SIP Working Group
Internet-Draft
Intended status: Informational
Expires: October 29, 2008

L. Veltri
Univ. of Parma
S. Salsano
A. Polidoro
Univ. of Rome Tor Vergata
April 27, 2008

HTTP digest authentication using alternate password storage schemes
draft-veltri-sip-alt-auth-00

Status of this Memo

Abstract

This document proposes to extend the HTTP Digest Authentication by adding a set new algorithms. These algorithms use different hash functions and combination of various information such as user name, realm, password, salt, and/or other data, in order to achieve compatibility with existing mechanisms used to store user credentials in various authentication/autorization servers.

Table of Contents

1. Introduction 3

2. HTTP Digest Authentication for SIP 4

3. New extensible authentication scheme 8
3.1. First solution 11
3.2. Second solution 13

4. Security considerations 17

5. Informative References 18

Authors' Addresses 19
Intellectual Property and Copyright Statements 20

1. Introduction

According to the current SIP specification [RFC3261], the SIP protocol uses the HTTP Digest Authentication defined in [RFC2617] as default mechanism for authenticating and authorizing User Agent Clients (UACs) against remote User Agent Servers (UASs) or intermediate proxys. HTTP Digest Authentication is a challenge-response authentication method and requires that both the supplicant (i.e. the UAC) and the authenticator (i.e. the UAS or the proxy) access the clear-text user password or a non-revertible function (hash-derived) of the password and other information. The HTTP Digest Authentication [RFC2617] specifically define also an authentication scheme named MD5 that requires the UAs and proxys to compute or retrive from a database the MD5 digest of a concatenation (colon-separated) of username, realm, and password. Unfortunately other user authentication/authorization mechanisms use different and non compatible mechanisms to store non-revertible hashes of passwords. This prevents using an existing database of user credentials to offer SIP based services requiring authentication. This document tries to extend the stantard SIP/HTTP Digest Authentication mechanism in order to consider other password-storing schemes that do not naturally cooperate with the current HTTP Digest Authentication scheme. Examples of such password-storing schemes are those generally used in LDAP servers, Unix shadow/password files, Apache's htpasswd file, or SQL-based storage systems used by other specific applications.

2. HTTP Digest Authentication for SIP

HTTP Digest Authentication [RFC2617] is a general challenge-response mechanism in which a UAS authenticates a UAC based on a shared secret. The challenge response is computed through the use of a one-way function based on various user's credentials. The standard also specifies the use of a particular function (in turn based on the MD5 hash function) that requires that both the client and server knows the user secret (normally a password) or, at least, the hash of the concatenation of the user name, the realm, and the password.

When the HTTP Digest Authentication [RFC2617] is used in SIP, an UAS that receives a SIP request (example a REGISTER) may challenge the UAC by sending a 401 "Unauthorized" error response (or 407 "Proxy Authentication Required" for proxy authentication) containing a fresh random nonce value as challenge. Both the UAC and the UAS share a secret (usually a password) and they use this secret, together with the nonce value, realm, and other information, respectively to compute the challenge response (the UAC) and to verify such the response (the UAS). The UAS sends the challenge in the 401 "Unauthorized" SIP response within a WWW-Authenticate header field (Proxy-Authenticate header for proxy authentication), then the UAC sends the challenge response in a new request message within a Authorization header field (Proxy-Authorization header for proxy authentication), and finally the UAS sends the authentication and authorization result with a new response message (2xx if it successed).

According to [RFC2617] the challenge response is computed as:

response = KD(H(A1), nonce:nc:cnonce:qop:H(method:uri))

where KD(secret,data) is a general two-parameters digest algorithm applied to "data" with secret "secret", while H(data) is a digest algorithm applied to "data", and A1 is a quantity that should take into account user credentials.

As specified in [RFC2617], by default or in case of "algorithm=MD5, the KD(,) H(), and A1 are respectively:

KD(secret, data) = MD5(secret:data)

H(data) = MD5(data)

A1 = username:realm:passwd

Particularly, the first term of KD(,) (indicated as "secret") and computed as H(A1) becomes:

secret = H(A1) = MD5(username:realm:passwd)

Note that both the requestor (the UAC) and the auhenticator (the UAS) do not need to know the user password (eventually retrieved from a local archive), but rather this "secret" i.e. a digest function of username, realm and password itself.

