            HTTP Authentication Extensions for Interactive Clients

Abstract

   This document specifies extensions for the HTTP authentication
   framework for interactive clients. Currently, fundamental features
   of HTTP-level authentication are insufficient for complex
   requirements of various Web-based applications. This forces these
   applications to implement their own authentication frameworks by
   means such as HTML forms, which becomes one of the hurdles against
   introducing secure authentication mechanisms handled jointly by
   servers and user agents. The extended framework fills gaps between
   Web application requirements and HTTP authentication provisions to
   solve the above problems, while maintaining compatibility with
   existing Web and non-Web uses of HTTP authentication.

Status of This Memo

Table of Contents

1. Introduction

   This document defines several extensions to the current HTTP
   authentication framework, to provide functionality comparable with
   current, widely used, form-based Web authentication. A majority of
   the recent websites on the Internet use custom application-layer
   authentication implementations using Web forms. The reasons for
   these may vary, but many people believe that the current HTTP Basic
   and Digest authentication methods do not have enough functionality
   (including good user interfaces) to support most realistic Web-based
   applications. However, such use of form-based Web authentication has
   several weaknesses against attacks like phishing, because all
   behavior of the authentication is controlled from the server-side
   application. This makes it really hard to implement any
   cryptographically strong authentication mechanisms into Web systems.
   To overcome this problem, we need to "modernize" the HTTP
   authentication framework so that better client-controlled secure
   methods can be used with Web applications. The extensions proposed
   in this document include:

   o optional authentication on HTTP (Section 3),

   o log out from both the server and client side (Section 4), and

   o finer control for redirection depending on the authentication
      status (Section 4)

1.1. Terminology

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
   "OPTIONAL" in this document are to be interpreted as described in
   [RFC2119].

   This document distinguishes the terms "client" and "user" in the
   following way: a "client" is an entity understanding and talking HTTP
   and the specified authentication protocol, usually computer software;
   a "user" is a (usually natural) person who wants to access data
   resources using "a client".

2. Definitions

2.1. Terms for Describing Authentication Protocol Flow

   HTTP Authentication defined in [RFC7235] can involve several pairs of
   HTTP requests/responses. Throughout this document, the following
   terms are used to categorize those messages.

   For requests:

   1) A non-authenticating request is a request not attempting any
      authentication: a request without any Authorization header field.

   2) An authenticating request is the opposite: a request with an
      Authorization header field.

   For responses:

   1) A non-authenticated response is a response that does not involve
      any HTTP authentication. It does not contain any WWW-Authenticate
      ([RFC7235]) or Authentication-Info header field ([RFC7615]).

      Servers send this response when the requested resource is not
      protected by an HTTP authentication mechanism. In the context of
      this specification, non-authentication-related negative responses
      (e.g., 403 and 404) are also considered non-authenticated
      responses.

      (See the note on successfully authenticated responses below for
      some ambiguous cases.)

   2) An authentication-initializing response is a response that
      requires or allows clients to start authentication attempts.
      Servers send this response when the requested resource is
      protected by an HTTP authentication mechanism, and the request
      meets one of the following cases:

      * The request is a non-authenticating request, or

      * The request contained an authentication trial directed to a
         protection space (realm) other than the one that the server
         expected.

      The server will specify the protection space for authentication in
      this response.

Upon receiving this response, the client's behavior is further
divided to two possible cases:

* If the client has no prior knowledge on authentication
  credentials (e.g., a username and a password) related to the
  requested protection space, the protocol flow terminates and
  the client will ask the user to provide authentication
  credentials.

* On the other hand, if the client already has enough
  authentication credentials to the requested protection space,
  the client will automatically send an authenticating request.
  Such cases often occur when the client does not know beforehand
  that the current request-URL requires authentication.

3) A successfully authenticated response is a response for an
   authenticating request meaning that the authentication attempt was
   granted. (Note: if the authentication scheme used does not use an
   Authentication-Info header field, it can't be distinguished from a
   non-authenticated response.)

4) An intermediate authenticating response is a response for an
   authenticating request that requires more reaction by the client
   software without involving users. Such a response is required
   when an authentication scheme requires two or more round-trip
   messages to perform authentication, or when an authentication
   scheme uses some speculative short-cut method (such as uses of
   cached shared secrets) and it fails.

5) A negatively authenticated response is a response for an
   authenticating request, which means that the authentication
   attempt was declined and cannot continue without a different set
   of authentication credentials. Clients typically erase the memory
   of the active credentials and ask the user for other ones.

   Usually the format of these responses is the same as the one for
   authentication-initializing responses. Clients can distinguish
   negatively authenticated responses from authentication-
   initializing responses by comparing the protection spaces
   contained in the request and in the response.

