HTTPAUTH Working Group
Internet-Draft
Intended status: Experimental
Expires: January 8, 2017

Y. Oiwa
H. Watanabe
H. Takagi
ITRI, AIST
T. Hayashi
Lepidum
Y. Ioku
Individual
July 7, 2016

HTTP Authentication Extensions for Interactive Clients
draft-ietf-httpauth-extension-07

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 like HTML forms, which becomes one of the hurdles against introducing secure authentication mechanisms handled jointly by servers and user-agent. 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 authentications.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2017.

Copyright Notice

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 weakness 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:

- optional authentication on HTTP (Section 3),
- log out from both server and client side (Section 4), and
- finer control for redirection depending on 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].

The terms "encouraged" and "advised" are used for suggestions that do not constitute "SHOULD"-level requirements. People MAY freely choose not to include the suggested items. However, complying with those suggestions would be a best practice; it will improve the security, interoperability, and/or operational performance.

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 which does not involve any HTTP authentication. It does not contain any WWW-Authenticate or Authentication-Info header field.

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

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

2) An authentication-initializing response is a response which requires or allows clients to start authentication attempts. Servers send this response when the requested resource is protected by 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 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 user-name 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 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 did 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 distinguishable from a non-authenticated response.)

4) An intermediate authenticating response is a response for an authenticating request which 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 failed.

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

   Usually the format of these responses are as 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 on the diagram. Labels in the figure show the abbreviated names of response types.