Unfortunately, current user's credential databases or storage systems often protect user's password by implementing one-way cryptographic algorithms different from the MD5 hashing mechanism specified for the HTTP Digest Authentication.

For example, in LDAP (Lightweight Directory Access Protocol) servers, password values can be stored as plaintext or as one of a variety of hashes. [RFC3112] specifically describes the "MD5" and "SHA1" schemes for a LDAP directory. The MD5 scheme computes the hashed-protected password as the base64 encoding of an MD5 [RFC1321] digest of the concatenation the user password and salt that must be at least 64 bit long. The SHA1 scheme computes the hashed-protected password in the same way, by using SHA1 [RFC3174] hash function instead of MD5.

These two hash-protected password schemes are also referred as:

- SSHA - Salted SHA-1 based hash
- SMD5 - Salted MD5 based hash

Other hash-protected password schemes normally supported by an LDAP server are:

- SHA - SHA-1 based hash.
- MD5 - MD5 based hash.
- CRYPT - Unix crypt() hash, based on DES, also referred as UNIX; see later.

Note that these three schemes are weaker than the previous two, due to the absence of a security salt value. In some lucky cases a LDAP server may also support other schemes that use pre-calculated hash values compatible with HTTP Digest Authentication.

An other example is the Unix system in which passwords are usually stored in the "/etc/shadow" file or, in older Unix versions, in the "/etc/password" file. In both cases, passwords are normally stored encrypted (actually hashed) with a one-way algorithm generally referred as "crypt", together with a salt value and an indication of

the used algorithm. The traditional crypt algorithm implementation uses a modified form of the DES algorithm that performs 25 DES passes to encrypt an all-bits-zero block using with a 56-bit key formed by the first 7 password characters. A 12-bit salt is used to perturb the original DES algorithm. The salt and the final ciphertext are base64-encoded into a printable string stored in the password or shadow file. Other more robust crypt functions have been defined based on other cryptographic or hash algorithms such as MD5, blowfish, or SHA-1. Such functions generally allow users to have any length password (> 8bytes), and do not limit the password to ASCII (7-bit) text. Currently, the most common crypt function used by Unix/Linux systems supports both the original DES-based and the MD5-based (MD5-crypt) algorithms. The MD5-crypt function is really not a straight implementation of MD5: first the password and salt are MD5 hashed together in a first digest, then 1000 iteration loops continuously remix the password, salt and intermediate digest values; the output of the last of these rounds is the resulting hash. A typical output of the stored password together with username, salt, and other information is:

alice:1BZftq3sP$xEeZmr2fGEnKjVAxzj:12747:0:99999:7:::

where 1 indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAxzjQo68 is the password hash.

Other applications also store user's credentials in local files or database, that have they own format but that re-use similar hashing algorithms. For example the Apache's htpasswd tool supports four different methods:

PLAINTEXT: passwords are stored without any encryption mechanism. In this case in the file will contain lines of the form: user: passwd

CRYPT: passwords are stored encrypted using the traditional Unix crypt function described in the previous section.

SHA1: passwords are stored by base64-encoding the SHA-1 digest of the password. The corresponding htpasswd file has lines that look like:

alice:{SHA1}VBPuJHI7uixaa6LQGWx4s+5GKNE=

where VBPuJHI7uixaa6LQGWx4s+5GKNE= is the base-64 hash of the password.

MD5: passwords are stored encrypted through the MD5-crypt function described in the previous section using an Apache-modified version of MD5. An example of a corresponding htpasswd line is:

alice:$apr1$r31.....$HqJZimcKQFAMYayBlzkrA/

where $apr1$ indicates the use of the Apache's MD5-crypt function, followed by the salt and the effective password hash.

3. New extensible authentication scheme

The HTTP Digest Authentication [RFC2617] requires that the response to the challenge, regardless of the selected algorithm, is in the form of:

response = KD(H(A1),nonce:nc:cnonce:qop:H(method:uri))

In case of "algorithm=MD5", KD(secret,data) is MD5(secret:data), H(data) is MD5(data), while the H(A1) becomes MD5(username:realm: passwd).

