        An Extension to HTTP : Digest Access Authentication

Status of this Memo

Abstract

   The protocol referred to as "HTTP/1.0" includes the specification for
   a Basic Access Authentication scheme. This scheme is not considered
   to be a secure method of user authentication, as the user name and
   password are passed over the network as clear text. A specification
   for a different authentication scheme is needed to address this
   severe limitation. This document provides specification for such a
   scheme, referred to as "Digest Access Authentication". Like Basic,
   Digest access authentication verifies that both parties to a
   communication know a shared secret (a password); unlike Basic, this
   verification can be done without sending the password in the clear,
   which is Basic's biggest weakness. As with most other authentication
   protocols, the greatest sources of risks are usually found not in the
   core protocol itself but in policies and procedures surrounding its
   use.

Table of Contents

Introduction

1.1 Purpose

    The protocol referred to as "HTTP/1.0" includes specification for a
    Basic Access Authentication scheme[1]. This scheme is not considered
    to be a secure method of user authentication, as the user name and
    password are passed over the network in an unencrypted form. A
    specification for a new authentication scheme is needed for future
    versions of the HTTP protocol. This document provides specification
    for such a scheme, referred to as "Digest Access Authentication".

    The Digest Access Authentication scheme is not intended to be a
    complete answer to the need for security in the World Wide Web. This
    scheme provides no encryption of object content. The intent is simply
    to create a weak access authentication method which avoids the most
    serious flaws of Basic authentication.

    It is proposed that this access authentication scheme be included in
    the proposed HTTP/1.1 specification.

1.2 Overall Operation

   Like Basic Access Authentication, the Digest scheme is based on a
   simple challenge-response paradigm. The Digest scheme challenges
   using a nonce value. A valid response contains a checksum (by
   default the MD5 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. Just as with the Basic
   scheme, the username and password must be prearranged in some fashion
   which is not addressed by this document.

1.3 Representation of digest values

   An optional header allows the server to specify the algorithm used to
   create the checksum or digest. By default the MD5 algorithm is used
   and that is the only algorithm described in this document.

   For the purposes of this document, an MD5 digest of 128 bits is
   represented as 32 ASCII printable characters. The bits in the 128
   bit digest are converted from most significant to least significant
   bit, four bits at a time to their ASCII presentation as follows.
   Each four bits is represented by its familiar hexadecimal notation
   from the characters 0123456789abcdef. That is, binary 0000 gets
   represented by the character '0', 0001, by '1', and so on up to the
   representation of 1111 as 'f'.

1.4 Limitations

   The digest authentication scheme described in this document suffers
   from many known limitations. It is intended as a replacement for
   basic authentication and nothing more. It is a password-based system
   and (on the server side) suffers from all the same problems of any
   password system. In particular, no provision is made in this
   protocol for the initial secure arrangement between user and server
   to establish the user's password.

   Users and implementors should be aware that this protocol is not as
   secure as kerberos, and not as secure as any client-side private-key
   scheme. Nevertheless it is better than nothing, better than what is
   commonly used with telnet and ftp, and better than Basic
   authentication.

2. Digest Access Authentication Scheme

2.1 Specification of Digest Headers

   The Digest Access Authentication scheme is conceptually similar to
   the Basic scheme. The formats of the modified WWW-Authenticate

header line and the Authorization header line are specified below,
using the extended BNF defined in the HTTP/1.1 specification, section
2.1. In addition, a new header, Authentication-info, is specified.

2.1.1 The WWW-Authenticate Response Header

If a server receives a request for an access-protected object, and an
acceptable Authorization header is not sent, the server responds with
a "401 Unauthorized" status code, and a WWW-Authenticate header,
which is defined as follows:

```
    WWW-Authenticate    = "WWW-Authenticate" ":" "Digest"
                            digest-challenge

    digest-challenge    = 1#( realm | [ domain ] | nonce |
                         [ digest-opaque ] |[ stale ] | [ algorithm ] )

    realm               = "realm" "=" realm-value
    realm-value         = quoted-string
    domain              = "domain" "=" <"> 1#URI <">
    nonce               = "nonce" "=" nonce-value
    nonce-value         = quoted-string
    opaque              = "opaque" "=" quoted-string
    stale               = "stale" "=" ( "true" | "false" )
    algorithm           = "algorithm" "=" ( "MD5" | token )
```

The meanings of the values of the parameters used above are as
follows:

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". The realm is a
"quoted-string" as specified in section 2.2 of the HTTP/1.1
specification [2].

domain
A comma-separated list of URIs, as specified for HTTP/1.0. The
intent is that the client could use this information to know the
set of URIs for which the same authentication information should be
sent. The URIs in this list may exist on different servers. If
this keyword is omitted or empty, the client should assume that the
domain consists of all URIs on the responding server.

nonce
A server-specified data string which may 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 nonce are implementation dependent. The
quality of the implementation depends on a good choice. A
recommended nonce would include

        H(client-IP ":" time-stamp ":" private-key)

Where client-IP is the dotted quad IP address of the client making
the request, time-stamp is a server-generated time value, private-
key is data known only to the server. With a nonce of this form a
server would normally recalculate the nonce after receiving the
client authentication header and reject the request if it did not
match the nonce from that header. In this way the server can limit
the reuse of a nonce to the IP address to which it was issued and
limit the time of the nonce's validity. Further discussion of the
rationale for nonce construction is in section 3.2 below.

An implementation might choose not to accept a previously used
nonce or a previously used digest 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 3.
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. It is recommended that this
string be base64 or hexadecimal data. This field is a
"quoted-string" as specified in section 2.2 of the HTTP/1.1
specification [2].

stale
A flag, indicating that the previous request from the client was
rejected because the nonce value was stale. If stale is TRUE (in
upper or lower case), the client may wish to simply retry the
request with a new encrypted response, without reprompting 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 but with a valid digest for that nonce (indicating that
the client knows the correct username/password).

       algorithm
       A string indicating a pair of algorithms used to produce the
       digest and a checksum. If this not present it is assumed to be
       "MD5". 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).

       For the "MD5" algorithm

          H(data) = MD5(data)

       and

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

    i.e., the digest is the MD5 of the secret concatenated with a colon
    concatenated with the data.

2.1.2 The Authorization Request Header

   The client is expected to retry the request, passing an Authorization
   header line, which is defined as follows.

Authorization   = "Authorization" ":" "Digest" digest-response

digest-response   = 1#(username | realm | nonce | digest-uri |
                           response | [digest] | [algorithm] |
                           opaque)

username   = "username" "=" username-value
username-value   = quoted-string
digest-uri   = "uri" "=" digest-uri-value
digest-uri-value   = request-uri   ; As specified by HTTP/1.1
response   = "response" "=" response-digest
digest   = "digest" "=" entity-digest

response-digest   = <"> *LHEX <">
entity-digest   = <"> *LHEX <">
LHEX   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                        "8" | "9" | "a" | "b" | "c" | "d" | "e" | "f"

   The definitions of response-digest and entity-digest above indicate
   the encoding for their values. The following definitions show how the
   value is computed:

```
   response-digest     =
       <"> < KD ( H(A1), unquoted nonce-value ":" H(A2) > <">

   A1              = unquoted username-value ":" unquoted realm-value
                                              ":" password
   password        = < user's password >
   A2              = Method ":" digest-uri-value
```

The "username-value" field is a "quoted-string" as specified in
section 2.2 of the HTTP/1.1 specification [2]. However, the
surrounding quotation marks are removed in forming the string A1.
Thus if the Authorization header includes the fields

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

and the user Mufasa has password "CircleOfLife" then H(A1) would be
H(Mufasa:myhost@testrealm.com:CircleOfLife) 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 in the illustrated above must
be Mufasa:myhost@testrealm.com:CircleOfLife with no white space on
either side of the colons. 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.

"Method" is the HTTP request method as specified in section 5.1 of
[2]. The "request-uri" value is the Request-URI from the request
line as specified in section 5.1 of [2]. This may be "*", an
"absoluteURL" or an "abs_path" as specified in section 5.1.2 of [2],
but it MUST agree with the Request-URI. In particular, it MUST be an
"absoluteURL" if the Request-URI is an "absoluteURL".

The authenticating server must assure that the document designated by
the "uri" parameter is the same as the document served. 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. This altered (but presumably semantically
equivalent) request would not result in the same digest as that
calculated by the client.

