HTTPAuth
Internet-Draft
Obsoletes: 2617 (if approved)
Intended status: Standards Track
Expires: July 24, 2015

R. Shekh-Yusef, Ed.
Avaya
D. Ahrens
Independent
S. Bremer
Netzkonform
January 20, 2015

HTTP Digest Access Authentication
draft-ietf-httpauth-digest-11

Abstract

HTTP provides a simple challenge-response authentication mechanism that may be used by a server to challenge a client request and by a client to provide authentication information. This document defines the HTTP Digest Authentication scheme that can be used with the HTTP authentication mechanism.

Editorial Note (To be removed by RFC Editor before publication)

Discussion of this draft takes place on the HTTPAuth working group mailing list (http-auth@ietf.org), which is archived at [1].

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 http://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 July 24, 2015.

Copyright Notice

Copyright (c) 2015 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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 3
1.1. Terminology 3
2. Syntax Convention 4
2.1. Examples 4
2.2. ABNF 4
3. Digest Access Authentication Scheme 4
3.1. Overall Operation 4
3.2. Representation of Digest Values 4
3.3. The WWW-Authenticate Response Header 5
3.4. The Authorization Request Header Field 8
3.4.1. Response 10
3.4.2. A1 10
3.4.3. A2 11
3.4.4. Username Hashing 11
3.4.5. Parameter Values and Quoted-String 11
3.4.6. Various Considerations 12
3.5. The Authentication-Info Header 13
3.5.1. Digest Usage of Authentication-Info 13
3.6. Digest Operation 15
3.7. Security Protocol Negotiation 16
3.8. Proxy-Authenticate and Proxy-Authorization 16
3.9. Examples 17
3.9.1. Example with SHA-256 and MD5 17
3.9.2. Example with SHA-512-256, Charset, and Userhash . . . 19

4. Internationalization 20
5. Security Considerations 20
5.1. Limitations 20
5.2. Storing passwords 21
5.3. Authentication of Clients using Digest Authentication . . 21
5.4. Limited Use Nonce Values 22
5.5. Replay Attacks 23
5.6. Weakness Created by Multiple Authentication Schemes . . . 23
5.7. Online dictionary attacks 24
5.8. Man in the Middle 24
5.9. Chosen plaintext attacks 25
5.10. Precomputed dictionary attacks 25
5.11. Batch brute force attacks 25
5.12. Summary 26
6. IANA Considerations 26
6.1. HTTP Digest Hash Algorithms Registry 26
6.2. Digest Scheme Registration 27
6.3. Authentication-Info Header Registration 27
7. Acknowledgments 27
8. References 28
8.1. Normative References 28
8.2. Informative References 29

1. Introduction

HTTP provides a simple challenge-response authentication mechanism that may be used by a server to challenge a client request and by a client to provide authentication information. This document defines the HTTP Digest Authentication scheme that can be used with the HTTP authentication mechanism.

The details of the challenge-response authentication mechanism are specified in the "Hypertext Transfer Protocol (HTTP/1.1): Authentication" [RFC7235].

The combination of this document with the definition of the "Basic" authentication scheme [BASIC] and [RFC7235] obsolete [RFC2617].

1.1. Terminology

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].

2. Syntax Convention

2.1. Examples

In the interest of clarity and readability, the extended parameters or the header fields and parameters in the examples in this document might be broken into multiple lines. Any line that is indented in this document is a continuation of the preceding line.

2.2. ABNF

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], and the ABNF List Extension of [RFC7230].

3. Digest Access Authentication Scheme

3.1. Overall Operation

The Digest scheme is based on a simple challenge-response paradigm. The Digest scheme challenges using a nonce value, and might indicate that username hashing is supported. A valid response contains a checksum of the username, the password, the given nonce value, the HTTP method, and the requested URI. In this way, the password is never sent in the clear, and the username can be hashed, depending on the indication received from the server. The username and password must be prearranged in some fashion not addressed by this document.

The security of this protocol is critically dependent on the randomness of the randomly chosen parameters, such as client and server nonces. These should be generated by a strong random or properly seeded pseudorandom source (see [RFC4086]).

For the following parameters, the value can be encoded using [RFC5987] encoding: username.

3.2. Representation of Digest Values

An optional header field allows the server to specify the algorithm used to create the checksum or digest. This documents adds SHA-256 and SHA-512/256 algorithms. To maintain backwards compatibility with [RFC2617], the MD5 algorithm is still supported but NOT RECOMMENDED.

The size of the digest depends on the algorithm used. The bits in the digest are converted from the most significant to the least significant bit, four bits at a time to the ASCII representation as follows. Each four bits is represented by its familiar hexadecimal notation from the characters 0123456789abcdef, that is binary 0000 is represented by the character '0', 0001 by '1' and so on up to the

representation of 1111 as 'f'. If the MD5 algorithm is used to calculate the digest, then the MD5 digest will be represented as 32 hexadecimal characters, while SHA-256 and SHA-512/256 are represented as 64 hexadecimal characters.

