HTTP Authentication: Token Access Authentication

draft-hammer-http-token-auth-01

Abstract

This document specifies the HTTP Token Access Authentication scheme.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 7, 2010.

Copyright Notice

Copyright (c) 2010 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 (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
Table of Contents

1. Introduction .................................................. 4
   1.1. Terminology ........................................... 5
   1.2. Example ............................................... 5
   1.3. Notational Conventions ................................. 6
2. Making Requests ................................................ 7
3. Verifying Requests ............................................ 8
4. The WWW-Authenticate Response Header .......................... 8
   4.1. The ‘realm’ Attribute ................................ 9
   4.2. The ‘coverage’ Attribute ............................... 9
   4.3. The ‘timestamp’ Attribute .............................. 9
5. The Authorization Request Header ............................ 10
   5.1. The ‘token’ Attribute ................................ 10
   5.2. The ‘coverage’ Attribute ............................... 10
   5.3. The ‘nonce’ Attribute ................................. 10
   5.4. The ‘timestamp’ Attribute ............................. 10
   5.5. The ‘auth’ Attribute .................................. 11
6. The Authentication-Error Response Header ................... 11
   6.1. The ‘error-code’ attribute ............................. 11
   6.2. The ‘error-info’ attribute .............................. 11
   6.3. The ‘error-message’ attribute ........................ 11
7. Authentication Methods ......................................... 11
   7.1. The ‘none’ Method ..................................... 12
   7.2. The ‘hmac-sha-1’ Method ................................ 12
   7.3. The ‘hmac-sha-256’ Method ............................. 13
   7.4. The ‘rsassa-pkcs1-v1.5-sha-256’ Method ............... 13
8. Coverage Methods ................................................ 14
   8.1. The ‘base’ Method ...................................... 14
      8.1.1. String Construction ................................ 15
   8.2. The ‘base+body-hmac-sha-256’ Method .................. 15
9. Scheme Extensions ............................................... 16
10. Security Considerations .................................... 16
    10.1. Credentials Transmission ............................. 16
    10.2. Confidentiality of Requests .......................... 16
    10.3. Spoofing by Counterfeit Servers ...................... 16
    10.4. Plaintext Storage of Credentials ..................... 17
    10.5. Scoping of Access Requests ........................... 17
    10.6. Entropy of Secrets ................................... 17
    10.7. Denial of Service / Resource Exhaustion Attacks .... 18
    10.8. Coverage Limitations ................................ 18
11. IANA Considerations .......................................... 18
12. Acknowledgments .............................................. 19

Appendix A. Document History .................................. 19
13. References ......................................................... 19
13.1. Normative References ......................................... 19
13.2. Informative References ........................................ 20
Author’s Address ....................................................... 20
1. Introduction

With the growing use of distributed web services and cloud computing, clients need to allow other parties access to the resources they control. When granting access, clients should not be required to share their credentials (typically a username and password). Clients should also have the ability to restrict access to a limited subset of the resources they control or limit access to the methods supported by these resources. These goals require new classes of authentication credentials.

The HTTP Basic and Digest Access authentication schemes defined by [RFC2617], enable clients to make authenticated HTTP requests by using a username (or userid) and a password. In most cases, the client uses a single set of credentials to access all the resources it controls which are hosted by the server.

While the Basic and Digest schemes can be used to send credentials other than a username and password, their wide deployment and well-established behavior in user-agents preclude them from being used with other classes of credentials. Extending these schemes to support new classes would require an impractical change to their existing deployment.

The Token Access Authentication scheme provides a method for making authenticated HTTP requests using a token - an identifier used to denote an access grant with specific scope, duration, cryptographic properties, and other attributes. Tokens can issued by the server, self-issued by the client, or issued by a third-party.

The token scheme supports an extensible set of credential classes, authentication methods (e.g. cryptographic algorithm), and authentication coverage (the elements of the HTTP request - such as the request URI or entity-body - covered by the authentication).