Figure 1 shows a state diagram of generic HTTP authentication with
the above message categorization. Note that many authentication
schemes use only a subset of the transitions described in the
diagram. Labels in the figure show the abbreviated names of response
types.

```
       ==========                       ----------------
       NEW REQUEST                      ( UNAUTHENTICATED )
       ==========                       ----------------
           |                                ^ non-auth.
           v                                | response
  +--------------------+ NO                  +------------+
  | The requested URI  |-------------------------->| send normal |
  | known to be auth'ed? |     --------------->|   request   |
  +--------------------+        /              +------------+
        YES |                  /         initializing|
            v                 /                      |
   +----------------+ NO     /                       |
   | Can auth-req.(*1)|---------                      |
   | be constructed?  |                               |
   +----------------+        initializing            |
        YES |        --------------------------------------.  |
            |       /                                    v v
            |      |      ---------------  NO   +----------+
            |      |      ( AUTH-REQUESTED )<------| passwords |
            |      |      ---------------         |etc. known?|
            v      |                              +----------+
     +----------+ negative  ------------  negative  |YES
     |  send    |--------->( AUTH-FAILED )<--------,  |
    /| auth-req |          ------------         |  |
   / +----------+\                              |  v
   |     \  \  intermediate                  +----------+
   |      \  --------------------------------->|   send   |
   |       \                                 | auth-req |
   | non-auth.  \successful         successful +----------+
   | response (*2)  \                        /   |  ^
   v                 \                      /    |  |
  ----------------    \   ------------    /   `----'
  ( UNAUTHENTICATED ) ----->( AUTH-SUCCEED )<----  intermediate
  ----------------        ------------
```
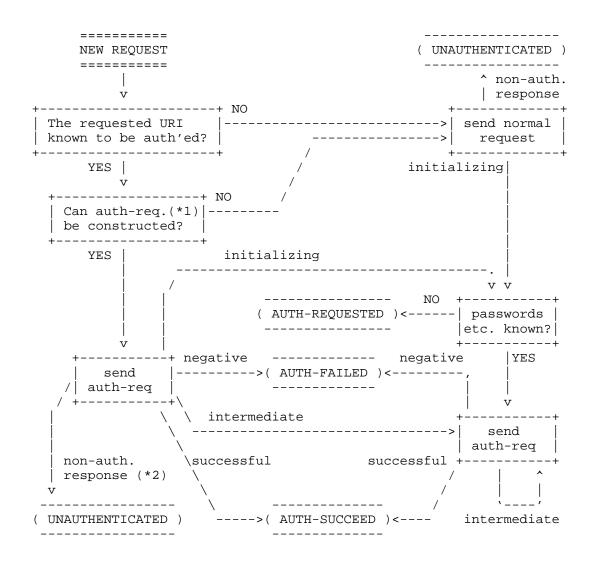
          Figure 1: Generic State Diagram for HTTP Authentication

   Notes:
   (*1) For example, the "Digest" scheme requires a server-provided
   nonce to construct client-side challenges.
   (*2) In "Basic" and some others, this cannot be distinguished from a
   successfully authenticated response.

2.2. Syntax Notation

   This specification uses an extended ABNF syntax defined in [RFC7230]
   and [RFC5234]. The following syntax definitions are quoted from
   [RFC7230] and [RFC7235]: auth-scheme, quoted-string, auth-param, SP,
   BWS, header-field, and challenge. It also uses the convention of
   using header field names for specifying the syntax of values for the
   header field.

   Additionally, this specification uses the following syntax
   definitions as a refinement for token and the right-hand-side of
   auth-param in [RFC7235].

```
 bare-token           = bare-token-lead-char *bare-token-char
 bare-token-lead-char = %x30-39 / %x41-5A / %x61-7A
 bare-token-char      = %x30-39 / %x41-5A / %x61-7A / "-" / "_"
 extension-token      = "-" bare-token 1*("." bare-token)
 extensive-token      = bare-token / extension-token
 integer              = "0" / (%x31-39 *%x30-39)  ; no leading zeros
```

                  Figure 2: The BNF Syntax for Common Notations

   Extensive-tokens are used in this protocol where the set of
   acceptable tokens includes private extensions. Any extensions of
   this protocol MAY use either bare-tokens allocated by IANA (under the
   procedure described in Section 7), or extension-tokens with the
   format "-<token>.<domain-name>", where <domain-name> is a valid
   (sub-)domain name on the Internet owned by the party who defines the
   extension.

3. Optional Authentication

   The Optional-WWW-Authenticate header enables a non-mandatory
   authentication, which is not possible under the current HTTP
   authentication mechanism.

   In several Web applications, users can access the same contents as
   both a guest user and an authenticated user. In most Web
   applications, this functionality is implemented using HTTP cookies
   [RFC6265] and custom form-based authentication. The new
   authentication method using this message will provide a replacement
   for these authentication systems.

Servers MAY send HTTP non-interim responses containing the
Optional-WWW-Authenticate header as a replacement for a 401 response
when it is authentication-initializing. The
Optional-WWW-Authenticate header MUST NOT be sent on 401 responses
(i.e., a usual WWW-Authenticate header MUST be used on 401
responses).

```
 Optional-WWW-Authenticate = 1#challenge
```

        Figure 3: BNF Syntax for Optional-WWW-Authenticate Header

```
    Example:
    HTTP/1.1 200 OK
    Optional-WWW-Authenticate: Basic realm="xxxx"
```

The challenges contained in the Optional-WWW-Authenticate header are
the same as those for a 401 response corresponding to the same
request. For authentication-related matters, an optional
authentication request will have the same meaning as a 401 message
with a corresponding WWW-Authenticate header (as an authentication-
initializing response). (The behavior for other matters MAY be
different between the optional authentication and 401 messages. For
example, clients MAY choose to cache the 200 messages with the
Optional-WWW-Authenticate header field but not the 401 messages by
default.)