The optional "digest" field contains a digest of the entity body and
some of the associated entity headers. This digest can be useful in
both request and response transactions. In a request it can insure
the integrity of POST data or data being PUT to the server. In a

response it insures the integrity of the served document. The value
of the "digest" field is an <entity-digest> which is defined as
follows.

```
entity-digest = <"> KD (H(A1), unquoted nonce-value ":" Method ":"
                            date ":" entity-info ":" H(entity-body)) <">
      ; format is <"> *LHEX <">

date = = rfc1123-date              ; see section 3.3.1 of [2]
entity-info = H(
          digest-uri-value ":"
          media-type ":"          ; Content-type, see section 3.7 of [2]
          *DIGIT ":"              ; Content length, see 10.12 of [2]
          content-coding ":"      ; Content-encoding, see 3.5 of [2]
          last-modified ":"       ; last modified date, see 10.25 of [2]
          expires                 ; expiration date; see 10.19 of [2]
          )

last-modified   = rfc1123-date  ; see section 3.3.1 of [2]
expires         = rfc1123-date
```

The entity-info elements incorporate the values of the URI used to
request the entity as well as the associated entity headers Content-
type, Content-length, Content-encoding, Last-modified, and Expires.
These headers are all end-to-end headers (see section 13.5.1 of [2])
which must not be modified by proxy caches. The "entity-body" is as
specified by section 10.13 of [2] or RFC 1864.

Note that not all entities will have an associated URI or all of
these headers. For example, an entity which is the data of a POST
request will typically not have a digest-uri-value or Last-modified
or Expires headers. If an entity does not have a digest-uri-value or
a header corresponding to one of the entity-info fields, then that
field is left empty in the computation of entity-info. All the
colons specified above are present, however. For example the value
of the entity-info associated with POST data which has content-type
"text/plain", no content-encoding and a length of 255 bytes would be
H(:text/plain:255:::). Similarly a request may not have a "Date"
header. In this case the date field of the entity-digest should be
empty.

In the entity-info and entity-digest computations, except for the
blank after the comma in "rfc1123-date", there must be no white space
between "words" and "tspecials", and exactly one blank between
"words" (see section 2.2 of [2]).

Implementors should be aware of how authenticated transactions
interact with proxy caches. The HTTP/1.1 protocol specifies that
when a shared cache (see section 13.10 of [2]) has received a request
containing an Authorization header 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 14.9 of [2])
directives 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 headers 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.

2.1.3 The AuthenticationInfo Header

When authentication succeeds, the Server may optionally provide a
Authentication-info header indicating that the server wants to
communicate some information regarding the successful authentication
(such as an entity digest or a new nonce to be used for the next
transaction). It has two fields, digest and nextnonce. Both are
optional.

    AuthenticationInfo = "Authentication-info" ":"
                                      1#(digest | nextnonce)

    nextnonce   = "nextnonce" "=" nonce-value

    digest = "digest" "=" entity-digest

The optional digest allows the client to verify that the body of the
response has not been changed en-route. The server would probably
only send this when it has the document and can compute it. The
server would probably not bother generating this header for CGI
output. The value of the "digest" is an <entity-digest> which is
computed as described above.

The value of the nextnonce parameter is the nonce the server wishes
the client to use for the next authentication response. Note that
either field is optional. In particular the server may send the
Authentication-info header with only the nextnonce field as a means
of implementing one-time nonces. If the nextnonce field is present
the client is strongly encouraged to use it for the next WWW-
Authenticate header. Failure of the client to do so may result in a
request to re-authenticate from the server with the "stale=TRUE."

2.2 Digest Operation

   Upon receiving the Authorization header, the server may check its
   validity by looking up its known password which corresponds to the
   submitted username. Then, the server must perform the same MD5
   operation performed by the client, and compare the result to the
   given response-digest.

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

   A client may remember the username, password and nonce values, so
   that future requests within the specified <domain> may include the
   Authorization header preemptively. The server may choose to accept
   the old Authorization header information, even though the nonce value
   included might not be fresh. Alternatively, the server could 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
   hints to the client that the request should be retried with the new
   nonce, without reprompting the user for a new username and password.

   The opaque data is useful for transporting state information around.
   For example, a server could be responsible for authenticating content
   which actually sits on another server. The first 401 response would
   include a domain field which includes the URI on the second server,
   and the opaque field for specifying state information. The client
   will retry the request, at which time the server may respond with a
   301/302 redirection, pointing to the URI on the second server. The
   client will follow the redirection, and pass the same Authorization
   header, including the <opaque> data which the second server may
   require.

   As with the basic scheme, proxies must be completely transparent in
   the Digest access authentication scheme. That is, they must forward
   the WWW-Authenticate, Authentication-info and Authorization headers
   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 headers described in section 2.5
   below.

