

Network Working Group
Internet-Draft
Expires: October 8, 2006

R. Sayre

A. Melnikov
Isode Ltd.
April 6, 2006

HMAC Digest Access Authentication for HTTP
draft-sayre-http-hmac-digest-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#). This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

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 October 8, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document specifies an HTTP authentication scheme based on cryptographic hashes.

Editor's Note

Internet-Draft

HMAC Digest Authentication

April 2006

To discuss this draft, please join the ietf-http-auth mailing list [[1](#)]. Membership is open to all.

Table of Contents

1.	Introduction	3
2.	WWW-Authenticate	3
3.	Authorization	5
4.	The Request Digest	6
5.	Header Handling	7
6.	Acknowledgements	8
7.	IANA Considerations	8
8.	Security Considerations	8
9.	Normative References	8
Appendix A.	Example Implementations	9
A.1.	Example Server	9
A.2.	Example Client	11
Appendix B.	Change Log	13
	Authors' Addresses	14
	Intellectual Property and Copyright Statements	15

1. Introduction

This document specifies an HTTP authentication scheme similar to the Digest scheme [[RFC2617](#)]. It borrows heavily from that scheme's specification, but there are substantive differences. Most importantly, the algorithm is based on a hash of the user password, rather than the password itself. In addition, the scheme defined in this document allows for additional message integrity checks on HTTP request headers. It omits quality-of-protection options and Authentication-Info headers from the server. Like Digest authentication, the scheme specified in this document suffers from many known weaknesses, and is only intended to improve on Basic authentication [[RFC2617](#)].

This specification is a companion to the HTTP/1.1 specification [[RFC2616](#)]. It uses the augmented BNF [section 2.1](#) of that document, and relies on both the non-terminals defined in that document and other aspects of the HTTP/1.1 specification.

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 [RFC 2119](#) [[RFC2119](#)].

2. WWW-Authenticate

When a server receives a request for an access-protected object without an acceptable Authorization header, it responds with a "401 Unauthorized" status code, and a WWW-Authenticate header [[RFC2617](#)]. For the HMAC Digest scheme, the value of the header is as follows:

```
challenge      = "HMACDigest" digest-challenge
digest-challenge = 1#( realm | snonce | [domain] | [reason] |
                    [algorithm] | [pw-algorithm] |
                    [salt] | [auth-param] )
```

```

realm          = "realm" "=" quoted-string
snonce         = "snonce" "=" quoted-string
domain         = "domain" "=" <"> URI *( 1*SP URI ) <">
URI            = absoluteURI | abs_path
reason         = "reason" "=" ("unauthorized" | "integrity" |
                                token)
algorithm       = "algorithm" "=" ( "HMAC-SHA-1" | "HMAC-MD5" |
                                token )
pw-algorithm    = "pw-algorithm" "=" ( "SHA-1" | "MD5" |
                                token )
salt           = "salt" "=" quoted-string

```

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".

snonce: A server-specified data string which should be uniquely generated each time a 401 response is made. It is recommended that this string be base64 or hexadecimal data. Specifically, since the string is passed in the header lines as a quoted string, the double-quote character is not allowed.

The contents of the snonce are implementation dependent and opaque to the client. The quality of the implementation depends on a good choice. An snonce might, for example, be constructed as the base64 encoding of

```
time-stamp hash(time-stamp ":" ETag ":" private-key)
```

where time-stamp is a server-generated time or other non-repeating value, ETag is the value of the HTTP ETag header associated with the requested entity, and private-key is data known only to the server. With an snonce of this form, a server would recalculate the hash portion after receiving the client authentication header and reject the request if it did not match the snonce from that header or if the time-stamp value is not recent enough. In this way, the server can limit the time of the snonce's validity.[@ Eliminate replay text?] The inclusion of the ETag prevents a replay request for an updated version of the resource. (Note:

including the IP address of the client in the snonce would appear to offer the server the ability to limit the reuse of the snonce to the same client that originally got it. However, that would break proxy farms, where requests from a single user often go through different proxies in the farm. Also, IP address spoofing is not that hard.)

domain: A quoted, space-separated list of URI references [[RFC3986](#)] that define the protection space. If a URI is an abs_path, it is relative to the canonical root URL [[RFC2617](#)] of the server being accessed. An absoluteURI in this list may refer to a different server than the one being accessed. 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 directive is omitted or its value is empty, the client should assume that the protection space consists of all URIs on the responding server.