A response with an Optional-WWW-Authenticate header SHOULD be
returned from the server only when the request is either non-
authenticated or authenticating to a wrong (not the server's
expected) protection space. If a response is either an intermediate
or a negative response to a client's authentication attempt, the
server MUST respond with a 401 status response with a
WWW-Authenticate header instead. Failure to comply with this rule
will render clients unable to distinguish between authentication
successes and failures.

The server is NOT RECOMMENDED to include an Optional-WWW-Authenticate
header in a positive response when a client's authentication attempt
succeeds.

Whenever an authentication scheme supports servers sending some
parameter that gives a hint about the URL space for the corresponding
protection space for the same realm (e.g., "path" or "domain"),
servers requesting non-mandatory authentication SHOULD send such a
parameter with the response. Clients supporting non-mandatory
authentication MUST recognize the parameter and MUST send a request
with an appropriate authentication credential in an Authorization
header for any URI inside the specified paths.

Implementations are not required to support this header for all of
their supported authentication schemes (i.e., they may choose to
implement it only for a subset of their supported schemes). New
authentication schemes can require support of the optional
authentication as a prerequisite, though.

3.1. Note on Optional-WWW-Authenticate and Use of WWW-Authenticate
      Header with Non-401 Status

In the current specification of HTTP/1.1, it is clarified that the
WWW-Authenticate header can be used with messages with status codes
other than 401 (Authentication Required). In particular, the use of
the WWW-Authenticate header with the 200 status messages implies a
very similar meaning to the above-defined Optional-WWW-Authenticate
header.

The design of Optional-WWW-Authenticate header expects that the use
of a new header guarantees that clients that are unaware of this
extension will ignore the header, and that Web developers can rely on
that behavior to implement a secondary fallback method of
authentication. Several behavioral requirements written in the above
section also assume this property and define a necessary
functionality to implement an optional authentication reliably and
consistently.

On the other hand, some experiments and discussions on the IETF
mailing list revealed that most of (but not necessarily all of) the
existing HTTP clients, at the time of writing, just ignore the WWW-
Authenticate headers in non-401 messages, giving similar behavior
with the Optional-WWW-Authenticate. However, every corner case of
behavior was not fully tested or well-defined in the existing
specifications.

Considering these situations, the authors of this document chose to
use a new header for a new feature "experiment". This is to avoid
defining every corner-case behavior for the existing standard WWW-
Authentication header in this experimental document, which could be
considered by some implementers as an incompatible changes to
existing specification.

Experimentally, the authors propose that implementers of the standard
HTTP/1.1 specification (especially implementers of this extension)
implement undefined (implementation-dependent) detailed handling of
the WWW-Authenticate header with non-401 status messages similar as
those defined above for the Optional-WWW-Authenticate header. For
example, we propose that servers return the 401 status for failed
authentication attempts, even when the unauthenticated request to the
same resource will result in the 200 status. This can determine how

(whether) non-mandatory authentication using the standard header
fields and status codes can be implemented. If this experiment is
successful, a future revision of this experimental document may
"bless" and recommend the use of a standard WWW-Authenticate header,
with some stricter requirements on some corner-case behavior.

4. Authentication-Control Header

```
Authentication-Control = 1#auth-control-entry
auth-control-entry     = auth-scheme 1*SP 1#auth-control-param
auth-control-param     = extensive-token BWS "=" BWS token
                         / extensive-token "*" BWS "=" BWS ext-value
ext-value              = <see RFC 5987, Section 3.2>
```

     Figure 4: The BNF Syntax for the Authentication-Control Header

The Authentication-Control header provides more precise control of
the client behavior for Web applications using an HTTP authentication
protocol. This header is supposed to be generated in the application
layer, as opposed to the WWW-Authenticate headers, which will usually
be generated by the Web servers.

Clients MAY freely choose any subset of these parameters to be
supported. Also, these may choose to support any of the parameters
for only a subset of their supported authentication schemes.
However, authentication schemes can require/recommend support for
some of these parameters as a prerequisite.

The Authentication-Control header contains one or more
"authentication control entries", each of which corresponds to a
single realm for a specific authentication scheme. If the
auth-scheme specified for an entry supports the HTTP "realm" feature,
that entry MUST contain the "realm" parameter. If not, the entry
MUST NOT contain the "realm" parameter.

Among the multiple entries in the header, the relevant entries in the
header are those corresponding to an auth-scheme and a realm (if any)
for which "the authentication process is being performed or going to
be performed". In more detail:

(1) If the response is either an authentication-initializing
     response or a negatively authenticated response, there can be
     multiple challenges in the WWW-Authenticate header (or the
     Optional-WWW-Authenticate header defined in this extension),
     each of which corresponds to a different scheme and realm. In
     this case, the client has a choice about the scheme and realm
     they will use to authenticate. Only the entry in the

         Authentication-Control header corresponding to that scheme and
         realm are relevant.