Assuming for the moment that we have no interest in changing the KD function, we can envisage two possible solutions:

a. we can define a different A1 and H(A1) function compatible with the specific authentication system (A1 is a formatted set of parameters that are taken into account by the H() function). For example A1 could be the concatenation passwd:salt and H(A1) could become: H(A1) = MD5(passwd:salt)

b. we can reuse the definition of H(A1) and only replace the password parameter with a new derived pseudo-password A3 defined as A3=KP(password,other-params). In this case we are introducing a new function KP() with a new set of parameters that includes the user password. The value A1 becomes: A1=username:realm:A3.

For example, if we chose KP() as MD5(passwd:salt) the value H(A1) becomes:

H(A1) = MD5(username:realm:A3), i.e.:

H(A1) = MD5(username:realm:MD5(passwd:salt))

The first solution (a) has the advantage that it may save some computation of crypto functions. The second option (b) has the advantage that it inherits all the security properties of the current MD5 solution. Moreover one could store in the client the derived password (i.e. the A3 value) instead of the password and maintain a compatibility with existing clients. We are here referring to the approach of several SIP user agents to store the SIP user password rather then requesting it to the user each time. During this "one-time" password storing operation, the A3 value could be computed externally and then manually stored in the SIP client. Of course this is not our suggested solution, i.e. we believe that SIP stacks should be enhanced with the proposed solution so that SIP UA can natively support the new authentication method, anyway it was worth mentioning this possibility to reuse legacy clients in the short

term.

Once that we have chosen which solution for the algorithm, we should discuss:

1. how to introduce the indication of this different authentication algorithms within SIP signaling;

2. how to convey the parameters that are needed by the new authentication algorithms.

Again, we introduce two different options concerning issue 1):

i introduce new algorithms specified as parameter "algorithm" within authentication headers. We could have for example algorithm=md5-ldap-sha1, algorithm=md5-crypt-des and so on.

ii introduce a new authentication parameter called "pwd-algo" in order to indicate the chosen algorithm used to compute only the derived-passwd A3.

These two choices are not completely independent from the choice of the algorithm definition, as shown in the following table.

	i)introduce new values for the algorithm parameter	ii)introduce a new pwd-algo parameter
a)definition of a new A1 and H(A1)	Yes	No
b)reuse H(A1) and include a new A3=KP(pwd,params) in place of the passwd value	Possible	Suggested

Figure 1: Alternatives

If we define new A1 and H(A1) this should be signaled by introducing new values for the algorithm parameter. At the contrary, if we reuse H(A1) and introduce A3=KP() in place of the clear password, both options for the signaling are feasible, but we think that keeping the algorithm parameter unchanged (that specify A1, H(), and KD()) and introducing a new pwd-algo is preferable.

Now we finally need to discuss how to convey the additional parameter that may be needed by the different authentication mechanisms. We think that three options are possible:

1. introduce new parameters with specific names for each authentication algorithm;

2. introduce a generic parameter named pwd-param to carry algorithm specific parameters;

3. carry the new parameters added inside the nonce parameter. This is the approach that has been followed for example in the specification of the AKA authentication mechanism.

We believe that 1) is the worst solution as it may lead to add several new parameters. 2) and 3) are both feasible, where 2) is a "cleaner" approach that requires the definition of an additional parameter, while 3) has the advantage that does not require any new parameter. Considering the various discussed options, we believe that a first possible solution is based on:

a. reuse of H(A1) with a modified version of A1 in which the password value is simply replaced by A3 = KP(password,other-params)

b. introduction of new "pwd-algo" parameter

c. introduction of a new generic "pwd-param" parameter

Examples of the new pwd-algo parameter are:

pwd-algo= ssha

pwd-algo= crypt-md5

pwd-algo= crypt-apache

A second possible solution, more conservative from the point of view of SIP signaling is the use of A3 = KP(password,other-params) as above, but specifying the chosen algorithm in the existing "algorithm" parameter and carrying the algorithm specific parameters within the "nonce" parameter. Examples of the "algorithm" parameter are:

algorithm=md5-pwd-hashed

algorithm=sha1-pwd-salt-hashed

When we need to specify variants of the algorithm we think that a simple and efficient solution is to carry the name of the variant into the eventual pwd-param or as part of the nonce value.

