http-state Working Group	A. Barth
Internet-Draft	U.C. Berkeley
Expires: February 16, 2010	August 15, 2009

HTTP State Management Mechanism
draft-abarth-cookie-01

Status of this Memo

Copyright Notice

Abstract

This document defines the HTTP Cookie and Set-Cookie headers.
NOTE:

> This document is currently a "straw-man" cookie proposal. Much of the text herein is completely wrong. If you have suggestions for improving the draft, please send email to http-state@ietf.org. Suggestions with test cases are especially appreciated.

Table of Contents

1. Introduction

TOC

This document defines the HTTP Cookie and Set-Cookie header.

2. Terminology

TOC

The terms user agent, client, server, proxy, and origin server have the same meaning as in the HTTP/1.0 specification.
Fully-qualified host name (FQHN) means either the fully-qualified domain name (FQDN) of a host (i.e., a completely specified domain name ending in a top-level domain such as .com or .uk), or the numeric Internet Protocol (IP) address of a host. The fully qualified domain name is preferred; use of numeric IP addresses is strongly discouraged. [TODO: What does "strongly discouraged" mean?]
The terms request-host and request-URI refer to the values the client would send to the server as, respectively, the host (but not port) and abs_path portions of the absoluteURI (http_URL) of the HTTP request line. Note that request-host must be a FQHN. Hosts names can be specified either as an IP address or a FQHN string. Sometimes we compare one host name with another. Host A's name domain-matches host B's if

- both host names are IP addresses and their host name strings match exactly; or

- both host names are FQDN strings and their host name strings match exactly; or

- A is a FQDN string and has the form NB, where N is a non-empty name string, B has the form .B, and B is a FQDN string. (So, x.y.com domain-matches .y.com but not y.com.)

Note that domain-match is not a commutative operation: a.b.c.com domain-matches .c.com, but not the reverse.
Because it was used in Netscape's original implementation of state management, we will use the term cookie to refer to the state information that passes between an origin server and user agent, and that gets stored by the user agent.

3. State and Sessions

TOC

This document describes a way to create stateful sessions with HTTP requests and responses. HTTP servers respond to each client request without relating that request to previous or subsequent requests; the technique allows clients and servers that wish to exchange state information to place HTTP requests and responses within a larger context, which we term a "session". This context might be used to create, for example, a "shopping cart", in which user selections can be aggregated before purchase, or a magazine browsing system, in which a user's previous reading affects which offerings are presented.
There are, of course, many different potential contexts and thus many different potential types of session. The designers' paradigm for sessions created by the exchange of cookies has these key attributes:

1. Each session has a beginning and an end.
2. Each session is relatively short-lived.
3. Either the user agent or the origin server may terminate a session.
4. The session is implicit in the exchange of state information.

4. Overview

TOC

We outline here a way for an origin server to send state information to the user agent, and for the user agent to return the state information to the origin server.
The two state management headers, Set-Cookie and Cookie, have common syntactic properties involving attribute-value pairs. The following grammar uses the notation, and tokens DIGIT (decimal digits) and token (informally, a sequence of non-special, non-white space characters) from the HTTP/1.1 specification [RFC 2068] to describe their syntax.

4.1. Examples

TOC

TOC

5. Server Conformance

5.1. General

TOC

The origin server initiates a session, if it so desires. (Note that "session" here does not refer to a persistent network connection but to a logical session created from HTTP requests and responses. The presence or absence of a persistent connection should have no effect on the use of cookie-derived sessions). To initiate a session, the origin server returns an extra response header to the client, Set-Cookie. (The details follow later.)

A user agent returns a Cookie request header (see below) to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send the client a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all. The origin server effectively ends a session by sending the client a Set-Cookie header with Max-Age=0. [TODO: Need to say something about Expires here.]

Servers may return a Set-Cookie response headers with any response. User agents should send Cookie request headers, subject to other rules detailed below, with every request.

An origin server may include multiple Set-Cookie headers in a response. Note that an intervening gateway MUST NOT fold multiple Set-Cookie headers into a single header. [TODO: Investigate how UAs cope with folded headers.]

5.2. Set-Cookie

TOC

5.2.1. Syntax

TOC

Informally, the Set-Cookie response header comprises the token Set-Cookie:, followed by a comma-separated list of one or more cookies. Each cookie begins with a name-value-pair, followed by zero or more semi-colon-separated attribute-value pairs. The NAME=VALUE attribute-value pair must come first in each cookie.

```
set-cookie-header = "Set-Cookie:" name-value-pairs
name-value-pairs  = name-value-pair *(";" name-value-pair)
name-value-pair   = name ["=" value]        ; optional value
name              = token
value             = token
```

[TODO: Investigate what token actually means.]
Attributes names are case-insensitive. White space is permitted between tokens. Note that although the above syntax description shows value as optional, some attributes require values.
The cookie-value is opaque to the user agent and MAY be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. "Opaque" implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone who examines the Set-Cookie header.
NOTE: The syntax above allows whitespace between the attribute and the = sign. Servers wishing to interoperate with some legacy user agents might wish to elide this extra white space to maximize compatibility.