This specification defines four token authentication methods to support the most common use cases and describes their security properties. The methods through which clients obtain tokens supporting these methods are beyond the scope of this specification. The OAuth protocol [I-D.ietf-oauth-web-delegation] defines one such set of methods for obtaining token credentials.
1.1. Terminology

client
An HTTP client (per [RFC2616]) capable of making Token-
authenticated requests (Section 2).

server
An HTTP server (per [RFC2616]) capable of accepting Token-
authenticated requests (Section 2).

protected resource
An access-restricted resource (per [RFC2616]) hosted by the
server and accessible by making a Token-authenticated request
(Section 2).

token credentials
A set of a unique identifier (token) and an authentication
method with an OPTIONAL shared secret (symmetric or
asymmetric), as well as other attributes (e.g. duration,
scope), used by the client to make authenticated requests.

normalized request string
A string representing various elements of the HTTP request,
normalized and concatenated together. The elements included in
the normalize request string are determined by the
authentication coverage supported by the server.

1.2. Example

The following HTTP request:

GET /resource/1 HTTP/1.1
Host: example.com

returns the following authentication challenge:

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Token realm="http://example.com/",
coverage="base base+body-sha-256",
timestamp="137131190"

The response means the server is expecting the client to authenticate
using the token scheme, with a set of token credentials issued for
the "http://example.com/" realm. The server supports the "base" and
"base+body-sha-256" coverage methods which means the client must sign
the base request components (e.g. host, port, request URI), and may
also sign the request payload (entity-body). It also provides its current time to assist the client in synchronizing its clock with the server’s clock for the purpose of producing a unique nonce value (used with some of the authentication methods).

The client has previously obtained a set of token credentials for accessing resources in the "http://example.com/" realm. The credentials issued to the client by the server included the following attributes:

token: h480djs93hd8
method: hmac-sha-1
secret: 489dks293j39
expiration: 137217600

The client attempts the HTTP request again, this time using the token credentials issued by the server earlier to authenticate. The client uses the "base" coverage method and applies the "hmac-sha-1" authentication method as dictated by the token credentials.

GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Token token="h480djs93hd8",
coverage="base",
timestamp="137131200",
nonce="dj83hs9s",
auth="djosJKDKJSD8743243/jdk33klY="

to which the server respond with the requested resource representation after validating the request.

1.3. Notational Conventions

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].

This document uses the Augmented Backus-Naur Form (ABNF) notation of [I-D.ietf-httpbis-p1-messaging]. Additionally, the following rules are included from [RFC2617]: realm, auth-param.
2. Making Requests

The client makes authenticated requests by calculating the values of a set of attributes and adding them to the HTTP request using the Authorization header field (Section 5). Authenticated request can be sent either directly (without first receiving a challenge), or in response to an authentication challenge.

To make an authenticated request, the client obtains information about the attributes supported by the server. This information is provided by the server via the WWW-Authenticate header field (Section 4). The client SHOULD only send an authenticated request to the server (without first receiving a challenge) if it has prior knowledge of the attributes supported by server.

The client chooses an available token suitable for accessing the resource realm as well as a supported authentication coverage. The methods through which the client obtains a valid token, or the criteria used to choose a token if more than one is available are beyond the scope of this specification.

Once the client selects the appropriate token credentials it proceeds to:

1. Assign values based on its selection to the following attributes:
   * "token"
   * "coverage"

2. If the client uses a coverage method other than "none" it MUST assign values to the following attributes:
   * "nonce"
   * "timestamp"

3. Assigns value to any additional method-specific, or coverage-specific attributes as defined by protocol extensions.

4. If the client uses a coverage method other than "none" it constructs the normalized request string based on the selected coverage as described in Section 8.

5. Calculates the value of the "auth" attribute as defined by the selected authentication method.
6. Adds the assigned attributes to the request via the Authorization header field (Section 5).