3.1. First solution

In the first solution we define two new parameters as follows:

pwd-algo = "pwd-algo" "=" ("plain" | token)

pwd-param = "pwd-param" "=" quoted-string

where "pwd-algo" specifies the function KP() used to compute the derived-password A3, while "pwd-param" has a completely opaque value, depending on the particular selected function KP, indicating the values of the (eventual) parameters used in KP() computation. A "pwd-algo=plain" value should indicate that none algorithm has to be used, and hence A3=password. This corresponds to the case in which no pwd-algo parameter is present, as in case of standard MD5-based Digest Authentication. Examples of such parameters are:

pwd-algo=crypt-des, pwd-param="fzwhEV6E"

pwd-algo=alternative, pwd-param="12"

pwd-algo=plain

Particularly, in order to support current LDAP, Unix-based, and other storing mechanisms, the following new values are defined:

pwd-algo = "pwd-algo" "=" ("plain" | "ssha" | "smd5" | "sha" | "md5" | crypt-algo | token)

crypt-algo = "crypt-" crypt-hash

crypt-hash = "des" | "md5" | "blowfish" | "apache" | token

pwd-param = "pwd-param" "=" LDQUOT salt-value RDQUOT

salt-value = <base-64 encoding of the salt value>

In case of ssha or smd5 or crypt-XXX, the A3 value is computed as follows:

A3 = KP(password,salt) = H(password||salt)

where H() is respectively SHA1, MD5, or the the Unix crypt function.

Instead, in the remaining cases:

A3 = KP(password) = H(password)

In case of ssha or smd5 or crypt-XXX, the pwd-param will contain the base-64 encoding of the salt value.

Examples of use of such parameters within a SIP transaction are:

INVITE sip:bob@neverland.net SIP/2.0

To: sip:bob@neverland.net

From: sip:alice@wonderland.net

[...]

SIP/2.0 401 Unauthorized

To: sip:bob@neverland.net

From: sip:alice@wonderland.net

WWW-Authenticate: Digest realm="example.com",

nonce="cc5a61b2954e03541847f227102f",

qop="auth", algorithm="MD5", pwd-algo="crypt-md5",

pwd-param="fzwhEV6E"

[...]

INVITE sip:bob@neverland.net SIP/2.0

To: sip:bob@neverland.net

From: sip:alice@wonderland.net

Authorization: Digest username="alice", realm="example.com",

nonce="cc5a61b2954e03541847f227102f",

pwd-algo="crypt-md5", pwd-param="MD5-fzwhEV6E"

response="587410ee2dc5edd9bbe9370ddc1fA3a1",

uri="sip:bob@neverland.net", qop="auth", nc="00000001"

cnonce="226827CAD1C949A18B17FD71EC68"

[...]

SIP/2.0 200 OK

To: sip:bob@neverland.net

From: sip:alice@wonderland.net

[...]

3.2. Second solution

The second solution does not require any new authentication parameters since both the selected function KP() (used to generate the new "password" value A3 used in A1) and the eventual parameters of KP() are indicated respectively in the standard algorithm and nonce authentication parameters. According with this solution, the "algorithm" parameter defined in [RFC3261] can be extended as follows:

algorithm = "algorithm" EQUAL (algorithm-value | new-algorithm)

algorithm-value = "MD5" | "MD5-sess" | token

where "new-algorithm" is a new algorithm name that completely specifies the KD(), H(), A1, and KP() functions for the new authentication scheme. For example, in order to support authentication against a server with Unix-based password archive, we could define:

new-algorithm = crypt-algorithm

crypt-algorithm = algorithm-value "-crypt"