5.3. Semantics

TOC

When the user agent receives a Set-Cookie header, the user agent stores the cookie in its cookie store. When the user agent makes another HTTP request to the origin server, the user agent will return the cookie in the Cookie header.
The server can override the default handling of cookies by specifying a number of cookie attributes. User agents ignore unrecognized cookie attributes.

5.3.1. Cookie Attributes

TOC

This section describes the semantics of a number of cookie attributes.

5.3.1.1. Max-Age

TOC

Syntax A sequence of ASCII numerals.

Semantics The value of the Max-Age attribute represents the maximum lifetime of the cookie, measured in seconds from the moment the user agent receives the cookie. If the server does not supply an Expires or a Max-Age attribute, the lifetime of the cookie is limited to the current session (as defined by the user agent).

5.3.1.2. Expires

TOC

Syntax An RFC 1123 date [cite]. (User agents use a very forgiving date parers; see Section [TODO]).

Semantics The value of the Expires attribute represents the maximum lifetime of the cookie, represented as the point in time at which the cookie expires. If the server does not supply an Expires or a Max-Age attribute, the lifetime of the cookie is limited to the current session (as defined by the user agent).

5.3.1.3. Domain

TOC

[TODO: Test Domain.] The Domain attribute specifies the domain for which the cookie is valid. The leading dot isn't required. If there is no Domain attribute, the default is to return the cookie only to the origin server. [TODO: You can only set cookies for related domains.]

5.3.1.4. Path

TOC

[TODO: Test path.] The Path attribute specifies the subset of URLs to which this cookie applies.

5.3.1.5. Secure

TOC

Syntax The empty string.

Semantics The user agent SHOULD protect the confidentiality of cookies with the Secure attribute.

5.3.1.6. HttpOnly

TOC

Syntax The empty string.

Semantics The user agent SHOULD protect confidentiality of cookies with the HttpOnly attribute by including HttpOnly cookies only when generating cookie strings for use in HTTP requests.

5.4. Cookie

TOC

5.4.1. Syntax

TOC

The user agent returns stored cookies to the origin server in the cookie header. The Cookie header shares a common syntax with the Set-Cookie header, but the semantics of the header differ dramatically.

```
cookie-header     = "Cookie:" name-value-pairs
name-value-pairs  = name-value-pair *(";" name-value-pair)
name-value-pair   = name "=" value
name              = token
value             = token
```

NOTE: If the server supplies a Set-Cookie header that does not conform to the grammar in Section TODO, the user agent might not supply a Cookie header that conforms to the grammar in this Section.

5.4.2. Semantics

TOC

Each name-value-pair represents a cookie stored by the user agent. The cookie name is returned in as the name and the cookie value is returned as the value.

TOC

5.5. Controlling Caching

[TODO: Should we go into this much detail here? This seems redundant with the HTTP specs.]
An origin server must be cognizant of the effect of possible caching of both the returned resource and the Set-Cookie header. Caching "public" documents is desirable. For example, if the origin server wants to use a public document such as a "front door" page as a sentinel to indicate the beginning of a session for which a Set-Cookie response header must be generated, the page should be stored in caches "pre-expired" so that the origin server will see further requests. "Private documents", for example those that contain information strictly private to a session, should not be cached in shared caches.
If the cookie is intended for use by a single user, the Set-Cookie header should not be cached. A Set-Cookie header that is intended to be shared by multiple users may be cached.
The origin server should send the following additional HTTP/1.1 response headers, depending on circumstances: [TODO: Is this good advice?]

* To suppress caching of the Set-Cookie header: Cache-control: no-cache="set-cookie".

and one of the following:

* To suppress caching of a private document in shared caches: Cache-Control: private.

* To allow caching of a document and require that it be validated before returning it to the client: Cache-Control: must-revalidate.

* To allow caching of a document, but to require that proxy caches (not user agent caches) validate it before returning it to the client: Cache-Control: proxy-revalidate.

* To allow caching of a document and request that it be validated before returning it to the client (by "pre-expiring" it): Cache-Control: max-age=0. Not all caches will revalidate the document in every case.

HTTP/1.1 servers must send Expires: old-date (where old-date is a date long in the past) on responses containing Set-Cookie response headers unless they know for certain (by out of band means) that there are no downsteam HTTP/1.0 proxies. HTTP/1.1 servers may send other Cache-Control directives that permit caching by HTTP/1.1 proxies in addition to the Expires: old-date directive; the Cache-Control directive will override the Expires: old-date for HTTP/1.1 proxies.

6. User Agent Conformance

TOC

Not all origin servers conform to the behavior specified in the previous section. To ensure interoperability, user agents MUST process cookies in a manner that is "black-box" indistinguishable from the requirements in this section.

6.1. Parsing the Set-Cookie Header

TOC

Let an LWS character be either a U+20 (SPACE) or a U+09 (TAB) character.
A user agent MUST use the following algorithm to parse the Set-Cookie header:

1. [TODO: Deal with ',' characters.]