7. Sends the authenticated HTTP request to the server.

3. Verifying Requests

A server receiving an authenticated request validates it by performing the following REQUIRED steps:

1. Verify that the token used by the client as well as the coverage method matches the server’s requirements.

2. If the client used a coverage method other than "none", construct the normalized request string based on the selected coverage as described in Section 8.

3. If the client used an authentication method other than "none", recalculate the value of the "auth" attribute as described in Section 7 and compare it to the value received from the client via the "auth" attribute.

4. If the client used a coverage method other than "none", ensure that the combination of nonce, timestamp, and token received from the client has not been used before in a previous request (the server MAY reject requests with stale timestamps; the determination of staleness is left up to the server to define).

5. Verify the scope and status of the client credentials as represented by the token.

If the request fails verification, the server SHOULD respond with an HTTP 401 (unauthorized) status code, and SHOULD include a token scheme authentication challenge using the WWW-Authenticate header field (Section 6). The server MAY include further details about why the request was rejected using the Authorization-Error header field (Section 6).

4. The WWW-Authenticate Response Header

A server receiving a request for a protected resource without a valid Authorization header field (Section 5) MUST respond with a 401 status code (Unauthorized), and includes at least one "WWW-Authenticate" header field including a token scheme challenge.

The "WWW-Authenticate" header field uses the framework defined by
[RFC2617] as follows:

```plaintext
challenge       = "Token" RWS token-challenge

token-challenge = realm
    CS coverage-list
    [ CS timestamp ]

coverage-list   = "coverage" "=" <">
coverage-name   = "none" / "base" / "base+body-sha-256" /
    token

timestamp       = "timestamp" "=" <">

CS              = OWF "," OWF
```

4.1. The ‘realm’ Attribute

4.2. The ‘coverage’ Attribute

The list of authentication coverage names supported by the server, provided as a space-delimited list. If omitted, the attribute defaults to "base". Authentication coverage is described in Section 8.

4.3. The ‘timestamp’ Attribute

Signature-based and hash-based authentication methods use timestamps in combination with unique nonce values to protect against replay attacks when used over an unsecure channel.

The timestamp attribute is used by the server to publish its current time, enabling clients to synchronize their close with the server. The timestamp value is the current time expressed in the number of seconds since January 1, 1970 00:00:00 GMT, and MUST be a positive integer.

To avoid the need to retain an infinite number of nonce values for future checks, servers MAY choose to restrict the time period after which a request with an old timestamp is rejected. Servers applying such a restriction SHOULD provide their current time to the client either in every challenge or when a request fails due to a timestamp outside the allowed window.
5. The Authorization Request Header

A client making a request for a protected resource either directly, or in retrying a request after receiving a 401 status code (Unauthorized) with a token challenge, MUST include at least one "Authorization" header field including token scheme credentials.

The "Authorization" header field uses the framework defined by [RFC2617] as follows:

```
credentials    = "Token" RWS token-response

token-response = token-id
                   CS coverage
                     [ CS nonce ]
                     [ CS timestamp ]
                     [ CS auth ]

  token-id         = "token" "=" <"> token <">
  coverage         = "coverage" "=" <"> coverage-name <">
  nonce            = "nonce" "=" <"> token <">
  auth             = "auth" "=" <"> token <">
```

5.1. The 'token' Attribute

The value used to identify the set of token credentials used by the client to authenticate. The token identifier can be an opaque string or use a well-defined internal structure.

5.2. The 'coverage' Attribute

The name of the authentication coverage method used by the client to make the request. If the attribute is omitted, its value defaults to "base".

5.3. The 'nonce' Attribute

A random string, uniquely generated by the client to allow the server to verify that a request has never been made before and helps prevent replay attacks when requests are made over a non-secure channel. The nonce value MUST be unique across all requests with the same timestamp and token combinations.

5.4. The 'timestamp' Attribute

The timestamp value is the current time expressed in the number of seconds since January 1, 1970 00:00:00 GMT, and MUST be a positive
5.5. The 'auth' Attribute

The output of the authentication method function after applying it to the selected coverage as described in Section 7.