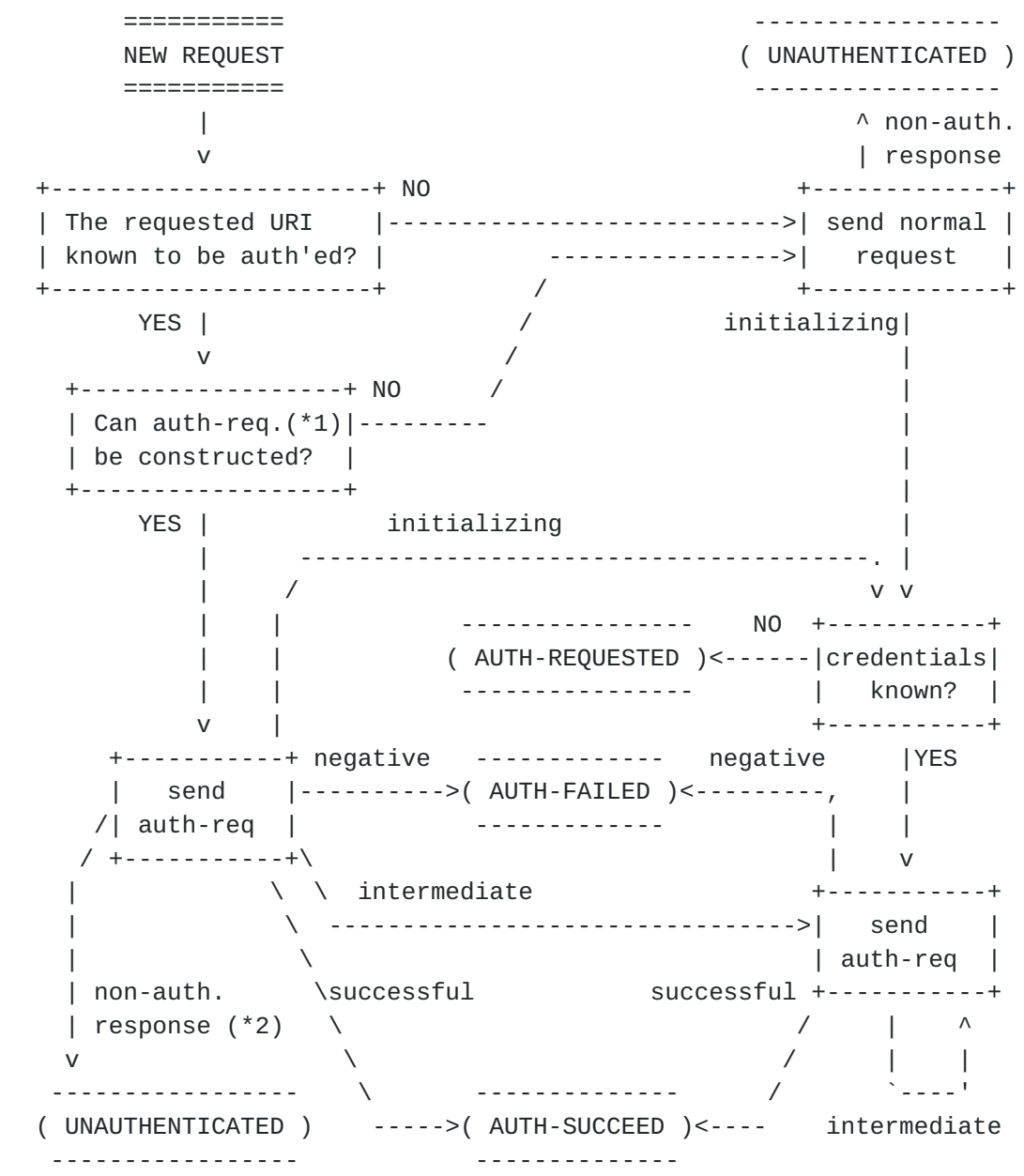

Figure 1: Generic state diagram for HTTP authentication

Note: (*1) For example, "Digest" scheme requires 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]. (Note: these definitions are consistent with those in [I-D.ietf-httpauth-mutual].)

```
bare-token        = 1*(%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 of a 401 response when it the response is authentication-initializing. The Optional-WWW-Authenticate header MUST NOT sent on 401 responses (i.e. a usual WWW-Authenticate header MUST be used on 401 responses.)

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

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

Figure 3: BNF syntax for Optional-WWW-Authenticate header

The challenges contained in the Optional-WWW-Authenticate header are the same as those for a 401 responses 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 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 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 which gives a hint of 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 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.

Support of this header is OPTIONAL; clients MAY also implement this extension only for some selected authentication schemes. New authentication schemes can make support of the optional authentication mandatory by its specification, 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). Especially, the use of 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 which is unaware of this extension will ignore the header, and that Web developers can rely on that behavior to implement a secondary fallback method of authentications. Several behavioral requirements written in the above section also assumes this property, and defines a necessary functionality to implement an HTTP 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 ignores the WWW-Authenticate headers in non-401 messages, giving the similar behavior with the Optional-WWW-Authenticate. However, every corner case of behavior was not fully tested, nor well-defined in the existing specifications.

Considering these situations, the author 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 implementer as an "incompatible changes to existing specification".

Experimentally, the authors propose implementer of the standard HTTP/1.1 specification (especially implementer of this extension) to implement undefined (implementation-dependant) detailed handling of WWW-Authenticate header with non-401 status messages as similar as those defined above for the Optional-WWW-Authenticate header. For example, we propose for servers to return 401 status for failed authentication attempts, even when the unauthenticated request to the same resource will result in the 200 status. This can realize how (whether) we can implement non-mandatory authentication using the standard header fields and status codes. If this experiment is successful, the future revision of this experimental document may "bless" and recommend the use of standard WWW-Authenticate header, with some "standard-level" 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 a 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 WWW-Authenticate headers which will usually be generated by the Web servers.

Support of this header is OPTIONAL, and clients MAY choose any subset of these parameters to be supported. The set of supported parameters MAY also be authentication scheme-dependent. However, some authentication schemes can require mandatory/recommended support for some or all of the features provided in this header.

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 on 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.

Each 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 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 an plain (not an ext-value) form. (Note: the rest of this document will show all string-value parameters in quoted forms, and 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 log-out for protecting 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 used syntax.

For example, a parameter "username" with 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 which cannot be replaced by the "realm*" extend parameter. It means that the use of non-ASCII values for an authentication realm is not the defined behavior in the HTTP. Unfortunately, some people currently use 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 non-ASCII "realm" parameter in the extended header fields. If needed, the authors propose to use 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 preferences 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 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 the opportunity to perform authentication.

The default behavior for clients is implementation-dependent, and it may also depending 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 a 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 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 where any unauthenticated clients should be redirected to. 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 an 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., 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 authentications. The value of this parameter MUST be a token "true". If the value is incorrect, 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.

This parameter SHOULD NOT be used along with the location-when-unauthenticated parameter. If both were supplied, clients MAY choose which one is to be honored.

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 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 an 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 requests termination of an authentication period, and if the client currently displays a page supplied by a response with this parameter, the client will be redirected to the specified location by a new GET request (as if it received a 303 response). The log-out operation (e.g. erasing memories of user name, authentication credential and all related one-time credentials such as nonce or keys) SHOULD occur before processing a redirection.

When the user requests 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 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.

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 a time specified in this header (in seconds) has passed since from the time this header was received. The value MUST be an integer. As a special case, the value 0 means that the client is requested to immediately log-out from the current authentication space and revert to an unauthenticated status. This does not, however, mean that the long-term memories for the passwords and passwords-related details (such as the 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 that the only "user name" to be accepted by the server is the value given in this parameter. This parameter is particularly useful, for example, for routers and other appliances with a Web configuration interface.

This parameter MAY be used with authentication-initiating responses or negatively-authenticated responses requiring another attempt of 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 user names (e.g. Basic and Digest do not allow colons in user names), the specified value MUST follow that limitation. Clients SHOULD ignore any values which do not conform to such limitations.

Also, if the used authentication scheme requires a specific style of text preparation for the user name (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 user name as is).

Clients MAY still send any authentication requests with other user names, possibly in vain. Servers are not strictly required to reject user names other than specified, but doing so will give bad user experiences and may confuse users and clients.