   (2)   If the response is either an intermediate authenticating
         response or a successfully authenticated response, the scheme
         and realm given in the Authorization header of the HTTP request
         will determine the currently ongoing authentication process.
         Only the entry corresponding to that scheme and realm are
         relevant.

   The server MAY send an Authentication-Control header containing non-
   relevant entries. The client MUST ignore all non-relevant entries it
   received.

   Every entry contains one or more parameters, each of which is a name-
   value pair. The name of each parameter MUST be an extensive-token.
   Clients MUST ignore any unknown parameters contained in this header.
   The entries for the same auth-scheme and the realm MUST NOT contain
   duplicated parameters for the same name. Clients MAY either take any
   one of those duplicated entries or ignore all of them.

   The type of parameter value depends on the parameter name as defined
   in the following subsections. Regardless of the type, however, the
   recipients MUST accept both quoted and unquoted representations of
   values as defined in HTTP. If the parameter is defined to have a
   string value, implementations MUST send any value outside of the
   "token" ABNF syntax in either a quoted form or an ext-value form (see
   Section 4.1). If the parameter is defined as a token (or similar) or
   an integer, the value SHOULD follow the corresponding ABNF syntax
   after possible unquoting of the quoted-string value (as defined in
   HTTP) and MUST be sent in a plain (not an ext-value) form. (Note:
   the rest of this document will show all string-value parameters in
   quoted forms, and it will show others in unquoted forms.)

   Any parameters contained in this header MAY be ignored by clients.
   Also, even when a client accepts this header, users are able to
   circumvent the semantics of this header. Therefore, if this header
   is used for security purposes, its use MUST be limited to providing
   some non-fundamental additional security measures valuable for end-
   users (such as client-side logout for protection against console
   takeover). Server-side applications MUST NOT rely on the use of this
   header for protecting server-side resources.

   Note: The header syntax allows servers to specify Authentication-
   Control for multiple authentication schemes, either as multiple
   occurrences of this header or as a combined single header (see
   Section 3.2.2 of [RFC7230] for rationale). The same care as for
   parsing multiple authentication challenges needs to be taken.

4.1. Non-ASCII Extended Header Parameters

   Parameters contained in the Authentication-Control header MAY be
   extended to non-ASCII values using the framework described in
   [RFC5987]. All servers and clients MUST be capable of receiving and
   sending values encoded in [RFC5987] syntax.

   If a value to be sent contains only ASCII characters, the field MUST
   be sent using plain RFC 7235 syntax. The syntax as extended by
   ext-value MUST NOT be used in this case.

   If a value (except the "realm" header) contains one or more non-ASCII
   characters, the parameter SHOULD be sent using the ext-value syntax
   defined in Section 3.2 of [RFC5987]. Such a parameter MUST have a
   charset value of "UTF-8", and the language value MUST always be
   omitted (have an empty value). The same parameter MUST NOT be sent
   more than once, regardless of the syntax used.

   For example, a parameter "username" with the value "Renee of France"
   SHOULD be sent as < username="Renee of France" >. If the value is
   "Ren<e acute>e of France", it SHOULD be sent as
   < username*=UTF-8''Ren%C3%89e%20of%20France > instead.

   Interoperability note: [RFC7235], Section 2.2, defines the "realm"
   authentication parameter that cannot be replaced by the "realm*"
   extend parameter. This means that the use of non-ASCII values for an
   authentication realm is not the defined behavior in HTTP.
   Unfortunately, some people currently use a non-ASCII realm parameter
   in reality, but even its encoding scheme is not well defined.
   Given this background, this document does not specify how to handle a
   non-ASCII "realm" parameter in the extended header fields. If
   needed, the authors propose using a non-extended "realm" parameter
   form, with a wish for maximum interoperability.

4.2. Auth-Style Parameter

   Example:
   Authentication-Control: Digest realm="protected space",
   auth-style=modal

   The parameter "auth-style" specifies the server's preference for user
   interface behavior for user authentication. This parameter can be
   included in any kind of response; however, it is only meaningful for
   either authentication-initializing or negatively authenticated
   responses. The value of this parameter MUST be one of the bare-
   tokens, "modal" or "non-modal". When the Optional-WWW-Authenticate
   header is used, the value of this parameter MUST be disregarded and
   the value "non-modal" is implied.

The value "modal" means that the server thinks the content of the
response (body and other content-related headers) is valuable only
for users refusing the authentication request. The clients are
expected to ask the user for a password before processing the
content. This behavior is common for most of the current
implementations of Basic and Digest authentication schemes.

The value "non-modal" means that the server thinks that the content
of the response (body and other content-related headers) is valuable
for users before processing an authentication request. The clients
are expected to first process the content and then provide users with
the opportunity to perform authentication.

The default behavior for clients is implementation dependent, and it
may also depend on authentication schemes. The proposed default
behavior is "modal" for all authentication schemes unless otherwise
specified.