2.3 Security Protocol Negotiation

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

   If this proposal is accepted as a required part of the HTTP/1.1
   specification, then a server may assume Digest support when a client

      identifies itself as HTTP/1.1 compliant.

      It is possible that a server may want 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 any authentication scheme it cannot handle.

2.4 Example

      The following example assumes that an access-protected document is
      being requested from the server. The URI of the document is
      "http://www.nowhere.org/dir/index.html". Both client and server know
      that the username for this document is "Mufasa", and the password is
      "CircleOfLife".

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

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest     realm="testrealm@host.com",
                             nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                             opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

   The client may prompt the user for the username and password, after
   which it will respond with a new request, including the following
   Authorization header:

```
Authorization: Digest        username="Mufasa",
                             realm="testrealm@host.com",
                             nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                             uri="/dir/index.html",
                             response="e966c932a9242554e42c8ee200cec7f6",
                             opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

2.5 Proxy-Authentication and Proxy-Authorization

      The digest authentication scheme may also be used for authenticating
      users to proxies, proxies to proxies, or proxies to end servers by
      use of the Proxy-Authenticate and Proxy-Authorization headers. These
      headers are instances of the general Proxy-Authenticate and Proxy-
      Authorization headers specified in sections 10.30 and 10.31 of the
      HTTP/1.1 specification [2] 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 "HTTP/1.1 401 Unauthorized" header followed by a
      "Proxy-Authenticate" header of the form

```
   Proxy-Authentication       = "Proxy-Authentication" ":" "Digest"
                                 digest-challenge
```

   where digest-challenge is as defined above in section 2.1. The
   client/proxy must then re-issue the request with a Proxy-Authenticate
   header of the form