6. The Authentication-Error Response Header

A server receiving a request for a protected resource with an invalid Authorization header field (Section 5) MAY includes the "Authentication-Error" header field providing the client with information to help it successfully authenticate with the server.

The "Authentication-Error" header field is defined as follows:

```
Authentication-Error = "Authentication-Error" ";"
OWS #1error-param

error-param = error-code /
error-info /
error-message /
auth-param

error-code = "error-code" "=" token
error-info = "error-info" "=" token
error-message = "error-message" "=" quoted-string
```

6.1. The 'error-code' attribute

6.2. The 'error-info' attribute

6.3. The 'error-message' attribute

7. Authentication Methods

In order for the server to verify the authenticity of the request and prevent unauthorized access, the client must prove it is the rightful owner of the credentials. This is accomplished using the authentication method associated with the token.

This specification provides three methods for the client to prove its rightful ownership of the credentials: "hmac-sha-1", "hmac-sha-256", and "rsassa-pkcs1-v1.5-sha-256". In addition, the "none" method is defined to allow the use of bearer token which does not utilizes any
cryptographic means.

```
method-name = "none" / "hmac-sha-1" / "hmac-sha-256" / "rsassa-pkcs1-v1.5-sha-256" /
```

token

The authentication process does not change the request or its parameters, with the exception of the "auth" attribute.

### 7.1. The 'none' Method

The "none" method does not employ a cryptographic algorithm and does not provide any security on its own. Servers utilizing this method use the token identifier as a bearer token, relying solely on the value of the token identifier to authenticate the client.

The "nonce", "timestamp", and "auth" attributes are not used, and SHOULD NOT be included in authenticated requests. The "coverage" attribute MUST be set to "none" but MAY be omitted from the request. Nevertheless, these attributes MUST be included in the normalized request string together with any other authentication attributes.

### 7.2. The 'hmac-sha-1' Method

The "hmac-sha-1" authentication method uses the HMAC-SHA1 algorithm as defined in [RFC2104]:

\[
\text{digest} = \text{HMAC-SHA1} \text{ (key, text)}
\]

The HMAC-SHA1 function variables are used in following way:

- **text** is set to the value of the normalize request string as described in Section 8.
- **key** is set to the shared-secret associated with the token.
- **digest** is used to set the value of the "auth" attribute, after the result octet string is base64-encoded per [RFC2045] section 6.8.
7.3. The 'hmac-sha-256' Method

The "hmac-sha-256" authentication method uses the HMAC algorithm as defined in [RFC2104] together with the SHA-256 hash function defined in [NIST FIPS-180-3]:

$$\text{digest} = \text{HMAC-SHA2-256} \ (\text{key}, \ \text{text})$$

The HMAC-SHA256 function variables are used in following way:

- **text** is set to the value of the normalize request string as described in Section 8.
- **key** is set to the shared-secret associated with the token.
- **digest** is used to set the value of the "auth" attribute, after the result octet string is base64-encoded per [RFC2045] section 6.8.

7.4. The ‘rsassa-pkcs1-v1.5-sha-256’ Method

The "rsassa-pkcs1-v1.5-sha-256" signature method uses the RSASSA-PKCS1-v1_5 signature algorithm as defined in [RFC3447] section 8.2 (also known as PKCS#1), using SHA-256 as the hash function as defined in [NIST FIPS-180-3] for EMSA-PKCS1-v1_5.

The normalized request string is signed using the RSA private key associated with the token as defined in [RFC3447] section 8.2.1:

$$S = \text{RSASSA-PKCS1-V1_5-SIGN} \ (K, \ M)$$

Where:

- **K** is set to the RSA private key associated with the token,
- **M** is set to the value of the normalized request string described in Section 8, and
S is the result signature used to set the value of the "auth" attribute, after the result octet string is base64-encoded per [RFC2045] section 6.8.

