Digest access authentication

Digest access authentication is one of the agreed-upon methods a web server can use to negotiate credentials, such as username or password, with a user's web browser. This can be used to confirm the identity of a user before sending sensitive information, such as online banking transaction history. It applies a hash function to a password before sending it over the network, which is safer than basic access authentication, which sends plaintext.

Technically, digest authentication is an application of MD5 cryptographic hashing with usage of nonce values to prevent replay attacks. It uses the HTTP protocol.

1 Overview

Digest access authentication was originally specified by RFC 2069 (*An Extension to HTTP: Digest Access Authentication*). RFC 2069 specifies roughly a traditional digest authentication scheme with security maintained by a server-generated *nonce value*. The authentication response is formed as follows (where HA1 and HA2 are names of string variables):

> HA1=MD5(username:realm:password)
>
> HA2=MD5(method:digestURI)
>
> response=MD5(HA1:nonce:HA2)

RFC 2069 was later replaced by RFC 2617 (*HTTP Authentication: Basic and Digest Access Authentication*). RFC 2617 introduced a number of optional security enhancements to digest authentication; **"quality of protection" (qop)**, nonce counter incremented by client, and a client-generated random nonce. These enhancements are designed to protect against, for example, chosen-plaintext attack cryptanalysis.

If the algorithm directive's value is "MD5" or unspecified, then HA1 is

> HA1=MD5(username:realm:password)

If the algorithm directive's value is "MD5-sess", then HA1 is

> HA1=MD5(MD5(username:realm:
> password):nonce:cnonce)

If the qop directive's value is "auth" or is unspecified, then HA2 is

> HA2=MD5(method:digestURI)

If the qop directive's value is "auth-int", then HA2 is

> HA2=MD5(method:
> digestURI:MD5(entityBody))

If the qop directive's value is "auth" or "auth-int", then compute the response as follows:

> response=MD5(HA1:nonce:nonceCount:
> clientNonce:qop:HA2)

If the qop directive is unspecified, then compute the response as follows:

> response=MD5(HA1:nonce:HA2)

The above shows that when qop is not specified, the simpler RFC 2069 standard is followed.

2 Impact of MD5 security on digest authentication

The MD5 calculations used in HTTP digest authentication is intended to be "one way", meaning that it should be difficult to determine the original input when only the output is known. If the password itself is too simple, however, then it may be possible to test all possible inputs and find a matching output (a brute-force attack) – perhaps aided by a dictionary or suitable look-up list.

The HTTP scheme was designed by Phillip Hallam-Baker at CERN in 1993 and does not incorporate subsequent improvements in authentication systems, such as the development of keyed-hash message authentication code (HMAC). Although the cryptographic construction that is used is based on the MD5 hash function, collision attacks were in 2004 generally believed to not affect applications where the plaintext (i.e. password) is not known.[1] However, claims in 2006[2] cause some doubt over other MD5 applications as well. So far, however, MD5 collision attacks have not been shown to pose a threat to digest authentication, and the RFC 2617 allows servers to implement mechanisms to detect some collision and replay attacks.

3 HTTP digest authentication considerations

3.1 Advantages

HTTP digest authentication is designed to be more secure than traditional digest authentication schemes, for example "significantly stronger than (e.g.) CRAM-MD5 ..." (RFC 2617).

Some of the security strengths of HTTP digest authentication are:

- The password is not used directly in the digest, but rather HA1 = MD5(username:realm:password). This allows some implementations (e.g. JBoss[3]) to store HA1 rather than the cleartext password
- Client nonce was introduced in RFC 2617, which allows the client to prevent chosen-plaintext attacks, such as rainbow tables that could otherwise threaten digest authentication schemes
- Server nonce is allowed to contain timestamps. Therefore, the server may inspect nonce attributes submitted by clients, to prevent replay attacks
- Server is also allowed to maintain a list of recently issued or used server nonce values to prevent reuse

3.2 Disadvantages

Digest access authentication is intended as a security trade-off. It is intended to replace unencrypted HTTP basic access authentication. It is not, however, intended to replace strong authentication protocols, such as public-key or Kerberos authentication.