```
     Proxy-Authorization      = "Proxy-Authorization" ":"
                                  digest-response
```

   where digest-response is as defined above in section 2.1. When
   authentication succeeds, the Server may optionally provide a Proxy-
   Authentication-info header of the form

Proxy-Authentication-info = "Proxy-Authentication-info" ":" nextnonce

   where nextnonce has the same semantics as the nextnonce field in the
   Authentication-info header described above in section 2.1.

   Note that in principle a client could be asked to authenticate itself
   to both a proxy and an end-server. It might receive an "HTTP/1.1 401
   Unauthorized" header followed by both a WWW-Authenticate and a
   Proxy-Authenticate header. However, it can never receive more than
   one Proxy-Authenticate header since such headers are only for
   immediate connections and must not be passed on by proxies. If the
   client receives both headers, it must respond with both the
   Authorization and Proxy-Authorization headers as described above,
   which will likely involve different combinations of username,
   password, nonce, etc.

3. Security Considerations

   Digest Authentication does not provide a strong authentication
   mechanism. That is not its intent. It is intended solely to replace
   a much weaker and even more dangerous authentication mechanism: Basic
   Authentication. An important design constraint is that the new
   authentication scheme be free of patent and export restrictions.

   Most needs for secure HTTP transactions cannot be met by Digest
   Authentication. For those needs SSL or SHTTP are more appropriate
   protocols. In particular digest authentication cannot be used for
   any transaction requiring encrypted content. Nevertheless many
   functions remain for which digest authentication is both useful and
   appropriate.

3.1 Comparison with Basic Authentication

   Both Digest and Basic Authentication are very much on the weak end of
   the security strength spectrum. But a comparison between the two
   points out the utility, even necessity, of replacing Basic by Digest.

   The greatest threat to the type of transactions for which these
   protocols are used is network snooping. This kind of transaction
   might involve, for example, online access to a database whose use is
   restricted to paying subscribers. With Basic authentication an
   eavesdropper can obtain the password of the user. This not only
   permits him to access anything in the database, but, often worse,
   will permit access to anything else the user protects with the same
   password.

   By contrast, with Digest Authentication the eavesdropper only gets
   access to the transaction in question and not to the user's password.
   The information gained by the eavesdropper would permit a replay
   attack, but only with a request for the same document, and even that
   might be difficult.

3.2 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 response 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. A GET request containing form data could only be "replayed"
   with the identical data. However, this could be problematic if it
   caused a CGI script to take some action on the server.

   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, 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 response digests which will not
be honored for a second use. This requires the overhead of the
server remembering which digests have been used until the nonce
time-stamp (and hence the digest built with it) has expired, but it
effectively protects against replay attacks. Instead of maintaining a
list of the values of used digests, a server would hash these values
and require re-authentication whenever a hash collision occurs.

An implementation must give special attention to the possibility of
replay attacks with POST and PUT requests. A successful replay
attack could result in counterfeit form data or a counterfeit version
of a PUT file. The use of one-time digests or one-time nonces is
recommended. It is also recommended that the optional <digest> be
implemented for use with POST or PUT requests to assure the integrity
of the posted data. Alternatively, a server may choose to allow
digest authentication only with GET requests. Responsible server
implementors will document the risks described here as they pertain
to a given implementation.

3.3 Man in the Middle

Both Basic and Digest authentication are vulnerable to "man in the
middle" attacks, for example, from a hostile or compromised proxy.
Clearly, this would present all the problems of eavesdropping. But
it could also offer some additional threats.

A simple but effective attack would be to replace the Digest
challenge with a Basic challenge, to spoof the client into revealing
their password. To protect against this attack, clients should
remember if a site has used Digest authentication in the past, and
warn the user if the site stops using it. It might also be a good
idea for the browser 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.

There are several attacks on the "digest" field in the
Authentication-info header. A simple but effective attack is just to
remove the field, so that the client will not be able to use it to
detect modifications to the response entity. Sensitive applications
may wish to allow configuration to require that the digest field be
present when appropriate. More subtly, the attacker can alter any of
the entity-headers not incorporated in the computation of the digest,
The attacker can alter most of the request headers in the client's

request, and can alter any response header in the origin-server's
reply, except those headers whose values are incorporated into the
"digest" field.

Alteration of Accept* or User-Agent request headers can only result
in a denial of service attack that returns content in an unacceptable
media type or language. Alteration of cache control headers also can
only result in denial of service. Alteration of Host will be
detected, if the full URL is in the response-digest. Alteration of
Referer or From is not important, as these are only hints.

3.4 Spoofing by Counterfeit Servers

Basic Authentication is vulnerable to spoofing by counterfeit
servers. If a user can be led to believe that she is connecting to a
host containing information protected by a password she knows, when
in fact she is connecting to a hostile server, then the hostile
server can request a password, store it away for later use, and feign
an error. This type of attack is more difficult with Digest
Authentication -- but the client must know to demand that Digest
authentication be used, perhaps using some of the techniques
described above to counter "man-in-the-middle" attacks.

3.5 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.

3.6 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 many, 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 while others may satisfied with a nonce like
the one recommended above restricted to a single IP address and 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.

4. Acknowledgments

In addition to the authors, valuable discussion instrumental in
creating this document has come from Peter J. Churchyard, Ned Freed,
and David M. Kristol.

5. References

   [1] Berners-Lee, T., Fielding, R., and H. Frystyk,
        "Hypertext Transfer Protocol -- HTTP/1.0",
        RFC 1945, May 1996.

   [2] Berners-Lee, T., Fielding, R., and H. Frystyk,
        "Hypertext Transfer Protocol -- HTTP/1.1"
        RFC 2068, January 1997.

   [3] Rivest, R., "The MD5 Message-Digest Algorithm",
        RFC 1321, April 1992.

6. Authors' Addresses

    John Franks
    Professor of Mathematics
    Department of Mathematics
    Northwestern University
    Evanston, IL 60208-2730, USA

    EMail: john@math.nwu.edu

    Phillip M. Hallam-Baker
    European Union Fellow
    CERN
    Geneva
    Switzerland

    EMail: hallam@w3.org

    Jeffery L. Hostetler
    Senior Software Engineer
    Spyglass, Inc.
    3200 Farber Drive
    Champaign, IL 61821, USA

    EMail: jeff@spyglass.com

    Paul J. Leach
    Microsoft Corporation
    1 Microsoft Way
    Redmond, WA 98052, USA

    EMail: paulle@microsoft.com

    Ari Luotonen
    Member of Technical Staff
    Netscape Communications Corporation
    501 East Middlefield Road
    Mountain View, CA 94043, USA

    EMail: luotonen@netscape.com

    Eric W. Sink
    Senior Software Engineer
    Spyglass, Inc.
    3200 Farber Drive
    Champaign, IL 61821, USA

    EMail: eric@spyglass.com

    Lawrence C. Stewart
    Open Market, Inc.
    215 First Street
    Cambridge, MA 02142, USA

    EMail: stewart@OpenMarket.com