3.3. The WWW-Authenticate Response Header

If a server receives a request for an access-protected object, and an acceptable Authorization header field is not sent, the server responds with a "401 Unauthorized" status code and a WWW-Authenticate header field with Digest scheme as per the framework defined above. The value of the header field can include parameters from the following list:

realm

A string to be displayed to users so they know which username and password to use. This string should contain at least the name of the host performing the authentication and might additionally indicate the collection of users who might have access. An example might be "registered_users@gotham.news.com". (See Section 2.2 of [RFC7235] for more details).

domain

A quoted, space-separated list of URIs, as specified in [RFC3986], that define the protection space. If a URI is an abs_path, it is relative to the canonical root URL (See Section 2.2 of [RFC7235]) of the web-origin [RFC6454]. An absolute-URI in this list may refer to a different server than the web-origin. The client can use this list to determine the set of URIs for which the same authentication information may be sent: any URI that has a URI in this list as a prefix (after both have been made absolute) MAY be assumed to be in the same protection space. If this parameter is omitted or its value is empty, the client SHOULD assume that the protection space consists of all URIs on the web-origin. All URIs in this list SHOULD use the same scheme (https or http); mixing them is a bad idea.

This parameter is not meaningful in Proxy-Authenticate header fields, for which the protection space is always the entire proxy; if present it MUST be ignored.

nonce

A server-specified data string which should be uniquely generated each time a 401 response is made. It is advised that this string be base64 or hexadecimal data. Specifically, since the string is

passed in the header field lines as a quoted string, the double-quote character is not allowed, unless suitably escaped.

The contents of the nonce are implementation dependent. The quality of the implementation depends on a good choice. A nonce might, for example, be constructed as the base 64 encoding of

time-stamp H(time-stamp ":" ETag ":" secret-data)

where time-stamp is a server-generated time, which preferably includes micro or nano seconds, or other non-repeating values, ETag is the value of the HTTP ETag header field associated with the requested entity, and secret-data is data known only to the server. With a nonce of this form a server would recalculate the hash portion after receiving the client authentication header field and reject the request if it did not match the nonce from that header field or if the time-stamp value is not recent enough. In this way the server can limit the time of the nonce's validity. The inclusion of the ETag prevents a replay request for an updated version of the resource. Including the IP address of the client in the nonce would appear to offer the server the ability to limit the reuse of the nonce to the same client that originally got it. However, that would break when requests from a single user often go through different proxies. Also, IP address spoofing is not that hard.

An implementation might choose not to accept a previously used nonce or a previously used digest, in order to protect against a replay attack. Or, an implementation might choose to use one-time nonces or digests for POST or PUT requests and a time-stamp for GET requests. For more details on the issues involved see Section 5 of this document.

The nonce is opaque to the client.

opaque

A string of data, specified by the server, which SHOULD be returned by the client unchanged in the Authorization header field of subsequent requests with URIs in the same protection space. It is RECOMMENDED that this string be base64 or hexadecimal data.

stale

A case-insensitive flag indicating that the previous request from the client was rejected because the nonce value was stale. If stale is TRUE, the client may wish to simply retry the request with a new encrypted response, without re-prompting the user for a

new username and password. The server SHOULD only set stale to TRUE if it receives a request for which the nonce is invalid. If stale is FALSE, or anything other than TRUE, or the stale parameter is not present, the username and/or password are invalid, and new values MUST be obtained.

algorithm

A string indicating a pair of algorithms used to produce the digest and a checksum. If this is not present it is assumed to be "MD5". If the algorithm is not understood, the challenge SHOULD be ignored (and a different one used, if there is more than one).

When used with the Digest mechanism, each one of the algorithms has two variants: Session variant and non-Session variant. The non-Session variant is denoted by "<algorithm>", e.g. "SHA-256", and the Session variant is denoted by "<algorithm>-sess", e.g. "SHA-256-sess".

In this document the string obtained by applying the digest algorithm to the data "data" with secret "secret" will be denoted by KD(secret, data), and the string obtained by applying the checksum algorithm to the data "data" will be denoted H(data). The notation unq(X) means the value of the quoted-string X without the surrounding quotes and with quoting slashes removed.

For "<algorithm>" and "<algorithm>-sess"

H(data) = <algorithm>(data)

and

KD(secret, data) = H(concat(secret, ":", data))

For example:

For the "SHA-256" and "SHA-256-sess" algorithms

H(data) = SHA-256(data)

i.e., the digest is the "<algorithm>" of the secret concatenated with a colon concatenated with the data. The "<algorithm>-sess" algorithm is intended to allow efficient 3rd party authentication servers; for the difference in usage, see the description in section 3.4.2.

qop

This parameter MUST be used by all implementations. It is a quoted string of one or more tokens indicating the "quality of protection" values supported by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection; see the descriptions below for calculating the response parameter value for the application of this choice. Unrecognized options MUST be ignored.

charset

This is an OPTIONAL parameter that is used by the server to indicate the encoding scheme it supports.

userhash

This is an OPTIONAL parameter that is used by the server to indicate that it supports username hashing. Valid values are: "true" or "false". Default value is "false".

For historical reasons, a sender MUST only generate the quoted-string syntax values for the following parameters: realm, domain, nonce, opaque, and qop.

For historical reasons, a sender MUST NOT generate the quoted-string syntax values for the following parameters: stale and algorithm.

3.4. The Authorization Request Header Field

The client is expected to retry the request, passing an Authorization header field line with Digest scheme, which is defined according to the framework above. The values of the opaque and algorithm fields must be those supplied in the WWW-Authenticate response header field for the entity being requested.

