restrict applications to this simple challenge-response framework for access authentication. Additional mechanisms can be used, such as authentication at the transport level or via message encapsulation, and with additional header fields specifying authentication information. However, such additional mechanisms are not defined by this specification.

## 2.2. Protection Space (Realm)

The "*realm*" authentication parameter is reserved for use by authentication schemes that wish to indicate a scope of protection.

A *protection space* is defined by the canonical root URI (the scheme and authority components of the effective request URI; see Section 5.5 of [RFC7230]) of the server being accessed, in combination with the realm value if present. These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, that can have additional semantics specific to the authentication scheme. Note that a response can have multiple challenges with the same auth-scheme but with different realms.

The protection space determines the domain over which credentials can be automatically applied. If a prior request has been authorized, the user agent MAY reuse the same credentials for all other requests within that protection space for a period of time determined by the authentication scheme, parameters, and/or user preferences (such as a configurable inactivity timeout). Unless specifically allowed by the authentication scheme, a single protection space cannot extend outside the scope of its server.

For historical reasons, a sender MUST only generate the quoted-string syntax. Recipients might have to support both token and quoted-string syntax for maximum interoperability with existing clients that have been accepting both notations for a long time.

# 3. Status Code Definitions

## 3.1. 401 Unauthorized

The *401 (Unauthorized)* status code indicates that the request has not been applied because it lacks valid authentication credentials for the target resource. The server generating a 401 response MUST send a WWW-Authenticate header field (Section 4.1) containing at least one challenge applicable to the target resource.

If the request included authentication credentials, then the 401 response indicates that authorization has been refused for those credentials. The user agent MAY repeat the request with a new or replaced Authorization header field (Section 4.2). If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user agent SHOULD present the enclosed representation to the user, since it usually contains relevant diagnostic information.

## 3.2. 407 Proxy Authentication Required

The *407 (Proxy Authentication Required)* status code is similar to 401 (Unauthorized), but it indicates that the client needs to authenticate itself in order to use a proxy. The proxy MUST send a Proxy-Authenticate header field (Section 4.3) containing a challenge applicable to that proxy for the target resource. The client MAY repeat the request with a new or replaced Proxy-Authorization header field (Section 4.4).

# 4. Header Field Definitions

This section defines the syntax and semantics of header fields related to the HTTP authentication framework.

## 4.1. WWW-Authenticate

The "WWW-Authenticate" header field indicates the authentication scheme(s) and parameters applicable to the target resource.

```
WWW-Authenticate = 1#challenge
```

A server generating a 401 (Unauthorized) response MUST send a WWW-Authenticate header field containing at least one challenge. A server MAY generate a WWW-Authenticate header field in other response messages to indicate that supplying credentials (or different credentials) might affect the response.

A proxy forwarding a response MUST NOT modify any WWW-Authenticate fields in that response.

User agents are advised to take special care in parsing the field value, as it might contain more than one challenge, and each challenge can contain a comma-separated list of authentication parameters. Furthermore, the header field itself can occur multiple times.

For instance:

```
WWW-Authenticate: Newauth realm="apps", type=1,
                  title="Login to \"apps\"", Basic realm="simple"
```

This header field contains two challenges; one for the "Newauth" scheme with a realm value of "apps", and two additional parameters "type" and "title", and another one for the "Basic" scheme with a realm value of "simple".

> **Note:** The challenge grammar production uses the list syntax as well. Therefore, a sequence of comma, whitespace, and comma can be considered either as applying to the preceding challenge, or to be an empty entry in the list of challenges. In practice, this ambiguity does not affect the semantics of the header field value and thus is harmless.

## 4.2. Authorization

The "Authorization" header field allows a user agent to authenticate itself with an origin server — usually, but not necessarily, after receiving a 401 (Unauthorized) response. Its value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
Authorization = credentials
```

If a request is authenticated and a realm specified, the same credentials are presumed to be valid for all other requests within this realm (assuming that the authentication scheme itself does not require otherwise, such as credentials that vary according to a challenge value or using synchronized clocks).

A proxy forwarding a request MUST NOT modify any Authorization fields in that request. See Section 3.2 of [RFC7234] for details of and requirements pertaining to handling of the Authorization field by HTTP caches.

## 4.3. Proxy-Authenticate

The "Proxy-Authenticate" header field consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the proxy for this effective request URI (Section 5.5 of [RFC7230]). A proxy MUST send at least one Proxy-Authenticate header field in each 407 (Proxy Authentication Required) response that it generates.