reason: The value of this directive indicates the reason for the rejection of the previous client request. "unauthorized" indicates that the request did not contain a valid digest. "stale" indicates that the previous request from the client was rejected because the snonce value was stale. The client may wish to simply retry the request with a new encrypted response, without reprompting the user for a new username and password. "integrity" indicates that the request contained unverified content that the server requires be included in the calculation of the digest. If the directive is not present, or a value other than "integrity" or "stale", the client should behave as though its value were "unauthorized".

algorithm: This directive indicates the HMAC construction to be used [[RFC2104](#)]. If not present, it is assumed to be "HMAC-SHA-1".

pw-algorithm: This directive indicates the algorithm to be used when preparing an HMAC key. If not present, it is assumed to be "SHA-1".

salt: If present, this directive indicates a value that is appended to the password before the initial hash function is applied.

3. Authorization

The Authorization request header contains client credentials generated according to the directives recieved in a WWW-Authenticate response header.

```
credentials      = "HMACDigest" digest-response
digest-response  = 1#( username | realm | cnonce | snonce |
                      digest-uri | created | response | [headers] |
                      [auth-param] )
username         = "username" "=" quoted-string
cnonce           = "cnonce" "=" quoted-string
digest-uri       = "uri" "=" request-uri
response         = "response" "=" request-digest
request-digest   = <"> *LHEX <">
LHEX             = "0" | "1" | "2" | "3" |
                  "4" | "5" | "6" | "7" |
                  "8" | "9" | "a" | "b" |
                  "c" | "d" | "e" | "f"
headers          = "headers" "=" header-list
header-list      = <"> field-name *( 1*SP field-name ) <">
```

username: The user's name in the specified realm.

cnonce: The cnonce value is an opaque quoted 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.

digest-uri: The URI from Request-URI of the Request-Line; duplicated here because proxies are allowed to change the Request-Line in transit.

response: The response value is a string containing hexadecimal data. The two HMAC constructions listed by this specification will produce strings of 32 or 40 characters in length.

created: The created value is an [RFC3339](#) timestamp [[RFC3339](#)].

headers: The headers value is a space-separated list of HTTP headers used to calculate the request-digest.

[4.](#) The Request Digest

This section describes the process a client uses to calculate the request digest, and how the server can verify it.

1. The client applies the algorithm specified by the pw-algorithm directive to the user password. If present, the value of the salt directive is appended to the password prior to calculation.
2. The client applies the algorithm specified by the pw-algorithm directive to the concatenation of the username, a colon, the lowercased hexadecimal digest of the result of step 1, a colon, and the value of the realm directive. The lowercased hexadecimal digest of the result serves as the HMAC key.
3. The client generates a cnonce, a data string which should be uniquely generated each time a request is made. It is recommended that this string be base64 or hexadecimal data. Specifically, since the string is passed in the header lines as a quoted string, the double-quote character is not allowed. A combination of a timestamp and a random number is sufficient for many purposes.
4. The client forms a list of request headers it wishes to include in the digest calculation. The most useful headers to include are entity headers such as Content-Type, Content-Length, and Content-MD5 (see [Section 5](#)).

5. The client generates a timestamp using the current time.
6. The client concatenates the request method, a colon, the request URI, a colon, the cnonce, a colon, the snonce, a colon, and the value of each applicable header in the header list (see [Section 5](#)). This value is the message data.
7. The client applies the HMAC construction specified by the

algorithm directive to the key and the message data. The lowercased hexadecimal digest of this calculation is the value of the response directive.

8. The client then uses the relevant values to compose an Authorization header, and sends the request.

When the server receives a request containing an Authorization header using the HMAC Digest scheme, it can validate its value using the procedure listed below.

1. The server should already have the hash of the user's password available, using the algorithm it instructed the client to use in the WWW-Authenticate header.
2. The server uses the realm and username directives supplied in the Authorization header to check for a candidate key.
3. The server concatenates the request method, a colon, the value of the uri directive, a colon, the value of the nonce directive, a colon, the value of the created directive, a colon, and the value of each header in the headers directive (see [Section 5](#)). This value is the message data.
4. The server uses the key to calculate the HMAC for the message data. If the hexadecimal digest of this calculation matches the value provided in the response directive, the request is valid.