The request can include parameters from the following list:

response

A string of the hex digits computed as defined below, which proves that the user knows a password.

username

The user's name in the specified realm.

uri

The Effective Request URI [RFC7230] from request-target of the Request-Line; duplicated here because proxies are allowed to change the Request-Line in transit.

qop

Indicates what "quality of protection" the client has applied to the message. Its value MUST be one of the alternatives the server indicated it supports in the WWW-Authenticate header field. These values affect the computation of the response. Note that this is a single token, not a quoted list of alternatives as in WWW-Authenticate.

cnonce

This parameter MUST be used by all implementations. The cnonce value is an opaque quoted ASCII-only string value provided by the client and used by both client and server to avoid chosen plaintext attacks, to provide mutual authentication, and to provide some message integrity protection. See the descriptions below of the calculation of the rspauth and response values.

nc

This parameter MUST be used by all implementations. The "nc" parameter stands for "nonce count". The nc value is the hexadecimal count of the number of requests (including the current request) that the client has sent with the nonce value in this request. For example, in the first request sent in response to a given nonce value, the client sends "nc=00000001". The purpose of this parameter is to allow the server to detect request replays by maintaining its own copy of this count - if the same nc value is seen twice, then the request is a replay. See the description below of the construction of the response value.

userhash

This OPTIONAL parameter is used by the client to indicate that the username has been hashed. Valid values are: "true" or "false". Default value is "false".

For historical reasons, a sender MUST only generate the quoted-string syntax for the following parameters: username, realm, nonce, uri, response, cnonce, and opaque.

For historical reasons, a sender MUST NOT generate the quoted-string syntax for the following parameters: algorithm, qop, and nc.

If a parameter or its value is improper, or required parameters are missing, the proper response is a 4xx error code. If the response is invalid, then a login failure SHOULD be logged, since repeated login failures from a single client may indicate an attacker attempting to guess passwords. The server implementation SHOULD be careful with the information being logged so that it won't put a cleartext password (e.g. entered into the username field) into the log.

The definition of the response above indicates the encoding for its value. The following definitions show how the value is computed.

3.4.1. Response

If the "qop" value is "auth" or "auth-int":

```
      response = <"> < KD ( H(A1), unq(nonce)
                                   ":" nc
                                   ":" unq(cnonce)
                                   ":" unq(qop)
                                   ":" H(A2)
                           ) <">
```

See below for the definitions for A1 and A2.

3.4.2. A1

If the "algorithm" parameter's value is "<algorithm>", e.g. "SHA-256", then A1 is:

```
      A1       = unq(username) ":" unq(realm) ":" passwd
```

where

```
      passwd   = < user's password >
```

If the "algorithm" parameter's value is "<algorithm>-sess", e.g. "SHA-256-sess", then A1 is calculated using the nonce value provided in the challenge from the server, and cnounce value from the request by the client following receipt of a WWW-Authenticate challenge from the server. It uses the server nonce from that challenge, herein called nonce-prime, and the client nonce value from the response, herein called cnonce-prime, to construct A1 as follows:

```
      A1       = H( unq(username) ":" unq(realm)
                    ":" passwd )
                    ":" unq(nonce-prime) ":" unq(cnonce-prime)
```

This creates a "session key" for the authentication of subsequent requests and responses which is different for each "authentication session", thus limiting the amount of material hashed with any one key. (Note: see further discussion of the authentication session in section 3.6.) Because the server need only use the hash of the user credentials in order to create the A1 value, this construction could be used in conjunction with a third party authentication service so that the web server would not need the actual password value. The specification of such a protocol is beyond the scope of this specification.

3.4.3. A2

If the "qop" parameter's value is "auth" or is unspecified, then A2 is:

```
A2       = Method ":" request-uri
```

If the "qop" value is "auth-int", then A2 is:

```
A2       = Method ":" request-uri ":" H(entity-body)
```

3.4.4. Username Hashing

To protect the transport of the username from the client to the server, the server SHOULD set the "userhash" parameter with the value of "true" in the WWW-Authentication header field.

If the client supports the "userhash" parameter, and the "userhash" parameter value in the WWW-Authentication header field is set to "true", then the client MUST calculate a hash of the username after any other hash calculation and include the "userhash" parameter with the value of "true" in the Authorization Request Header field. If the client does not provide the "username" as a hash value or the "userhash" parameter with the value of "true", the server MAY reject the request.

The following is the operation that the client will take to hash the username, using the same algorithm used to hash the credentials:

```
username = H( unq(username) ":" unq(realm) )
```

3.4.5. Parameter Values and Quoted-String

Note that the value of many of the parameters, such as "username" value, are defined as a "quoted-string". However, the "unq" notation indicates that surrounding quotation marks are removed in forming the

string A1. Thus if the Authorization header field includes the fields

username="Mufasa", realm=myhost@testrealm.com

and the user Mufasa has password "Circle Of Life" then H(A1) would be H(Mufasa:myhost@testrealm.com:Circle Of Life) with no quotation marks in the digested string.

No white space is allowed in any of the strings to which the digest function H() is applied unless that white space exists in the quoted strings or entity body whose contents make up the string to be digested. For example, the string A1 illustrated above must be

Mufasa:myhost@testrealm.com:Circle Of Life

with no white space on either side of the colons, but with the white space between the words used in the password value. Likewise, the other strings digested by H() must not have white space on either side of the colons which delimit their fields unless that white space was in the quoted strings or entity body being digested.