The above two different methods of authentication possibly introduce
an observable difference of semantics when the response contains
state-changing side effects; for example, it can affect how Cookie
headers [RFC6265] in 401 responses are processed. However, the
server applications SHOULD NOT depend on the existence of such side
effects.

4.3. Location-When-Unauthenticated Parameter

Example:
Authentication-Control: Mutual realm="auth-space-1",
location-when-unauthenticated="http://www.example.com/login.html"

The parameter "location-when-unauthenticated" specifies a location to
which any unauthenticated clients should be redirected. This header
can be used, for example, when there is a central login page for the
entire Web application. The value of this parameter is a string that
contains a URL location. If a received URL is not absolute, the
clients SHOULD consider it a relative URL from the current location.

This parameter MAY be used with a 401 response for an authentication-
initializing response. It can also be contained, although this is
NOT RECOMMENDED, in a positive response with an
Optional-WWW-Authenticate header. The clients MUST ignore this
parameter when a response is either successfully authenticated or
intermediately authenticated.

When a client receives an authentication-initiating response with
this parameter, and if the client has to ask users for authentication
credentials, the client will treat the entire response as if it were
a 303 "See Other" response with a Location header that contains the
value of this parameter (i.e., the client will be redirected to the
specified location with a GET request). Unlike a normal 303
response, if the client can process authentication without the user's
interaction, this parameter MUST be ignored.

4.4. No-Auth Parameter

Example:
Authentication-Control: Basic realm="entrance", no-auth=true

The parameter "no-auth" is a variant of the
location-when-unauthenticated parameter; it specifies that new
authentication attempts are not to be performed on this location in
order to improve the user experience, without specifying the
redirection on the HTTP level. This header can be used, for example,
when there is a central login page for the entire Web application and
when an explicit user interaction with the Web content is desired
before authentication. The value of this parameter MUST be a token
"true". If the value is incorrect, the client MAY ignore this
parameter.

This parameter MAY be used with authentication-initiating responses.
It can also be contained, although this is NOT RECOMMENDED, in a
positive response with an Optional-WWW-Authenticate header. The
clients MUST ignore this parameter when a response is either
successfully authenticated or intermediately authenticated.

When a client receives an authentication-initiating response with
this parameter, if the client has to ask users for authentication
credentials, the client will ignore the WWW-Authenticate header
contained in the response and treat the whole response as a normal
negative 4xx-class response instead of giving the user an opportunity
to start authentication. If the client can process authentication
without the user's interaction, this parameter MUST be ignored.

Using this parameter along with the location-when-unauthenticated
parameter is meaningless. If both were supplied, clients SHOULD
ignore the location-when-unauthenticated parameter.

This parameter SHOULD NOT be used as a security measure to prevent
authentication attempts, as it is easily circumvented by users. This
parameter SHOULD be used solely for improving the user experience of
Web applications.

4.5. Location-When-Logout Parameter

     Example:
     Authentication-Control: Digest realm="protected space",
     location-when-logout="http://www.example.com/byebye.html"

     The parameter "location-when-logout" specifies a location where the
     client is to be redirected when the user explicitly requests a
     logout. The value of this parameter MUST be a string that contains a
     URL location. If a given URL is not absolute, the clients MUST
     consider it a relative URL from the current location.

     This parameter MAY be used with successfully authenticated responses.
     If this parameter is contained in other kinds of responses, the
     clients MUST ignore this parameter.

     When the user tells the client to terminate the current
     authentication period, if the client currently displays a page
     supplied by a response with this parameter, the client will
     automatically change the current location to the location specified
     in this parameter using a new GET request, as if it has received a
     303 response. Any operations related to logout (e.g., erasing
     memories of username, authentication credential, and all related one-
     time credentials such as nonce or keys) SHOULD occur before
     processing a page transition.

     When the user requests the client for the termination of an
     authentication period, if the client supports this parameter but the
     server response does not contain this parameter, the client's
     RECOMMENDED behavior is as follows: if the request corresponding to
     the current content was the GET method, reload the page without the
     authentication credential. Otherwise, keep the current content as-is
     and simply forget the authentication status. The client SHOULD NOT
     replay a non-idempotent request without the user's explicit approval.

     Web applications are encouraged to send this parameter with an
     appropriate value for any responses (except those with redirection
     (3XX) statuses) for non-GET requests.

     See Section 5 for some examples for possible deployment of this
     parameter.

4.6. Logout-Timeout Parameter

     Example:
     Authentication-Control: Basic realm="entrance", logout-timeout=300

     The parameter "logout-timeout", when contained in a successfully
     authenticated response, means that any authentication credentials and
     state related to the current protection space are to be discarded if
     the time specified in this header (in seconds) has passed since the
     time this header was received. The value MUST be an integer. As a
     special case, the value 0 means that the server is logging the client
     out immediately from the current authentication space and that the
     client is now returned to the unauthenticated state. This does not,
     however, mean that the long-term memories for the passwords and
     passwords-related details (such as password reminders and auto fill-
     ins) should be removed. If a new timeout value is received for the
     same authentication space, it cancels the previous timeout and sets a
     new timeout.