[5](#). Header Handling

When selecting headers for inclusion in the Digest calculation, clients SHOULD NOT include hop-by-hop headers. HTTP 1.1 [\[RFC2616\]](#) defines eight hop-by-hop headers: Connection, Keep-Alive, Proxy-Authenticate, Proxy-Authorization, TE, Trailers, Transfer-Encoding, and Upgrade. HTTP 1.1 also requires that extension hop-by-hop headers are listed in the Connection header.

When creating or verifying a digest, leading whitespace in the header values MUST be stripped and header unfolding MUST NOT be done. If a

header in the header list appears multiple times, those values are be

combined in order. For example, if a client specifies headers="A B", and the request contains A,B,A headers (in that order), both sides MUST calculate the digest using header values in the order A,A,B.

6. Acknowledgements

This document is largely based on the Digest section of "HTTP Authentication: Basic and Digest Access Authentication" [[RFC2617](#)], and includes some sections of that document verbatim.

The technique of including header values in the digest calculation was originally proposed by James Underly for the SIP protocol.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

[TBD... depends on how much needs to be repeated from [RFC2617](#).]

9. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [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.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), July 2002.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66,

[RFC 3986](#), January 2005.

- [1] <<http://lists.osafoundation.org/cgi-bin/mailman/listinfo/ietf-http-auth>>
- [2] <<http://www.python.org>>

[Appendix A](#). Example Implementations

This section provides example implementations in the Python [\[2\]](#) programming language, version 2.4.

[A.1](#). Example Server

The example server program responds to all request URIs with the same response, and knows of only one user. If the server program is saved in the file "hmac-digest-server.py", it can be started by typing "python hmac-digest-server.py".

```
import BaseHTTPServer, cgi, urllib2
import time
import hmac, sha, md5, base64

PORT = 8888
user = "user"
password = "password"
salt = 'xyzzzy'
realm = "HMACDigest Sample"
algo = "HMAC-SHA-1"
pw_algo = "MD5"
key_str = "%s:%s:%s" % (user,
                        md5.new(password+salt).hexdigest(),
                        realm)
key = md5.new(key_str).hexdigest()

secret_key = "moo"
digest_header = 'HMACDigest realm="%s", '
digest_header += 'snonce="%s", '
digest_header += 'reason="%s", '
digest_header += 'domain="/ http://www.example.com/", '
digest_header += 'algorithm="%s", '
digest_header += 'pw-algorithm="%s", '
digest_header += 'salt="%s"'
```

```
class HMACDigestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
```

```
        auth = self.headers.getheader('authorization')
        result = self.check(auth)
        if result is False:
            self.send_401()
        elif result == 'stale':
            self.send_401(reason='stale')
        else:
            self.send_response(200)
            self.send_header('Content-type','text/plain')
            self.end_headers()
            self.wfile.write("\nAuthentication Successful!\n")

    def check(self, auth):
        if auth is None:
            return False
        token, fields = auth.split(' ', 1)
        if token != 'HMACDigest':
            return False
        cred = urllib2.parse_http_list(fields)
        cred = urllib2.parse_keqv_list(cred)
        if cred['username'] != user or cred['realm'] != realm:
            return False
        snonce = cred['snonce']
        nonce_time,s_hash = base64.b64decode(snonce).split()
        test_hash = md5.new(nonce_time +
                            ":fake_etag:" + secret_key).hexdigest()
        if s_hash != test_hash:
            return False
        now = time.mktime(time.gmtime())
        # allow 10 minute old nonces... this is arbitrary
        if now - float(nonce_time) > 600:
            return "stale"
        names = cred.get('headers','').split()
        vals = ''.join([self.headers.getheader(h) for h in names])
        msg = "%s:%s:%s:%s:%s" % (self.command, self.path,
                                   cred['cnonce'], snonce,
                                   vals)
        the_hmac = hmac.new(key, msg, sha).hexdigest()
        if cred['response'] == the_hmac:
```

```

        return True
    else:
        return False

def send_401(self, reason="unauthorized"):
    auth_header = digest_header % (realm, self.snonce(),
                                    reason, algo, pw_algo, salt)
    self.send_response(401)
    self.send_header('WWW-Authenticate', auth_header)