Also note that if integrity protection is applied (qop=auth-int), the H(entity-body) is the hash of the entity body, not the message body - it is computed before any transfer encoding is applied by the sender and after it has been removed by the recipient. Note that this includes multipart boundaries and embedded header fields in each part of any multipart content-type.

3.4.6. Various Considerations

The "Method" value is the HTTP request method, in US-ASCII letters, as specified in section 3.1.1 of [RFC7230]. The "request-target" value is the request-target from the request line as specified in section 3.1.1 of [RFC7230]. This MAY be "*", an "absolute-URI" or an "absolute-path" as specified in section 2.7 of [RFC7230], but it MUST agree with the request-target. In particular, it MUST be an "absolute-URI" if the request-target is an "absolute-URI". The "cnonce" value is a client-chosen value whose purpose is to foil chosen plaintext attacks.

The authenticating server MUST assure that the resource designated by the "uri" parameter is the same as the resource specified in the Request-Line; if they are not, the server SHOULD return a 400 Bad Request error. (Since this may be a symptom of an attack, server implementers may want to consider logging such errors.) The purpose of duplicating information from the request URL in this field is to deal with the possibility that an intermediate proxy may alter the

client's Request-Line. This altered (but presumably semantically equivalent) request would not result in the same digest as that calculated by the client.

Implementers should be aware of how authenticated transactions need to interact with shared caches. The HTTP protocol specifies that when a shared cache (see [RFC7234]) has received a request containing an Authorization header field and a response from relaying that request, it MUST NOT return that response as a reply to any other request, unless one of two Cache-Control (see section 3.2 of [RFC7234]) directive was present in the response. If the original response included the "must-revalidate" Cache-Control directive, the cache MAY use the entity of that response in replying to a subsequent request, but MUST first revalidate it with the origin server, using the request header fields from the new request to allow the origin server to authenticate the new request. Alternatively, if the original response included the "public" Cache-Control directive, the response entity MAY be returned in reply to any subsequent request.

3.5. The Authentication-Info Header

The Authentication-Info header field is a generic field that MAY be used by a server to communicate some information regarding the successful authentication of a client response. The following is the syntax of the header:

```
Authentication-Info = auth-info

auth-info = #auth-param
```

The auth-param is defined in [RFC7235].

3.5.1. Digest Usage of Authentication-Info

The Digest authentication scheme MAY add the Authentication-Info header field in the confirmation request and include parameters from the following list:

nextnonce

> The value of the nextnonce parameter is the nonce the server wishes the client to use for a future authentication response. The server MAY send the Authentication-Info header field with a nextnonce field as a means of implementing one-time or otherwise changing nonces. If the nextnonce field is present the client SHOULD use it when constructing the Authorization header field for its next request. Failure of the client to do so MAY result in a request to re-authenticate from the server with the "stale=TRUE".

Server implementations SHOULD carefully consider the performance implications of the use of this mechanism; pipelined requests will not be possible if every response includes a nextnonce parameter that MUST be used on the next request received by the server. Consideration SHOULD be given to the performance vs. security tradeoffs of allowing an old nonce value to be used for a limited time to permit request pipelining. Use of the "nc" parameter can retain most of the security advantages of a new server nonce without the deleterious affects on pipelining.

qop

Indicates the "quality of protection" options applied to the response by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection. The server SHOULD use the same value for the qop parameter in the response as was sent by the client in the corresponding request.

rspauth

The optional response digest in the "rspauth" parameter supports mutual authentication -- the server proves that it knows the user's secret, and with qop=auth-int also provides limited integrity protection of the response. The "rspauth" value is calculated as for the response in the Authorization header field, except that if "qop=auth" or is not specified in the Authorization header field for the request, A2 is

A2 = ":" request-uri

and if "qop=auth-int", then A2 is

A2 = ":" request-uri ":" H(entity-body)

cnonce and nc

The "cnonce" value and "nc" value MUST be the ones for the client request to which this message is the response. The "rspauth", "cnonce", and "nc" parameters MUST be present if "qop=auth" or "qop=auth-int" is specified.

The Authentication-Info header field is allowed in the trailer of an HTTP message transferred via chunked transfer-coding.

For historical reasons, a sender MUST only generate the quoted-string syntax for the following parameters: nextnonce, rspauth, and cnonce.

For historical reasons, a sender MUST NOT generate the quoted-string syntax for the following parameters: qop and nc.

For historical reasons, the nc value MUST be exactly 8 hexadecimal digits.

3.6. Digest Operation

Upon receiving the Authorization header field, the server MAY check its validity by looking up the password that corresponds to the submitted username. Then, the server MUST perform the same digest operation (e.g. MD5, SHA-256) performed by the client, and compare the result to the given response value.

Note that the HTTP server does not actually need to know the user's cleartext password. As long as H(A1) is available to the server, the validity of an Authorization header field can be verified.

The client response to a WWW-Authenticate challenge for a protection space starts an authentication session with that protection space. The authentication session lasts until the client receives another WWW-Authenticate challenge from any server in the protection space. A client SHOULD remember the username, password, nonce, nonce count and opaque values associated with an authentication session to use to construct the Authorization header field in future requests within that protection space. The Authorization header field MAY be included preemptively; doing so improves server efficiency and avoids extra round trips for authentication challenges. The server MAY choose to accept the old Authorization header field information, even though the nonce value included might not be fresh. Alternatively, the server MAY return a 401 response with a new nonce value, causing the client to retry the request; by specifying stale=TRUE with this response, the server tells the client to retry with the new nonce, but without prompting for a new username and password.