The server verifies the signature per [RFC3447] section 8.2.2:

RSASSA-PKCS1-V1_5-VERIFY ((n, e), M, S)

Where:

(n, e)
is set to the RSA public key associated with the token,

M is set to the value of the normalized request string described in Section 8, and

S is set to the octet string value of the "auth" attribute received from the client.

8. Coverage Methods

The normalized request string is a consistent, reproducible concatenation of several of the HTTP request elements into a single string. The string is used as an input to the authentication methods with the exception of "none".

8.1. The 'base' Method

When using the "base" method, the normalized request string includes the following components of the HTTP request:

- The HTTP request method (e.g. "GET", "POST", etc.).
- The authority as declared by the HTTP "Host" request header.
- The request resource URI.
- The Authorization header field (Section 5) attributes, with the exception of the "auth" attribute.

The "base" normalized request string does not cover the entire HTTP request. Most notably, it does not include the entity-body or most HTTP entity-headers. It is important to note that the server cannot
verify the authenticity of the excluded request elements without using additional protections such as SSL/TLS or other methods.

8.1.1. String Construction

The normalized request string is constructed by concatenating together, in order, the following HTTP request elements:

1. The HTTP request method in uppercase. For example: "HEAD", "GET", "POST", etc.

2. A "," character (ASCII code 44).

3. The hostname, colon-separated (ASCII code 58) from the TCP port used to make the request as included in the HTTP request "Host" header field. The port MUST be included even if it is not included in the "Host" header field (i.e. the default port for the scheme).

4. A "," character (ASCII code 44).

5. Any authentication attribute, with the exception of the "auth", which is assigned a value (including default values), are added to the normalized request string as follows:

   1. The name of each parameter is concatenated to its corresponding value using an "=" character (ASCII code 61) as separator, even if the value is empty.

   2. The name/value pairs are sorted using ascending byte value ordering.

   3. The sorted name/value pairs are concatenated together into a single string by using a "," character (ASCII code 44) as separator.

6. A "," character (ASCII code 44).

7. The request resource URI.

8.2. The 'base+body-hmac-sha-256' Method

The "base+body-hmac-sha-256" method added the request entity-body to the elements included in the normalized request string. It does not include the entity-body directly in the normalized string. Instead, it calculates the hash value of the entity-body using the SHA-256 hash function defined in [NIST FIPS-180-3].
The normalized request string is constructed following the same process defined in Section 8.1.1, with the following addition:

- Before constructing the string, the entity-body hash is calculated by applying the SHA-256 hash function on the raw entity-body content.
- The hash value is added to the list of authentication attributes by assigning its value to the "body-hash" attribute name. This is done prior to the attributes being sorted and added to the string.
- The "body-hash" attribute is only included in the normalized request string and is not added to the Authorization header field (Section 5).

9. Scheme Extensions

10. Security Considerations

As stated in [RFC2617], the greatest sources of risks are usually found not in the core protocol itself but in policies and procedures surrounding its use. Implementers are strongly encouraged to assess how this protocol addresses their security requirements.

10.1. Credentials Transmission

This specification does not describe any mechanism for obtaining or transmitting raw tokens credentials. Methods used to obtain tokens should ensure that these transmissions are protected using transport-layer mechanisms such as TLS or SSL.

10.2. Confidentiality of Requests

While this protocol provides a mechanism for verifying the integrity of requests, it provides no guarantee of request confidentiality. Unless further precautions are taken, eavesdroppers will have full access to request content. Servers should carefully consider the kinds of data likely to be sent as part of such requests, and should employ transport-layer security mechanisms to protect sensitive resources.

10.3. Spoofing by Counterfeit Servers

This protocol makes no attempt to verify the authenticity of the server. A hostile party could take advantage of this by intercepting the client’s requests and returning misleading or otherwise incorrect
responses. Service providers should consider such attacks when developing services using this protocol, and should require transport-layer security for any requests where the authenticity of the server or of request responses is an issue.