4.7. Username Parameter

     Example:
     Authentication-Control: Basic realm="configuration", username="admin"

     The parameter "username" tells us that the only "username" to be
     accepted by the server is the value given in this parameter.

     This parameter is particularly useful, for example, for routers and
     other network appliances with a Web configuration interface. Many of
     those use an HTTP Basic authentication with one predefined username,
     with many varieties such as "admin", "root", "user", etc. In the
     current situation, users have almost no hint about the valid username
     upon the authentication request. Some show the valid value in a
     "realm" string, some in the 401-status response page, shown _after_
     the user gave up the authentication and canceled the authentication
     dialog. If this parameter is given, the client Web browser can auto-
     fill the username field in the authentication dialog before the users
     attempt to authenticate themselves.

     This parameter MAY be used with authentication-initiating responses
     or negatively authenticated responses requiring another attempt at
     authentication. The clients MUST ignore this parameter when a
     response is either successfully authenticated or intermediately
     authenticated.

   If the authentication scheme to be used has a syntax limitation on
   the allowed usernames (e.g., Basic and Digest do not allow colons in
   usernames); the specified value MUST follow that limitation. Clients
   SHOULD ignore any values that do not conform to such limitations.

   Also, if the used authentication scheme requires a specific style of
   text preparation for the username (e.g., PRECIS [RFC7564] string
   preparation or Unicode normalization), the server SHOULD send the
   values satisfying such requirements (so that clients can use the
   given username as is).

   Clients MAY still send any authentication requests with other
   usernames, possibly in vain. Clients are not required (also not
   forbidden) to give users opportunities for supplying a username
   different from the server-specified one. Servers are also not
   strictly required to reject usernames other than specified, but doing
   so will usually result in bad user experiences and may confuse users
   and clients.

   Although this parameter is useful in a specific class of use cases,
   using it in a general use case has many security implications and
   possible pitfalls. Please consult Section 8.1 before deciding to use
   this parameter.

5. Usage Examples

   This section shows some examples for applying this extension to
   typical websites that use forms and cookies for managing
   authentication and authorization. The content of this section is not
   normative and is for illustrative purposes only.

   In these examples, we assume that there are two kinds of clients (Web
   browsers). One kind of these implements all features described in
   the previous sections. We also assume that browsers will have a user
   interface that allows users to deactivate (log out from) current
   authentication sessions. The other kind are the "existing"
   implementations that do not support any of these features.

   When not explicitly specified, all settings described below are to be
   applied with Authentication-Control headers, and these can be sent to
   clients regardless of the authentication status (these will be
   silently ignored whenever not effective).

5.1. Example 1: A Portal Site

   This subsection provides an example application for a site whose
   structure is somewhat similar to conventional portal sites. In
   particular, most Web pages are available for guest (unauthenticated)
   users, and, if authentication is performed, the content of these
   pages is customized for each user. We assume that the site has the
   following kinds of pages currently:

   o Content pages

   o Pages/mechanism for performing authentication:

      * There is one page that asks for a username and a password using
         a HTML POST form.

      * After the authentication attempt, the user will be redirected
         to either the page that was previously displayed before the
         authentication or some specific page.

   o A de-authentication (logout) page.

5.1.1. Case 1: A Simple Application

   When such a site does not require specific actions upon login and
   logout, the following simple settings can be used:

   o Set up an optional authentication to all pages available to
      guests. Set up an Authentication-Control header with the "auth-
      style=non-modal" setting.

   o If there are pages only available to authenticated users, set up a
      mandatory authentication with the "auth-style=non-modal" setting.

   o No specific pages for authentication are needed. It will be
      performed automatically, directed by the above setting.

   o A de-authentication page is also not needed. If the site has one,
      put "logout-timeout=0" there.

   o For all pages for POST requests, it is advisable to have a
      "location-when-logout=<some page>".

5.1.2. Case 2: Specific Action Required on Logout

   If the site requires specific actions upon logout, the following
   settings can be used:

   o All settings in Case 1 are applied.

   o For all pages, set up the Authentication-Control header "location-
      when-logout=<de-authentication page>".

   o In the de-authentication page, no specific setup is needed. If
      there are any direct links to the de-authentication page, put
      "logout-timeout=0".

5.1.3. Case 3: Specific Page Displayed before Login

   If the site needs to display a specific page before login actions
   (some announcements, user notices, or even advertisements), the
   following settings can be applied:

   o Set up an optional authentication to all pages available to
      guests. Set up an Authentication-Control header with
      "no-auth=true". Put a link to a specific login page in contents.

   o If there are pages only available to authenticated users, set up a
      mandatory authentication with the
      "location-when-unauthenticated=<the login page>".

   o For the specific login page, set up a mandatory authentication.

   o For all pages for POST requests, it is advisable to have
      "location-when-logout=<some page>", too.

   o De-authentication pages are not needed. If the site has one, put
      "logout-timeout=0".