Because the client is REQUIRED to return the value of the opaque parameter given to it by the server for the duration of a session, the opaque data can be used to transport authentication session state information. (Note that any such use can also be accomplished more easily and safely by including the state in the nonce.) For example, a server could be responsible for authenticating content that actually sits on another server. It would achieve this by having the first 401 response include a domain parameter whose value includes a URI on the second server, and an opaque parameter whose value contains the state information. The client will retry the request, at which time the server might respond with "HTTP Redirection (Section 6.4 of [RFC7231]), pointing to the URI on the second server.

The client will follow the redirection, and pass an Authorization header field, including the <opaque> data.

Proxies MUST be completely transparent in the Digest access authentication scheme. That is, they MUST forward the WWW-Authenticate, Authentication-Info and Authorization header fields untouched. If a proxy wants to authenticate a client before a request is forwarded to the server, it can be done using the Proxy-Authenticate and Proxy-Authorization header fields described in section 3.8 below.

3.7. Security Protocol Negotiation

It is useful for a server to be able to know which security schemes a client is capable of handling.

It is possible that a server wants to require Digest as its authentication method, even if the server does not know that the client supports it. A client is encouraged to fail gracefully if the server specifies only authentication schemes it cannot handle.

When a server receives a request to access a resource, the server might challenge the client by responding with "401 Unauthorized" response, and include one or more WWW-Authenticate header fields. If the server responds with multiple challenges, then each one of these challenges MUST use a different digest algorithm. The server MUST add these challenges to the response in order of preference, starting with the most preferred algorithm, followed by the less preferred algorithm.

This specification defines the following algorithms:

- SHA2-256 (mandatory to implement)
- SHA2-512/256 (as a backup algorithm)
- MD5 (for backward compatibility).

When the client receives the first challenge it SHOULD use the topmost header field that it supports, unless a local policy dictates otherwise. The client MUST ignore any challenge it does not understand.

3.8. Proxy-Authenticate and Proxy-Authorization

The digest authentication scheme can also be used for authenticating users to proxies, proxies to proxies, or proxies to origin servers by use of the Proxy-Authenticate and Proxy-Authorization header fields.

These header fields are instances of the Proxy-Authenticate and Proxy-Authorization header fields specified in sections 4.2 and 4.3 of the HTTP/1.1 specification [RFC7235] and their behavior is subject to restrictions described there. The transactions for proxy authentication are very similar to those already described. Upon receiving a request which requires authentication, the proxy/server MUST issue the "407 Proxy Authentication Required" response with a "Proxy-Authenticate" header field. The digest-challenge used in the Proxy-Authenticate header field is the same as that for the WWW-Authenticate header field as defined above in Section 3.3.

The client/proxy MUST then re-issue the request with a Proxy-Authorization header field, with parameters as specified for the Authorization header field in section 3.4 above.

On subsequent responses, the server sends Proxy-Authenticate-Info with parameters the same as those for the Authentication-Info header field.

Note that in principle a client could be asked to authenticate itself to both a proxy and an end-server, but never in the same response.

3.9. Examples

3.9.1. Example with SHA-256 and MD5

The following example assumes that an access protected document is being requested from the server via a GET request. The URI of the document is http://www.example.org/dir/index.html". Both client and server know that the username for this document is "Mufasa" and the password is "Circle of Life" (with one space between each of the three words).

The first time the client requests the document, no Authorization header field is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
    realm="http-auth@example.org",
    qop="auth, auth-int",
    algorithm=SHA-256,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS",
WWW-Authenticate: Digest
    realm="http-auth@example.org",
    qop="auth, auth-int",
    algorithm=MD5,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

The client can prompt the user for their username and password, after which it will respond with a new request, including the following Authorization header field if the client chooses MD5 digest:

```
Authorization: Digest username="Mufasa",
    realm="http-auth@example.org",
    uri=/dir/index.html,
    algorithm=MD5,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    nc=00000001,
    cnonce="f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ",
    qop=auth,
    response="8ca523f5e9506fed4657c9700eebdbec",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

If the client chooses to use the SHA-256 algorithm for calculating the response, the client responds with a new request including the following Authorization header field:

```
Authorization: Digest username="Mufasa",
    realm="http-auth@example.org",
    uri="/dir/index.html",
    algorithm=SHA-256,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    nc=00000001,
    cnonce="f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ",
    qop=auth,
    response="753927fa0e85d155564e2e272a28d1802ca10daf449
       6794697cf8db5856cb6c1",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

3.9.2. Example with SHA-512-256, Charset, and Userhash

The following example assumes that an access protected document is being requested from the server via a GET request. The URI for the request is "http://api.example.org/doe.json". Both client and server know the userhash of the username, support the UTF-8 character encoding scheme, and use the SHA-512-256 algorithm. The username for the request is a variation of "Jason Doe" and has the byte sequence 4A C3 A4 73 C3 B8 6E 20 44 6F 65. The password is "Secret, or not?".