10.4. Plaintext Storage of Credentials

When used with a symmetric shared-secret authentication method, the token shared-secret function the same way passwords do in traditional authentication systems. In order to compute the signatures used in methods, the server must have access to these secrets in plaintext form. This is in contrast, for example, to modern operating systems, which store only a one-way hash of user credentials.

If an attacker were to gain access to these secrets - or worse, to the server’s database of all such secrets - he or she would be able to perform any action on behalf of any resource owner. Accordingly, it is critical that servers protect these secrets from unauthorized access.

10.5. Scoping of Access Requests

By itself, this protocol does not provide any method for scoping the access rights granted to a client. However, most applications do require greater granularity of access rights. For example, servers may wish to make it possible to grant access to some protected resources but not others, or to grant only limited access (such as read-only access) to those protected resources.

When implementing this protocol, servers should consider the types of access resource owners may wish to grant clients, and should provide mechanisms to do so. Servers should also take care to ensure that resource owners understand the access they are granting, as well as any risks that may be involved.

10.6. Entropy of Secrets

Unless a transport-layer security protocol is used, eavesdroppers will have full access to authenticated requests and signatures, and will thus be able to mount offline brute-force attacks to recover the credentials used. Servers should be careful to assign shared-secrets which are long enough, and random enough, to resist such attacks for at least the length of time that the shared-secrets are valid.

For example, if shared-secrets are valid for two weeks, servers should ensure that it is not possible to mount a brute force attack that recovers the shared-secret in less than two weeks. Of course, servers are urged to err on the side of caution, and use the longest
secrets reasonable.

It is equally important that the pseudo-random number generator (PRNG) used to generate these secrets be of sufficiently high quality. Many PRNG implementations generate number sequences that may appear to be random, but which nevertheless exhibit patterns or other weaknesses which make cryptanalysis or brute force attacks easier. Implementers should be careful to use cryptographically secure PRNGs to avoid these problems.

10.7. Denial of Service / Resource Exhaustion Attacks

This specification includes a number of features which may make resource exhaustion attacks against servers possible. For example, this protocol requires servers to track used nonces. If an attacker is able to use many nonces quickly, the resources required to track them may exhaust available capacity. And again, this protocol can require servers to perform potentially expensive computations in order to verify the signature on incoming requests. An attacker may exploit this to perform a denial of service attack by sending a large number of invalid requests to the server.

Resource Exhaustion attacks are by no means specific to this specification. However, implementers should be careful to consider the additional avenues of attack that this protocol exposes, and design their implementations accordingly. For example, entropy starvation typically results in either a complete denial of service while the system waits for new entropy or else in weak (easily guessable) secrets. When implementing this protocol, servers should consider which of these presents a more serious risk for their application and design accordingly.

10.8. Coverage Limitations

The normalized request string has been designed to support the authentication methods defined in this specification. Those designing additional methods, should evaluated the compatibility of the normalized request string with their security requirements. Since the normalized request string does not cover the entire HTTP request, servers should employ additional mechanisms to protect such elements.

11. IANA Considerations
12. Acknowledgments

The author would like to thank Richard Barnes, Breno de Medeiros, Brian Eaton, Ben Laurie, Mark Nottingham, John Panzer, and Peter Saint-Andre for their suggestions, feedback, and continued support.

Appendix A. Document History

[[ To be removed by the RFC editor before publication as an RFC. ]]

-01

o Simplified challenge, moving the supported authentication methods to the token definition, as well as using the ‘realm’ parameter as defined by RFC 2617 instead of the ‘class’ parameter (which has been dropped).

-00

o Initial (incomplete) draft.

13. References

13.1. Normative References

[I-D.ietf-httpbis-p1-messaging]

[NIST FIPS-180-3]


informative references

[I-D.ietf-oauth-web-delegation]
draft-ietf-oauth-web-delegation-01 (work in progress),
July 2009.

URIs


Author’s Address

Eran Hammer-Lahav
Yahoo!

Email: eran@hueniverse.com
URI: http://hueniverse.com