Examples of the use of "algorithm" parameter are:

algorithm=MD5

algorithm=MD5-crypt

In case of KP() function requires additional parameters such as sub-algorithm type, salt, etc, their values will be included within the nonce parameter, in a form named compound-nonce. According to [RFC3261], the "nonce" parameter can be extended as follows:

nonce = "nonce" EQUAL (nonce-value | compound-nonce)

compound-nonce = LDQUOT compound-nonce-value RDQUOT

compound-nonce-value = algo-param ":" nonce-value

algo-param = *(unreserved | algo-mark)

algo-mark = ";" | "/" | "?" | "@" | "&" | "=" | "+" | "$" | ","

Examples of the use of nonce parameter are:

nonce="cc5a61b2954e0354184"

nonce=":cc5a61b2954e0354184"

nonce="1BZftq3sP:cc5a61b2954e0354184"

Particularly, in order to support current LDAP, Unix-based, and other storing mechanisms, the following new values are defined:

algorithm = "algorithm" EQUAL (algorithm-value | new-algorithm)

new-algorithm = algorithm-value ("-pwd-hashed" | "-pwd-salt-hashed")

In this case, the nonce parameter will contains indication of both password hash algorithm and salt, together with the actual nonce value; that is:

nonce = "nonce" EQUAL (nonce-value | compound-nonce)

compound-nonce = LDQUOT compound-nonce-value RDQUOT

compound-nonce-value = (salted-hash-algo | hash-algo) ":" nonce-value

compound-nonce-value = (crypt-algo-nonce) ":" nonce-value

hash-algo = "sha" | "md5" | "des" | "blowfish" | "apache" | token

salted-hash-algo = (hash-algo | crypt-algo) "-" salt-value

crypt-algo = "crypt-" hash-algo

salt-value = <base-64 encoding of the salt value>.

Examples of use of such parameters within a SIP transaction are:

```
INVITE sip:bob@neverland.net SIP/2.0
To: sip:bob@neverland.net
From: sip:alice@wonderland.net
[...]
SIP/2.0 401 Unauthorized
To: sip:bob@neverland.net
From: sip:alice@wonderland.net
WWW-Authenticate: Digest realm="example.com",
nonce="crypt-des-fzwhEV6E:cc5a61b2954e03541847f2",
qop="auth", algorithm="MD5-crypt"
[...]
INVITE sip:bob@neverland.net SIP/2.0
To: sip:bob@neverland.net
From: sip:alice@wonderland.net
Authorization: Digest username="alice", realm="example.com",
nonce="crypt-des-fzwhEV6E:cc5a61b2954e03541847f2",
response="587410ee2dc5edd9bbe9370ddc1fA3a1",
uri="sip:bob@neverland.net", qop="auth", nc="00000001"
cnonce="226827CAD1C949A18B17FD71EC68"
[...]
SIP/2.0 200 OK
To: sip:bob@neverland.net
From: sip:alice@wonderland.net
[...]
```

We believe that this second solution is to be preferred to the one presented in the previous sub-section, as it does not require addition of new parameters. This is the same approach that has been followed when defining the SIP-AKA authentication mechanism [RFC3310].

4. Security considerations

Put security considerations here

5. Informative References

[RFC3261] J. Rosenberg et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC2617] J. Franks et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2401] "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC3310] "Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA)", RFC 3310, September 2002.

[RFC3112] "LDAP Authentication Password Schema", RFC 3112, May 2001.

[RFC1321] "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC3174] "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

Authors' Addresses

Luca Veltri
DII, University of Parma
Viale delle Scienze 181/A
Parma 43100
Italy

Phone: +39 0521 90 5768
Email: luca.veltri@unipr.it
URI: http://www.tlc.unipr.it/veltri

Stefano Salsano
DIE, University of Rome "TorVergata"
Via Politecnico, 1
Rome 00133
Italy

Phone: +39 06 7259 7770
Email: stefano.salsano@uniroma2.it
URI: http://netgroup.uniroma2.it/Stefano_Salsano

Andrea Polidoro
DIE, University of Rome "TorVergata"
Via Politecnico, 1
Rome 00133
Italy

Phone: +39 06 7259 7773
Email: andrea.polidoro@uniroma2.it
URI: http://netgroup.uniroma2.it

Full Copyright Statement

Intellectual Property