The first time the client requests the document, no Authorization header field is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
    realm="api@example.org",
    qop=auth,
    algorithm=SHA-512-256,
    nonce="5TsQWLVdgBdmrQ0XsxbDODV+57QdFR34I9HAbC/RVvkK",
    opaque="HRPCssKJSGjCrkzDg8OhwpzCiGPChXYjwrI2QmXDnsOS",
    charset=UTF-8,
    userhash=true
```

The client can prompt the user for the required credentials and send a new request with following Authorization header field:

```
Authorization: Digest
    username="488869477bf257147b804c45308cd62ac4e25eb717
       b12b298c79e62dcea254ec",
    realm="api@example.org",
    uri="/doe.json",
    algorithm=SHA-512-256,
    nonce="5TsQWLVdgBdmrQ0XsxbDODV+57QdFR34I9HAbC/RVvkK",
    nc=00000001,
    cnonce="NTg6RKcb9boFIAS3KrFK9BGeh+iDa/sm6jUMp2wds69v",
    qop=auth,
    response="ae66e67d6b427bd3f120414a82e4acff38e8ecd9101d
       6c861229025f607a79dd",
    opaque="HRPCssKJSGjCrkzDg8OhwpzCiGPChXYjwrI2QmXDnsOS",
    userhash=true
```

If the client can not provide a hashed username for any reason, the client can try a request with this Authorization header field:

```
Authorization: Digest
    username*=UTF-8''J%C3%A4s%C3%B8n%20Doe,
    realm="api@example.org",
    uri="/doe.json",
    algorithm=SHA-512-256,
    nonce="5TsQWLVdgBdmrQ0XsxbDODV+57QdFR34I9HAbC/RVvkK",
    nc=00000001,
    cnonce="NTg6RKcb9boFIAS3KrFK9BGeh+iDa/sm6jUMp2wds69v",
    qop=auth,
    response="ae66e67d6b427bd3f120414a82e4acff38e8ecd9101d6
       c861229025f607a79dd",
    opaque="HRPCssKJSGjCrkzDg8OhwpzCiGPChXYjwrI2QmXDnsOS",
    userhash=false
```

4. Internationalization

In challenges, servers SHOULD use the "charset" authentication parameter (case-insensitive) to express the character encoding they expect the user agent to use when generating A1 (see section 3.4.2) and username hashing (see section 3.4.4).

The only allowed value is "UTF-8", to be matched case-insensitively (see [RFC2978], Section 2.3). It indicates that the server expects user name and password to be converted to Unicode Normalization Form C ("NFC", see Section 3 of [RFC5198]) and to be encoded into octets using the UTF-8 character encoding scheme [RFC3629].

For the username, recipients MUST support all characters defined in the "UsernameCasePreserved" profile defined in in Section 3.3 of [PRECIS], with the exception of the colon (":") character.

For the password, recipients MUST support all characters defined in the "OpaqueString" profile defined in in Section 4.2 of [PRECIS].

If the user agent does not support the encoding indicated by the server, it can fail the request.

5. Security Considerations

5.1. Limitations

HTTP Digest authentication, when used with human-memorable passwords, is vulnerable to dictionary attacks. Such attacks are much easier than cryptographic attacks on any widely used algorithm, including those that are no longer considered secure. In other words, algorithm agility does not make this usage any more secure.

As a result, Digest authentication SHOULD be used only with passwords that have a reasonable amount of entropy, e.g. 128-bit or more. Such passwords typically cannot be memorized by humans but can be used for automated web services.

Digest authentication SHOULD be used over a secure channel like HTTPS [RFC2818].

5.2. Storing passwords

Digest authentication requires that the authenticating agent (usually the server) store some data derived from the user's name and password in a "password file" associated with a given realm. Normally this might contain pairs consisting of username and H(A1), where H(A1) is the digested value of the username, realm, and password as described above.

The security implications of this are that if this password file is compromised, then an attacker gains immediate access to documents on the server using this realm. Unlike, say a standard UNIX password file, this information need not be decrypted in order to access documents in the server realm associated with this file. On the other hand, decryption, or more likely a brute force attack, would be necessary to obtain the user's password. This is the reason that the realm is part of the digested data stored in the password file. It means that if one Digest authentication password file is compromised, it does not automatically compromise others with the same username and password (though it does expose them to brute force attack).

There are two important security consequences of this. First the password file must be protected as if it contained unencrypted passwords, because for the purpose of accessing documents in its realm, it effectively does.

A second consequence of this is that the realm string SHOULD be unique among all realms which any single user is likely to use. In particular a realm string SHOULD include the name of the host doing the authentication. The inability of the client to authenticate the server is a weakness of Digest Authentication.

5.3. Authentication of Clients using Digest Authentication

Digest Authentication does not provide a strong authentication mechanism, when compared to public key based mechanisms, for example.

However, it is significantly stronger than (e.g.) CRAM-MD5, which has been proposed for use with LDAP [RFC4513], POP and IMAP (see

[RFC2195]). It was intended to replace the much weaker and even more dangerous Basic mechanism.

Digest Authentication offers no confidentiality protection beyond protecting the actual username and password. All of the rest of the request and response are available to an eavesdropper.

Digest Authentication offers only limited integrity protection for the messages in either direction. If qop=auth-int mechanism is used, those parts of the message used in the calculation of the WWW-Authenticate and Authorization header field response parameter values (see section 3.2 above) are protected. Most header fields and their values could be modified as a part of a man-in-the-middle attack.

Many needs for secure HTTP transactions cannot be met by Digest Authentication. For those needs TLS is more appropriate protocol. In particular Digest authentication cannot be used for any transaction requiring confidentiality protection. Nevertheless many functions remain for which Digest authentication is both useful and appropriate.

5.4. Limited Use Nonce Values

The Digest scheme uses a server-specified nonce to seed the generation of the response value (as specified in section 3.4.1 above). As shown in the example nonce in section 3.3, the server is free to construct the nonce such that it MAY only be used from a particular client, for a particular resource, for a limited period of time or number of uses, or any other restrictions. Doing so strengthens the protection provided against, for example, replay attacks (see 4.5). However, it should be noted that the method chosen for generating and checking the nonce also has performance and resource implications. For example, a server MAY choose to allow each nonce value to be used only once by maintaining a record of whether or not each recently issued nonce has been returned and sending a next-nonce parameter in the Authentication-Info header field of every response. This protects against even an immediate replay attack, but has a high cost checking nonce values, and perhaps more important will cause authentication failures for any pipelined requests (presumably returning a stale nonce indication). Similarly, incorporating a request-specific element such as the Etag value for a resource limits the use of the nonce to that version of the resource and also defeats pipelining. Thus it MAY be useful to do so for methods with side effects but have unacceptable performance for those that do not.

5.5. Replay Attacks

A replay attack against Digest authentication would usually be pointless for a simple GET request since an eavesdropper would already have seen the only document he could obtain with a replay. This is because the URI of the requested document is digested in the client request and the server will only deliver that document. By contrast under Basic Authentication once the eavesdropper has the user's password, any document protected by that password is open to him.

Thus, for some purposes, it is necessary to protect against replay attacks. A good Digest implementation can do this in various ways. The server created "nonce" value is implementation dependent, but if it contains a digest of the client IP, a time-stamp, the resource ETag, and a private server key (as recommended above) then a replay attack is not simple. An attacker must convince the server that the request is coming from a false IP address and must cause the server to deliver the document to an IP address different from the address to which it believes it is sending the document. An attack can only succeed in the period before the time-stamp expires. Digesting the client IP and time-stamp in the nonce permits an implementation which does not maintain state between transactions.

For applications where no possibility of replay attack can be tolerated the server can use one-time nonce values which will not be honored for a second use. This requires the overhead of the server remembering which nonce values have been used until the nonce time-stamp (and hence the digest built with it) has expired, but it effectively protects against replay attacks.

An implementation must give special attention to the possibility of replay attacks with POST and PUT requests. Unless the server employs one-time or otherwise limited-use nonces and/or insists on the use of the integrity protection of qop=auth-int, an attacker could replay valid credentials from a successful request with counterfeit form data or other message body. Even with the use of integrity protection most metadata in header fields is not protected. Proper nonce generation and checking provides some protection against replay of previously used valid credentials, but see 4.8.

5.6. Weakness Created by Multiple Authentication Schemes

An HTTP/1.1 server MAY return multiple challenges with a 401 (Authenticate) response, and each challenge MAY use a different auth-scheme. A user agent MUST choose to use the strongest auth-scheme it understands and request credentials from the user based upon that challenge.