In terms of security, there are several drawbacks with digest access authentication:

- Many of the security options in RFC 2617 are optional. If quality-of-protection (qop) is not specified by the server, the client will operate in a security-reduced legacy RFC 2069 mode
- Digest access authentication is vulnerable to a man-in-the-middle (MitM) attack. For example, a MitM attacker could tell clients to use basic access authentication or legacy RFC2069 digest access authentication mode. To extend this further, digest access authentication provides no mechanism for clients to verify the server's identity
- Some servers require passwords to be stored using reversible encryption. However, it is possible to instead store the digested value of the username, realm, and password[4]
- It prevents the use of a strong password hash (such as bcrypt) when storing passwords (since either the password, or the digested username, realm and password must be recoverable)

Also, since the MD5 algorithm is not allowed in FIPS, HTTP Digest authentication will not work with FIPS-certified[note 1] crypto modules.

3.3 Alternative authentication protocols

Some strong authentication protocols for web-based applications include:

- Public key authentication (usually implemented with a HTTPS / SSL client certificate)
- Kerberos or SPNEGO authentication, employed for example by Microsoft IIS running configured for Integrated Windows Authentication (IWA)
- Secure Remote Password protocol (preferably within the HTTPS / TLS layer)

Weak cleartext protocols are also often in use:

- Basic access authentication scheme
- HTTP+HTML form-based authentication

These weak cleartext protocols used together with HTTPS network encryption resolve many of the threats that digest access authentication is designed to prevent.

4 Example with explanation

The following example was originally given in RFC 2617 and is expanded here to show the full text expected for each request and response. Note that only the "auth" (authentication) quality of protection code is covered – as of April 2005, only the Opera and Konqueror web browsers are known to support "auth-int" (authentication with integrity protection). Although the specification mentions HTTP version 1.1, the scheme can be successfully added to a version 1.0 server, as shown here.

This typical transaction consists of the following steps:

1. The client asks for a page that requires authentication but does not provide a username and password.[note 2] Typically this is because the user simply entered the address or followed a link to the page.
2. The server responds with the 401 "Unauthorized" response code, providing the authentication realm and a randomly generated, single-use value called a *nonce*.

3. At this point, the browser will present the authentication realm (typically a description of the computer or system being accessed) to the user and prompt for a username and password. The user may decide to cancel at this point.
4. Once a username and password have been supplied, the client re-sends the same request but adds an authentication header that includes the response code.
5. In this example, the server accepts the authentication and the page is returned. If the username is invalid and/or the password is incorrect, the server might return the "401" response code and the client would prompt the user again.

Client request (no authentication)

GET /dir/index.html HTTP/1.0 Host: localhost

(followed by a new line, in the form of a carriage return followed by a line feed).[5]

Server response

HTTP/1.0 401 Unauthorized Server: HTTPd/0.9 Date: Sun, 10 Apr 2014 20:26:47 GMT WWW-Authenticate: Digest realm="testrealm@host.com", qop="auth,auth-int", nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", opaque="5ccc069c403ebaf9f0171e9517f40e41" Content-Type: text/html Content-Length: 153 <!DOCTYPE html> <html> <head> <meta charset="UTF-8" /> <title>Error</title> </head> <body> <h1>401 Unauthorized.</h1> </body> </html>

Client request (username "Mufasa", password "Circle Of Life")

GET /dir/index.html HTTP/1.0 Host: localhost Authorization: Digest username="Mufasa", realm="testrealm@host.com", nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", uri="/dir/index.html", qop=auth, nc=00000001, cnonce="0a4f113b", response="6629fae49393a05397450978507c4ef1", opaque="5ccc069c403ebaf9f0171e9517f40e41"

(followed by a blank line, as before).