```
Proxy-Authenticate = 1#challenge
```

Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the next outbound client on the response chain. This is because only the client that chose a given proxy is likely to have the credentials necessary for authentication. However, when multiple proxies are used within the same administrative domain, such as office and regional caching proxies within a large corporate network, it is common for credentials to be generated by the user agent and passed through the hierarchy until consumed. Hence, in such a configuration, it will appear as if Proxy-Authenticate is being forwarded because each proxy will send the same challenge set.

Note that the parsing considerations for WWW-Authenticate apply to this header field as well; see Section 4.1 for details.

## 4.4. Proxy-Authorization

The "Proxy-Authorization" header field allows the client to identify itself (or its user) to a proxy that requires authentication. Its value consists of credentials containing the authentication information of the client for the proxy and/or realm of the resource being requested.

```
Proxy-Authorization = credentials
```

Unlike Authorization, the Proxy-Authorization header field applies only to the next inbound proxy that demanded authentication using the Proxy-Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first inbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

# 5. IANA Considerations

## 5.1. Authentication Scheme Registry

The "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" defines the namespace for the authentication schemes in challenges and credentials. It has been created and is now maintained at <http://www.iana.org/assignments/http-authschemes>.

### 5.1.1. Procedure

Registrations MUST include the following fields:

- Authentication Scheme Name
- Pointer to specification text
- Notes (optional)

Values to be added to this namespace require IETF Review (see [RFC5226], Section 4.1).

### 5.1.2. Considerations for New Authentication Schemes

There are certain aspects of the HTTP Authentication Framework that put constraints on how new authentication schemes can work:

- HTTP authentication is presumed to be stateless: all of the information necessary to authenticate a request MUST be provided in the request, rather than be dependent on the server remembering prior requests. Authentication based on, or bound to, the underlying connection is outside the scope of this specification and inherently flawed unless steps are taken to ensure that the connection cannot be used by any party other than the authenticated user (see Section 2.3 of [RFC7230]).
- The authentication parameter "realm" is reserved for defining protection spaces as described in Section 2.2. New schemes MUST NOT use it in a way incompatible with that definition.
- The "token68" notation was introduced for compatibility with existing authentication schemes and can only be used once per challenge or credential. Thus, new schemes ought to use the auth-param syntax instead, because otherwise future extensions will be impossible.
- The parsing of challenges and credentials is defined by this specification and cannot be modified by new authentication schemes. When the auth-param syntax is used, all parameters ought to support both token and quoted-string syntax, and syntactical constraints ought to be defined on the field value after parsing (i.e., quoted-string processing). This is necessary so that recipients can use a generic parser that applies to all authentication schemes.

  **Note:** The fact that the value syntax for the "realm" parameter is restricted to quoted-string was a bad design choice not to be repeated for new parameters.
- Definitions of new schemes ought to define the treatment of unknown extension parameters. In general, a "must-ignore" rule is preferable to a "must-understand" rule, because otherwise it will be hard to introduce new parameters in the presence of legacy recipients. Furthermore, it's good to describe the policy for defining new parameters (such as "update the specification" or "use this registry").
- Authentication schemes need to document whether they are usable in origin-server authentication (i.e., using WWW-Authenticate), and/or proxy authentication (i.e., using Proxy-Authenticate).
- The credentials carried in an Authorization header field are specific to the user agent and, therefore, have the same effect on HTTP caches as the "private" Cache-Control response directive (Section 5.2.2.6 of [RFC7234]), within the scope of the request in which they appear.

  Therefore, new authentication schemes that choose not to carry credentials in the Authorization header field (e.g., using a newly defined header field) will need to explicitly disallow caching, by mandating the use of either Cache-Control request directives (e.g., "no-store", Section 5.2.1.5 of [RFC7234]) or response directives (e.g., "private").

## 5.2. Status Code Registration

The "Hypertext Transfer Protocol (HTTP) Status Code Registry" located at <http://www.iana.org/assignments/http-status-codes> has been updated with the registrations below:

| Value | Description | Reference |
|---|---|---|
| 401 | Unauthorized | Section 3.1 |
| 407 | Proxy Authentication Required | Section 3.2 |

## 5.3. Header Field Registration

HTTP header fields are registered within the "Message Headers" registry maintained at <http://www.iana.org/assignments/message-headers/>.

This document defines the following HTTP header fields, so the "Permanent Message Header Field Names" registry has been updated accordingly (see [BCP90]).

| Header Field Name | Protocol | Status | Reference |
|---|---|---|---|
| Authorization | http | standard | Section 4.2 |
| Proxy-Authenticate | http | standard | Section 4.3 |
| Proxy-Authorization | http | standard | Section 4.4 |
| WWW-Authenticate | http | standard | Section 4.1 |

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

# 6. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns specific to HTTP authentication. More general security considerations are addressed in HTTP messaging [RFC7230] and semantics [RFC7231].

Everything about the topic of HTTP authentication is a security consideration, so the list of considerations below is not exhaustive. Furthermore, it is limited to security considerations regarding the authentication framework, in general, rather than discussing all of the potential considerations for specific authentication schemes (which ought to be documented in the specifications that define those schemes). Various organizations maintain topical information and links to current research on Web application security (e.g., [OWASP]), including common pitfalls for implementing and using the authentication schemes found in practice.

## 6.1. Confidentiality of Credentials

The HTTP authentication framework does not define a single mechanism for maintaining the confidentiality of credentials; instead, each authentication scheme defines how the credentials are encoded prior to transmission. While this provides flexibility for the development of future authentication schemes, it is inadequate for the protection of existing schemes that provide no confidentiality on their own, or that do not sufficiently protect against replay attacks. Furthermore, if the server expects credentials that are specific to each individual user, the exchange of those credentials will have the effect of identifying that user even if the content within credentials remains confidential.

HTTP depends on the security properties of the underlying transport- or session-level connection to provide confidential transmission of header fields. In other words, if a server limits access to authenticated users using this framework, the server needs to ensure that the connection is properly secured in accordance with the nature of the authentication scheme used. For example, services that depend on individual user authentication often require a connection to be secured with TLS ("Transport Layer Security", [RFC5246]) prior to exchanging any credentials.

## 6.2. Authentication Credentials and Idle Clients

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP does not provide a mechanism for the origin server to direct clients to discard these cached credentials, since the protocol has no awareness of how credentials are obtained or managed by the user agent. The mechanisms for expiring or revoking credentials can be specified as part of an authentication scheme definition.

Circumstances under which credential caching can interfere with the application's security model include but are not limited to:

- Clients that have been idle for an extended period, following which the server might wish to cause the client to re-prompt the user for credentials.
- Applications that include a session termination indication (such as a "logout" or "commit" button on a page) after which the server side of the application "knows" that there is no further reason for the client to retain the credentials.

User agents that cache credentials are encouraged to provide a readily accessible mechanism for discarding cached credentials under user control.

## 6.3. Protection Spaces

Authentication schemes that solely rely on the "realm" mechanism for establishing a protection space will expose credentials to all resources on an origin server. Clients that have successfully made authenticated requests with a resource can use the same authentication credentials for other resources on the same origin server. This makes it possible for a different resource to harvest authentication credentials for other resources.

This is of particular concern when an origin server hosts resources for multiple parties under the same canonical root URI (Section 2.2). Possible mitigation strategies include restricting direct access to

authentication credentials (i.e., not making the content of the Authorization request header field available), and separating protection spaces by using a different host name (or port number) for each party.

# 7. Acknowledgments

This specification takes over the definition of the HTTP Authentication Framework, previously defined in RFC 2617. We thank John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart for their work on that specification. See Section 6 of [RFC2617] for further acknowledgements.

See Section 10 of [RFC7230] for the Acknowledgments related to this document revision.

# 8. References

## 8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.

## 8.2. Informative References

[BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[OWASP] van der Stock, A., Ed., "A Guide to Building Secure Web Applications and Web Services", The Open Web Application Security Project (OWASP) 2.0.1, July 2005, <https://www.owasp.org/>.

[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, June 1999.

[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, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

# Appendix A. Changes from RFCs 2616 and 2617

The framework for HTTP Authentication is now defined by this document, rather than RFC 2617.

The "realm" parameter is no longer always required on challenges; consequently, the ABNF allows challenges without any auth parameters. (Section 2)

The "token68" alternative to auth-param lists has been added for consistency with legacy authentication schemes such as "Basic". (Section 2)

This specification introduces the Authentication Scheme Registry, along with considerations for new authentication schemes. (Section 5.1)

# Appendix B. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII character).

The rules below are defined in [RFC7230]:

```
BWS           = <BWS, see [RFC7230], Section 3.2.3>
OWS           = <OWS, see [RFC7230], Section 3.2.3>
quoted-string = <quoted-string, see [RFC7230], Section 3.2.6>
token         = <token, see [RFC7230], Section 3.2.6>
```

## Appendix C. Collected ABNF

In the collected ABNF below, list rules are expanded as per Section 1.2 of [RFC7230].

```
Authorization = credentials

BWS = <BWS, see [RFC7230], Section 3.2.3>

OWS = <OWS, see [RFC7230], Section 3.2.3>

Proxy-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS
 challenge ] )
Proxy-Authorization = credentials

WWW-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS challenge
 ] )

auth-param = token BWS "=" BWS ( token / quoted-string )
auth-scheme = token

challenge = auth-scheme [ 1*SP ( token68 / [ ( "," / auth-param ) *(
 OWS "," [ OWS auth-param ] ) ] ) ]
credentials = auth-scheme [ 1*SP ( token68 / [ ( "," / auth-param )
 *( OWS "," [ OWS auth-param ] ) ] ) ]

quoted-string = <quoted-string, see [RFC7230], Section 3.2.6>

token = <token, see [RFC7230], Section 3.2.6>
token68 = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" / "+" / "/" )
 *"="
```

# Index

## Authors' Addresses

**Roy T. Fielding** (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA
Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

**Julian F. Reschke** (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany
Email: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/