Note that many browsers will only recognize Basic and will require that it be the first auth-scheme presented. Servers SHOULD only include Basic if it is minimally acceptable.

When the server offers choices of authentication schemes using the WWW-Authenticate header field, the strength of the resulting authentication is only as good as that of the of the weakest of the authentication schemes. See Section 5.7 below for discussion of particular attack scenarios that exploit multiple authentication schemes.

5.7. Online dictionary attacks

If the attacker can eavesdrop, then it can test any overheard nonce/response pairs against a list of common words. Such a list is usually much smaller than the total number of possible passwords. The cost of computing the response for each password on the list is paid once for each challenge.

The server can mitigate this attack by not allowing users to select passwords that are in a dictionary.

5.8. Man in the Middle

Digest authentication is vulnerable to "man in the middle" (MITM) attacks, for example, from a hostile or compromised proxy. Clearly, this would present all the problems of eavesdropping. But it also offers some additional opportunities to the attacker.

A possible man-in-the-middle attack would be to add a weak authentication scheme to the set of choices, hoping that the client will use one that exposes the user's credentials (e.g. password). For this reason, the client SHOULD always use the strongest scheme that it understands from the choices offered.

An even better MITM attack would be to remove all offered choices, replacing them with a challenge that requests only Basic authentication, then uses the cleartext credentials from the Basic authentication to authenticate to the origin server using the stronger scheme it requested. A particularly insidious way to mount such a MITM attack would be to offer a "free" proxy caching service to gullible users.

User agents should consider measures such as presenting a visual indication at the time of the credentials request of what authentication scheme is to be used, or remembering the strongest authentication scheme ever requested by a server and produce a warning message before using a weaker one. It might also be a good

idea for the user agent to be configured to demand Digest authentication in general, or from specific sites.

Or, a hostile proxy might spoof the client into making a request the attacker wanted rather than one the client wanted. Of course, this is still much harder than a comparable attack against Basic Authentication.

5.9. Chosen plaintext attacks

With Digest authentication, a MITM or a malicious server can arbitrarily choose the nonce that the client will use to compute the response. This is called a "chosen plaintext" attack. The ability to choose the nonce is known to make cryptanalysis much easier.

However, no way to analyze the one-way functions used by Digest using chosen plaintext is currently known.

The countermeasure against this attack is for clients to use the "cnonce" parameter; this allows the client to vary the input to the hash in a way not chosen by the attacker.

5.10. Precomputed dictionary attacks

With Digest authentication, if the attacker can execute a chosen plaintext attack, the attacker can precompute the response for many common words to a nonce of its choice, and store a dictionary of (response, password) pairs. Such precomputation can often be done in parallel on many machines. It can then use the chosen plaintext attack to acquire a response corresponding to that challenge, and just look up the password in the dictionary. Even if most passwords are not in the dictionary, some might be. Since the attacker gets to pick the challenge, the cost of computing the response for each password on the list can be amortized over finding many passwords. A dictionary with 100 million password/response pairs would take about 3.2 gigabytes of disk storage.

The countermeasure against this attack is to for clients to use the "cnonce" parameter.

5.11. Batch brute force attacks

With Digest authentication, a MITM can execute a chosen plaintext attack, and can gather responses from many users to the same nonce. It can then find all the passwords within any subset of password space that would generate one of the nonce/response pairs in a single pass over that space. It also reduces the time to find the first password by a factor equal to the number of nonce/response pairs

gathered. This search of the password space can often be done in parallel on many machines, and even a single machine can search large subsets of the password space very quickly -- reports exist of searching all passwords with six or fewer letters in a few hours.

The countermeasure against this attack is to for clients to use of the "cnonce" parameter.

5.12. Summary

By modern cryptographic standards Digest Authentication is weak. But for a large range of purposes it is valuable as a replacement for Basic Authentication. It remedies some, but not all, weaknesses of Basic Authentication. Its strength may vary depending on the implementation. In particular the structure of the nonce (which is dependent on the server implementation) may affect the ease of mounting a replay attack. A range of server options is appropriate since, for example, some implementations may be willing to accept the server overhead of one-time nonces or digests to eliminate the possibility of replay. Others may satisfied with a nonce like the one recommended above restricted to a single IP address and a single ETag or with a limited lifetime.

The bottom line is that *any* compliant implementation will be relatively weak by cryptographic standards, but *any* compliant implementation will be far superior to Basic Authentication.

6. IANA Considerations

6.1. HTTP Digest Hash Algorithms Registry

This specification creates a new IANA registry named "HTTP Digest Hash Algorithms". When registering a new hash algorithm, the following information MUST be provided:

Hash Algorithm

The textual name of the hash algorithm.

Digest Size

The size of the algorithm's output in bits.

Reference

A reference to the specification that describes the new algorithm.

The update policy for this registry shall be Specification Required.

The initial registry will contain the following entries:

Hash Algorithm	Digest Size	Reference
"MD5"	128	RFC XXXX
"SHA-512-256"	256	RFC XXXX
"SHA-256"	256	RFC XXXX

Each one of the algorithms defined in the registry might have a -sess variant, e.g. MD5-sess, SHA-256-sess, etc.

6.2. Digest Scheme Registration

This specification registers the Digest scheme with the Authentication Scheme Registry.

Authentication Scheme Name: Digest

Pointer to specification text: this specification

6.3. Authentication-Info Header Registration

This specification registers the Authentication-Info Header field with the Message Header Field Registry.

Header Field Name: Authentication-Info

Protocol: http

Status: standard

Reference: RFCXXXX, Section 3.5

7. Acknowledgments

The authors of this document would like to thank the authors of [RFC2617], as this document heavily borrows text from their document to provide a complete description of the digest scheme and its operations.

Special thanks to Julian Reschke for his reviews, comments, suggestions, and text provided to various areas in this document.

The authors would like to thank Stephen Farrell, Yoav Nir, Phillip Hallam-Baker, Manu Sporny, Paul Hoffman, Yaron Sheffer, Sean Turner, Geoff Baskwill, Eric Cooper, Bjoern Hoehrmann, Martin Durst, Peter

Saint-Andre, Michael Sweet, Daniel Stenberg, Brett Tate, Paul Leach, Ilari Liusvaara, and Gary Mort, Alexey Melnikov, and Benjamin Kaduk for their careful review and comments.

The authors would like to thank Jonathan Stoke, Nico Williams, Harry Halpin, and Phil Hunt for their comments on the mailing list when discussing various aspects of this document.

The authors would like to thank Paul Kyzivat and Dale Worley for their careful review and feedback on some aspects of this document.

8. References

8.1. Normative References

[PRECIS] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", draft-ietf-precis-saslprepbis-12 (work in progress), December 2014.

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

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

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

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[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.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

8.2. Informative References

[BASIC] Reschke, J., "The 'Basic' HTTP Authentication Scheme", draft-ietf-httpauth-basicauth-update-04 (work in progress), December 2014.

[RFC2195] Klensin, J., Catoe, R., and P. Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997.

[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.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4513] Harrison, R., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

Authors' Addresses

Rifaat Shekh-Yusef (editor)
Avaya
250 Sidney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
EMail: rifaat.ietf@gmail.com

David Ahrens
Independent
California
USA

EMail: ahrensdc@gmail.com

Sophie Bremer
Netzkonform
Germany

EMail: sophie.bremer@netzkonform.de