Server response

HTTP/1.0 200 OK Server: HTTPd/0.9 Date: Sun, 10 Apr 2005 20:27:03 GMT Content-Type: text/html Content-Length: 7984

(followed by a blank line and HTML text of the restricted page).

The "response" value is calculated in three steps, as follows. Where values are combined, they are delimited by colons.

1. The MD5 hash of the combined username, authentication realm and password is calculated. The result is referred to as HA1.
2. The MD5 hash of the combined method and digest URI is calculated, e.g. of "GET" and "/dir/index.html". The result is referred to as HA2.
3. The MD5 hash of the combined HA1 result, server nonce (nonce), request counter (nc), client nonce (cnonce), quality of protection code (qop) and HA2 result is calculated. The result is the "response" value provided by the client.

Since the server has the same information as the client, the response can be checked by performing the same calculation. In the example given above the result is formed as follows, where MD5() represents a function used to calculate an MD5 hash, backslashes represent a continuation and the quotes shown are not used in the calculation.

Completing the example given in RFC 2617 gives the following results for each step.

HA1 = MD5("Mufasa:testrealm@host.com:Circle Of Life") = 939e7578ed9e3c518a452acee763bce9 HA2 = MD5("GET:/dir/index.html") = 39aff3a2bab6126f332b942af96d3366 Response = MD5("939e7578ed9e3c518a452acee763bce9:\
dcd98b7102dd2f0e8b11d0f600bfb0c093:\
00000001:0a4f113b:auth:\
39aff3a2bab6126f332b942af96d3366") = 6629fae49393a05397450978507c4ef1

At this point the client may make another request, reusing the server nonce value (the server only issues a new nonce for each "401" response) but providing a new client nonce (cnonce). For subsequent requests, the hexadecimal request counter (nc) must be greater than the last value it used – otherwise an attacker could simply "replay" an old request with the same credentials. It is up to the server to ensure that the counter increases for each of the nonce values that it has issued, rejecting any bad requests appropriately. Obviously changing the method, URI and/or counter value will result in a different response value.

The server should remember nonce values that it has recently generated. It may also remember when each nonce value was issued, expiring them after a certain amount of time. If an expired value is used, the server should respond with the "401" status code and add stale=TRUE to the authentication header, indicating that the client should re-send with the new nonce provided, without prompting the user for another username and password.

The server does not need to keep any expired nonce values – it can simply assume that any unrecognised values have expired. It is also possible for the server to only allow each nonce value to be returned once, although this forces the client to repeat every request. Note that expiring a server nonce immediately will not work, as the client would never get a chance to use it.

5 The .htdigest file

.htdigest is a flat-file used to store usernames, realm and passwords for digest authentication of Apache HTTP Server. The name of the file is given in the .htaccess configuration, and can be anything, but ".htdigest” is the canonical name. The file name starts with a dot, because most Unix-like operating systems consider any file that begins with dot to be hidden. This file is often maintained with the shell command “htdigest” which can add, delete, and update users, and will properly encode the password for use.

The “htdigest” command is found in the **apache2-utils** package on dpkg package management systems and the **httpd-tools** package on RPM package management systems.

The syntax of the htdigest command:[6]

htdigest [-c] *passwdfile realm username*

The format of the .htdigest file:[6]

```
user1:Realm:5ea41921c65387d904834f8403185412
user2:Realm:734418f1e487083dc153890208b79379
```

6 SIP digest authentication

Session Initiation Protocol (SIP) uses basically the same digest authentication algorithm. It is specified by RFC 3261.

7 Browser implementation

Most browsers have substantially implemented the spec, some barring certain features such as auth-int checking or the MD5-sess algorithm. If the server requires that these optional features be handled, clients may not be able to authenticate (though note mod_auth_digest for Apache does not fully implement RFC 2617 either).