```

```

        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        self.wfile.write('\nUnauthorized\n')

def snonce(self):
    now = str(time.mktime(time.gmtime()))
    print now
    h = md5.new(now+":fake_etag:"+secret_key).hexdigest()
    return base64.b64encode(now + " " + h)

httpd = BaseHTTPServer.HTTPServer(("", PORT), HMACDigestHandler)
print "Serving at port", PORT
httpd.serve_forever()

```

[A.2.](#) Example Client

The example client program makes one request to the example server without an authorization header, examines the WWW-Authenticate header returned in the 401 response, and then creates an Authorization header to make a second (successful) request.

```

import httplib, urllib2, md5, sha, hmac, time, random, os

PORT = 8888
username = "user"
password = "password"
params = {}
headers = {"accept": "text/X-0h-Several-Things+xml, */*",
           "user-agent": "libwww-perl/5.803",
           "x-hopbyhop": "some proxy information",

```

```

        "x-fooproxy": "some more proxy info",
        "x-freedom-is-what-you-think-it-is":
        "But there ain't no train to Stockholm",
        "connection": "close, x-hopbyhop, x-fooproxy"}

# Make an initial request
conn = httplib.HTTPConnection("localhost",PORT)
conn.request("GET","",params,headers)
response = conn.getresponse()
data = response.read()

# Print the rejection letter
print "First request:",data

# Get the challenge
wa = response.getheader('WWW-Authenticate')

```

```

token, kv = wa.split(' ', 1)
challenge = urllib2.parse_keqv_list(urllib2.parse_http_list(kv))
realm = challenge['realm']
snonce = challenge['snonce']
algorithm = challenge.get('algorithm', 'HMAC-SHA-1')
pw_algorithm = challenge.get('pw-algorithm', 'SHA-1')
salt = challenge.get('salt','')

# Choose our HMAC construct
if algorithm == 'HMAC-SHA-1':
    hashmod = sha
elif algorithm == 'HMAC-MD5':
    hashmod = md5

# Choose our password algorithm
if pw_algorithm == 'SHA-1':
    pwhashmod = sha
elif pw_algorithm == 'MD5':
    pwhashmod = md5

# Make the key
key = "%s:%s:%s" % (username,
                    pwhashmod.new(password+salt).hexdigest(),
                    realm)
key = pwhashmod.new(key).hexdigest()

```

```

# Put together the headers
keys = set(headers.keys())
hop_by_hops = set(['connection', 'keep-alive', 'proxy-authenticate',
                  'proxy-authorization', 'te', 'trailers',
                  'transfer-encoding', 'upgrade'])
if 'connection' in keys:
    ext_hop_heads = urllib2.parse_http_list(headers['connection'])
    ext_hop_heads.remove('close')
    hop_by_hops = hop_by_hops.union(ext_hop_heads)
keys = keys.difference(hop_by_hops)
keylist = ''.join(["%s " % k for k in keys])
header_vals = ''.join([headers[k] for k in keys])

# Make a cnonce
created = time.strftime('%Y-%m-%dT%H:%M%SZ',time.gmtime())
cnonce = sha.new(str(random.getrandbits(512))+created).hexdigest()

# Calculate the HMAC
msg = "%s:%s:%s:%s:%s" % ("GET", "/", cnonce, snonce, header_vals)
response = hmac.new(key, msg, hashmod).hexdigest()

# Compose the Authorization header

```

```

auth = 'username="%s", realm="%s", cnonce="%s", uri="%s", ' \
       'snonce="%s", response="%s", headers="%s"'
auth = auth % (username, realm, cnonce, "/",
              snonce, response, keylist)
headers['Authorization'] = "HMACDigest " + auth

# wait a few seconds
time.sleep(2)
conn.request("GET", "/", params, headers)
response = conn.getresponse()
data = response.read()

# Print the successful response
print "Second request:", data
conn.close()

```

[Appendix B](#). Change Log

01: Add server nonce, remove client timestamp.

Specify header handling more extensively.

Add [RFC2119](#) terms for header handling.

Demo client excludes hop-by-hop headers.

Sayre & Melnikov

Expires October 8, 2006

[Page 13]

Internet-Draft

HMAC Digest Authentication

April 2006

Authors' Addresses

Robert Sayre

Email: rfsayre@boswijck.com

Alexey Melnikov
Isode Ltd.

Email: Alexey.Melnikov@isode.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to

pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.