5.2. Example 2: Authenticated User-Only Sites

   If almost all pages in the target site require authentication (e.g.,
   an Internet banking site), or if there is no need to support both
   unauthenticated and authenticated users on the same resource, the
   settings will become simpler. The following are examples for such a
   site:

   o Set up a mandatory authentication to all pages available to
      authenticated users. Set up an Authentication-Control header with
      the "auth-style=non-modal" setting.

   o Set up a handler for the 401-status that requests users to
      authenticate.

   o For all pages for POST requests, it is advisable to have a
      "location-when-logout=<some page>", too.

   o De-authentication pages are not needed. If the site will have
      one, put "logout-timeout=0" there.

5.3. When to Use Cookies

   In current websites using form-based authentication, Cookies
   [RFC6265] are used for managing both authorization and application
   sessions. Using the extensions in this document, the former features
   will be provided by using (extended) HTTP authentication/
   authorization mechanisms. In some cases, there will be ambiguity on
   whether some functions are for authorization management or for
   session management. The following hints will be helpful for deciding
   which features to use.

   o If there is a need to serve multiple sessions for a single user
      using multiple browsers concurrently, use a Cookie for
      distinguishing between sessions for the same user. (C.f. if there
      is a need to distinguish between sessions in the same browser,
      HTML5 Web Storage [W3C.REC-webstorage-20130730] features can be
      used instead of Cookies.)

   o If a website is currently deploying a session time-out feature,
      consider who benefits from the feature. In most cases, the main
      requirement for such a feature is to protect users from having
      their consoles and browsers hijacked (i.e., benefits are on the
      users' side). In such cases, the time-out features provided in
      this extension can be used. On the other hand, the requirement is
      to protect the server's privilege (e.g., when some regulations
      require limiting the time difference between a user's two-factor
      authentication and financial transaction commitment; the
      requirement is strictly on the servers' side), that should be
      managed on the server side using Cookies or other session-
      management mechanisms.

5.4. Parallel Deployment with Form/Cookie Authentication

   In some transition periods, sites may need to support both HTTP-layer
   and form-based authentication. The following example shows one way
   to achieve that.

   o If Cookies are used even for HTTP-authenticated users, each
      session determined by Cookies SHOULD identify which authentication
      has been used for the session.

   o First, set up any of the above settings for enabling HTTP-layer
      authentication.

   o For unauthenticated users, add the following things to the Web
      pages, unless the client supports this extension and HTTP-level
      authentication:

      * For non-mandatory authenticated pages, add a link to the form-
         based authenticated pages.

      * For mandatory authenticated pages, either put a link to form-
         based authenticated pages or put an HTML-level redirection
         (using <META http-equiv="refresh" ...> element) to such pages.

   o In the form-based authenticated pages, if users are not
      authenticated, the page can provide a redirection for HTTP-level
      authentication by the "location-when-unauthenticated" setting.

   o Users are identified for authorization and content customization
      by the following logic:

      * First, check the result of the HTTP-level authentication. If
         there is a Cookie session tied to a specific user, both should
         match.

      * If the user is not authenticated on the HTTP-level, use the
         conventional form-based method to determine the user.

      * If there is a Cookie tied to HTTP authentication but there is
         no corresponding HTTP authentication result, that session will
         be discarded (because it means that authentication is
         deactivated by the corresponding user).

6. Methods to Extend This Protocol

   If a private extension to this protocol is implemented, it MUST use
   the extension-param to avoid conflicts with this protocol and any
   other extensions. (Standardized extensions or extensions that are
   being standardized MAY use either bare-tokens or extension-tokens.)

   When bare-tokens are used in this protocol, these MUST be allocated
   by IANA. Any tokens used for non-private, non-experimental
   parameters are RECOMMENDED to be registered with IANA, regardless of
   the kind of tokens used.

   Extension-tokens MAY be freely used for any non-standard, private,
   and/or experimental uses. An extension-token MUST use the format
   "-<bare-token>.<domain-name>", where <domain-name> is a validly
   registered (sub-)domain name on the Internet owned by the party that
   defines the extensions. Any unknown parameter name is to be ignored
   regardless of whether it is an extension-token or a bare-token.

7. IANA Considerations

   This document defines two new entries for the "Permanent Message
   Header Field Names" registry.

   +-------------+-------------------------+-------------------------+
   | | Entry 1: | Entry 2: |
   +-------------+-------------------------+-------------------------+
Header	Optional-WWW-Authenticate	Authentication-Control
Field Name		
Protocol	http	http
Status	experimental	experimental
Change	IETF	IETF
Control		
Spec.	Section 3 of this	Section 4 of this
Document	document	document
   +-------------+-------------------------+-------------------------+

   This document also establishes the "HTTP Authentication Control
   Parameters" registry. The registry manages case-insensitive ASCII
   strings. The string MUST follow the extensive-token syntax defined
   in Section 2.2.

   To acquire registered tokens, a specification for the use of such
   tokens MUST be available as a publicly accessible document (see
   "Specification Required" in [RFC5226]).

Registrations for authentication control parameters are required to
include a description of the control extension. New registrations
are advised to provide the following information:

o Token: A token used in HTTP headers for identifying the algorithm.

o Specification: A reference for the specification defining the
   algorithm.