- Amaya
- Gecko-based: (not including auth-int[7])
  - Mozilla Application Suite
  - Mozilla Firefox
  - Netscape 7+
- iCab 3.0.3+
- KHTML- and WebKit-based: (not including auth-int[8])
  - iCab 4
  - Konqueror
  - Google Chrome
  - Safari
- Tasman-based:
  - Internet Explorer for Mac
- Trident-based:
  - Internet Explorer 5+[9] (not including auth-int)
- Presto-based:
  - Opera
  - Opera Mobile
  - Opera Mini
  - Nintendo DS Browser
  - Nokia 770 Browser
  - Sony Mylo 1's Browser
  - Wii Internet Channel Browser

8 See also

- AKA (security)
- Basic access authentication

9 Notes

[1] The following is a list of FIPS approved algorithms: “Annex A: Approved Security Functions for FIPS PUB 140-2, Security Requirements for Cryptographic Modules” (PDF). National Institute of Standards and Technology. January 31, 2014.

[2] A client may already have the required username and password without needing to prompt the user, e.g. if they have previously been stored by a web browser.

10 References

[1] “Hash Collision Q&A”. Cryptography Research. 2005-02-16. Archived from the original on 2010-03-06.

[2] Jongsung Kim, Alex Biryukov, Bart Preneel, Seokhie Hong. “On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1” (PDF). IACR.

[3] Scott Stark (2005-10-08). “DIGEST Authentication (4.0.4+)". JBoss.

[4] "HTTP Authentication: Basic and Digest Access Authentication: Storing passwords". IETF. June 1999.

[5] Tim Berners-Lee, Roy Fielding, Henrik Frystyk Nielsen (1996-02-19). "Hypertext Transfer Protocol -- HTTP/1.0: Request". W3C.

[6] "htdigest - manage user files for digest authentication". *apache.org*.

[7] Emanuel Corthay (2002-09-16). "Bug 168942 - Digest authentication with integrity protection". *Mozilla*.

[8] Timothy D. Morgan (2010-01-05). "HTTP Digest Integrity: Another look, in light of recent attacks" (PDF). vsecurity.com.

[9] "TechNet Digest Authentication". August 2013.

11 External links

- RFC 2617
- RFC 2069 (obsolete)

12 Text and image sources, contributors, and licenses

12.1 Text

- **Digest access authentication** *Source:* https://en.wikipedia.org/wiki/Digest_access_authentication?oldid=676647755 *Contributors:* Phoe6, Joy, Jeffq, Ashley Y, Lee J Haywood, Antandrus, RossPatterson, YUL89YYZ, TerraFrost, Dila, Jgfoot, Gpvos, LFaraone, Bsdlogical, Niqueco, Gudeldar, Riki, Andriyko, Fresheneesz, Typhoonhurricane, William Graham, Mcicogni, Gorgonzilla, Cedar101, AchimP, Bluezy, SmackBot, Glvgfz, Ennorehling, Chris the speller, Bluebot, Frap, Christan80, Zazpot, UU, Cybercobra, Blaufish, Maxtheterrible, JForget, Phatom87, Revolus, Supremedalek, TheJosh, Dawnseeker2000, Charles Brooking, Jen2000, Bernd vdB~enwiki, Gwern, =JeffH, Mårten Berglund, Bonadea, Rei-bot, Dawn Bard, Xrobau, WikiLaurent, Robenel, Jmbarton, Jamesmbarton, Wprlh, XLinkBot, Ost316, Steeljack, Addbot, Mabdul, Lightbot, Edoe, Benjamin Lamowski, Mindbuilder, AnomieBOT, Kufudo, PeaceLoveHarmony, VittGam, Javier.eguiluz, Wyverald, EmausBot, Zuse, ZéroBot, ClueBot NG, Adam.tolley, ChrisGualtieri, JYBot, Behnamhasimkhani, Rineez, ArmbrustBot, A jesin, Meteor sandwich yum and Anonymous: 64

12.2 Images

12.3 Content license