5. Usage examples

This section shows some examples for applying this extension to typical websites which are using Forms and cookies for managing authentication and authorization. The content of this section is not normative and 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 which allows users to deactivate (log-out from) current authentication sessions. The other kind are the "existing" implementations which 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 the site has the following kinds of pages currently:

- Content pages.
- Pages/mechanism for performing authentication:
  - There is one page which asks a user name and a password using a HTML POST form.
  - After the authentication attempt, the user will be redirected to either the page which is previously displayed before the authentication, or some specific page.
- A de-authentication (log-out) page.

5.1.1. Case 1: a simple application

When such a site does not require specific actions upon log-in and log-out, the following simple settings can be used.

- Set up an optional authentication to all pages available to guests. Set up an Authentication-Control header with "auth-style=non-modal" setting.
- If there are pages only available to authenticated users, set up a mandatory authentication with "auth-style=non-modal" setting.
- No specific pages for authentication are needed. It will be performed automatically, directed by the above setting.
- A de-authentication page is also not needed. If the site has one, put "logout-timeout=0" there.
- For all pages for POST requests, it is advisable to have "location-when-logout=<some page>".

5.1.2. Case 2: specific action required on log-out

If the site requires specific actions upon log-out, the following settings can be used.

- All settings in the Case 1 are applied.
- For all pages, set up the Authentication-Control header "location-when-logout=<de-authentication page>".
- In the de-authentication page, no specific set-up 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 log-in

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

- 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 log-in page in contents.
- If there are pages only available to authenticated users, set up a mandatory authentication with "location-when-unauthenticated=<the log-in page>".
- For the specific log-in page, set up a mandatory authentication.
- For all pages for POST requests, it is advisable to have "location-when-logout=<some page>", too.
- 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 are no needs to support both unauthenticated and authenticated users on the same resource, the settings will become simpler. The following are an example for such a site:

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

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

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

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

5.3. When to use Cookies

In the current Web sites using form-based authentications, 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.

- 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 sessions in the same browser, HTML5 Web Storage [W3C.REC-webstorage-20130730] features can be used instead of Cookies.)

- If a web site 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 server's privilege (e.g. when some regulations require to limit the time difference between 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 authentications

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

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

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

- 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, put a link to Form-based authenticated pages.

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

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

- Users are identified to 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 or being-standardized extensions 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 to 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-tokens 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 who 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.

Header Field Name	Protocol	Specification
Optional-WWW-Authenticate	http	Section 3 of this document
Authentication-Control	http	Section 4 of this document

This document also establishes a registry for HTTP authentication control parameters. 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 documents, as outlined as "Specification Required" level 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:

- Token: a token used in HTTP headers for identifying the algorithm.

- Specification: A reference for a 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 log-out 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 implementer 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.

The "username" parameter sometimes reveals sensitive information about the HTTP server and its configurations, useful for security attacks. The use of the "username" parameter SHOULD be limited to cases where the all of the following conditions are met:

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

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

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

Most importantly, the "username" parameter SHOULD NOT be used in any case when the valid user names can be changed 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

[I-D.ietf-httpauth-mutual] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP", draft-ietf-httpauth-mutual-08 (work in progress), July 2016.

[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>.

[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 messages

This section provides a cross-reference table showing the applicability of the features provided in this specification to each kind of responses 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)

Appendix B. (Informative) Draft Change Log

[To be removed on final publication]

B.1. Changes in Httpauth WG Revision 07

- WGLC comments are reflected to the text.

B.2. Changes in Httpauth WG Revision 06

- Several comments from reviewers are reflected to the text.

B.3. Changes in Httpauth WG Revision 05

- Authors' addresses updated.

B.4. Changes in Httpauth WG revision 04

- IANA consideration section added.

B.5. Changes in Httpauth WG revision 03

- Adopting RFC 5987 extended syntax for non-ASCII parameter values.

B.6. Changes in Httpauth WG revision 02

- Added realm parameter.

- Added username parameter. We acknowledge Michael Sweet's proposal for including this to the Basic authentication.

B.7. Changes in Httpauth WG revision 01

- Clarification on peers' responsibility about handling of relative URLs.

- Automatic reloading should be allowed only on safe methods, not always on idempotent methods.

B.8. Changes in Httpauth revision 00 and HttpBis revision 00

None.

B.9. Changes in revision 02

- Added usage examples.

B.10. Changes in revision 01

- Syntax notations and parsing semantics changed to match httpbis style.

B.11. Changes in revision 00

- Separated from HTTP Mutual authentication proposal (-09).

- Adopting httpbis works as a referencing point to HTTP.

- Generalized, now applicable for all HTTP authentication schemes.

- Added "no-auth" and "auth-style" parameters.

- Loosened standardization requirements for parameter-name tokens registration.

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
JP

Email: mutual-auth-contact-ml@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
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
JP

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

Yuichi Ioku
Individual