The initial content of this registry is as follows:

```
+-----------------------------+-----------------------------+
| Token                       | Specification               |
+-----------------------------+-----------------------------+
| auth-style                  | Section 4.2 of this document |
| location-when-unauthenticated | Section 4.3 of this document |
| no-auth                     | Section 4.4 of this document |
| location-when-logout        | Section 4.5 of this document |
| logout-timeout              | Section 4.6 of this document |
| username                    | Section 4.7 of this document |
+-----------------------------+-----------------------------+
```

8. Security Considerations

The purpose of the logout timeout feature in the Authentication-
control header is to protect users of clients from impersonation
caused by an attacker having access to the same console. The server
application implementers SHOULD be aware that the directive may
always be ignored by either malicious clients or clients not
supporting this extension. If the purpose of introducing a timeout
for an authentication period is to protect server-side resources,
this protection MUST be implemented by other means such as HTTP
Cookies [RFC6265].

All parameters in the Authentication-Control header SHOULD NOT be
used for any security-enforcement purposes. Server-side applications
MUST NOT assume that the header will be honored by clients and users.

8.1. Security Implication of the Username Parameter

The "username" parameter sometimes reveals sensitive information
about the HTTP server and its configurations that are useful for
security attacks. In general, common security practice suggests that
any kind of information on the existence/non-existence of a specific
username shall not be disclosed before successful authentication.
Obviously, the "username" parameter contradicts this practice.

Given this background, the use of the "username" parameter SHOULD be strictly limited to cases where all of the following conditions are met:

(1) the valid username is pre-configured and not modifiable (such as root, admin, or similar ones);

(2) the valid username for such an appliance is publicly known (for example, written in a manual document); and

(3) either the valid username for the server is easily guessable by other means (for example, from the model number shown in an unauthenticated page), or the server is accessible only from limited networks.

Most importantly, the "username" parameter SHOULD NOT be used in any case when the valid usernames can be changed/configured by users or administrators.

9. References

9.1. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

   [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

   [RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, DOI 10.17487/RFC5987, August 2010, <http://www.rfc-editor.org/info/rfc5987>.

   [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

   [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Authentication", RFC 7235,
              DOI 10.17487/RFC7235, June 2014,
              <http://www.rfc-editor.org/info/rfc7235>.

9.2. Informative References

   [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265,
              DOI 10.17487/RFC6265, April 2011,
              <http://www.rfc-editor.org/info/rfc6265>.

   [RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework:
              Preparation, Enforcement, and Comparison of
              Internationalized Strings in Application Protocols",
              RFC 7564, DOI 10.17487/RFC7564, May 2015,
              <http://www.rfc-editor.org/info/rfc7564>.

   [RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-
              Authentication-Info Response Header Fields", RFC 7615,
              DOI 10.17487/RFC7615, September 2015,
              <http://www.rfc-editor.org/info/rfc7615>.

   [W3C.REC-webstorage-20130730]
              Hickson, I., "Web Storage", World Wide Web Consortium
              Recommendation REC-webstorage-20130730, July 2013,
              <http://www.w3.org/TR/2013/REC-webstorage-20130730>.

Appendix A. (Informative) Applicability of Features for Each Message

   This section provides a cross-reference table showing the
   applicability of the features provided in this specification to each
   kind of response described in Section 2.1. The table provided in
   this section is for informative purposes only.

	init.	success.	intermed.	neg.
Optional auth.	O	n	N	N
auth-style	O	–	–	O
loc.-when-unauth.	O	I	I	i
no-auth	O	I	I	i
loc.-when-logout	–	O	–	–
logout-timeout	–	O	–	–
username	O	–	–	O

   Legends:
   O = MAY contain; n = SHOULD NOT contain; N = MUST NOT contain
   i = SHOULD be ignored; I = MUST be ignored;
   - = meaningless (to be ignored)

Authors' Addresses

   Yutaka Oiwa
   National Institute of Advanced Industrial Science and Technology
   Information Technology Research Institute
   Tsukuba Central 1
   1-1-1 Umezono
   Tsukuba-shi, Ibaraki
   Japan

   Email: y.oiwa@aist.go.jp

   Hajime Watanabe
   National Institute of Advanced Industrial Science and Technology
   Information Technology Research Institute
   Tsukuba Central 1
   1-1-1 Umezono
   Tsukuba-shi, Ibaraki
   Japan

   Email: h-watanabe@aist.go.jp

   Hiromitsu Takagi
   National Institute of Advanced Industrial Science and Technology
   Information Technology Research Institute
   Tsukuba Central 1
   1-1-1 Umezono
   Tsukuba-shi, Ibaraki
   Japan

   Email: takagi.hiromitsu@aist.go.jp

   Kaoru Maeda
   Lepidum Co. Ltd.
   Village Sasazuka 3, Suite #602
   1-30-3 Sasazuka
   Shibuya-ku, Tokyo
   Japan

   Email: maeda@lepidum.co.jp

   Tatsuya Hayashi
   Lepidum Co. Ltd.
   Village Sasazuka 3, Suite #602
   1-30-3 Sasazuka
   Shibuya-ku, Tokyo
   Japan

   Email: hayashi@lepidum.co.jp

   Yuichi Ioku
   Individual Contributor

   Email: mutual-work@ioku.org