2. If the header contains a ';' character:

    > the name-value string is characters up to, but not including, the first ';', and the unparsed-cookie-attributes are the remainder of the header (including the ';' in question).

    Otherwise:

    > the name-value string is all the character contained in the header, and the unparsed-cookie-attributes is the empty string.

3. If the first non-LWS character of the name-value string is '=', remove it.

4. If the name-value string contains a '=' character:

    > the name string is the characters up to, but not including, the first '=' character, and the value string is the characters after the first '=' character .

    Otherwise:

    > the name string is empty, and the value string is the entire name-value string.

5. Remove any leading or trailing space from the name string.

6. Remove any leading or trailing space from the value string.

7. The cookie-name is the name string.

8. The cookie-value is the value string.

The user agent MUST use the following algorithm to parse the unparsed-attributes:

1. [TODO: Figure out how to parse cookie attributes.]

[TODO: Can parsing a cookie ever fail?]
[TODO: Convert Max-Age to a date during parsing.]
When the user agent finishes parsing the Set-Cookie header, the user agent *receives a cookie* from the origin server with name cookie-name, value cookie-value, and attributes cookie-attributes.

6.2. Parsing Cookie Dates

TOC

Basically, cookie dates are a mess for historical reasons.
To be compatible with legacy servers, however, user agents should accept dates formated according to this grammar:

```
cookie-date       = rfc1123-like-date / mystery-date
rfc1123-like-date = weekday "," SP rfc1123-like-dmy SP time SP "GMT"
weekday           = "Monday" / "Mon" / "Tuesday" / "Tue" / ...
rfc1123-like-dmy  = day dmy-div month dmy-div year
dmy-div           = SP / "-"
day               = 2DIGIT / *1SP DIGIT
month             = "Jan" / "Feb" / ...
year              = 2DIGIT / 4DIGIT
time              = 2DIGIT ":" 2DIGIT ":" 2DIGIT

mystery-date      = *CHAR ; see below
```

[TODO: More information about mystery-date.]

6.3. Storage Model

TOC

When the user agent receives a cookie, the user agent SHOULD record the cookie in its cookie store as follows.
A user agent MAY ignore received cookies in their entirety if the user agent is configured to block receiving cookie for a particular response. For example, the user agent might wish to block receiving cookies from "third-party" responses.

The user agent stores the following fields about each cookie:

- *name (a sequence of bytes)

- *value (a sequence of bytes)

- *expiry (a date)

- *domain (a cookie-domain)

- *path (a cookie-path)

- *creation (a date)

- *last-access (a date)

- *persistent (a Boolean)

- *host-only (a Boolean)

- *secure-only (a Boolean)

- *http-only (a Boolean)

When the user agent receives a cookie, the user agent MUST follow the following algorithm:

1. Create a new cookie based on the parsed Set-Cookie header:

    1. Create a new cookie with the following default field values:

        - *name = the cookie-name

        - *value = the cookie-value

        - *expiry = the latest representable date

        - *domain = the request-host

        - *path = the path of the request URL that generated the Set-Cookie response, up to, but not including, the right-most / [TODO: Test! This seems wrong for paths that are just a single slash]

        - *last-access = the date and time the cookie was received

        - *last-access = the date and time the cookie was received

        - *persistent = false

*host-only = true

*secure-only = false

*http-only = false

2. Update the default field values according to the cookie-attributes:

> **expiry** If the cookie-attributes contains at least one Expires or a Max-Age attribute, store the value of the [TODO: first] such attribute in the expiry field. Store the value true in the persistent field.
>
> **domain** If the cookie-attributes contains at least one Domain attribute, store the value of the [TODO: first] such attribute in the domain field. Store the value false in the host-only field. [TODO: Reject cookies for unrelated domains.] [TODO: If the URL's host is an IP address, let Domain to be an IP address if it matches the URL's host exactly, but set the host-only flag.]
>
> **path** If the cookie-attributes contains at least one Path attribute, store the value of the [TODO: first] such attribute in the path field.
>
> **secure-only** If the cookie-attributes contains at least one Secure attribute, store the value true in the secure-only field.
>
> **http-only** If the cookie-attributes contains at least one HttpOnly attribute, store the value true in the http-only field.

2. Remove from the cookie store all cookies that have the share the same name, domain, path, and host-only fields as the newly created cookie. [TODO: Valiate this list!] [TODO: There's some funny business around http-only here.]

3. Insert the newly created cookie into the cookie store.

The user agent MUST evict a cookie from the cookie store if either of the following conditions are met:

*A cookie exists in the cookie store with an expiry date in the past.

*More than 50 cookies exist in the cookie store with the same domain field.

The user agent MAY evict cookies from the cookie store if the cookie store exceeds some maximum storage bound (such as 3000 cookies). When the user agent evicts cookies from the cookie store, the user agent MUST evict cookies in the following priority order:

1. A cookie with an expiry date in the past.

2. A cookie that shares a domain field with more than 50 other cookies in the cookie store.

3. All other cookies.

If two cookies have the same removal priority, the user agent MUST evict the cookie with the least recent last-access date first. When the user agent exits, the user agent MUST remove from the cookie store all cookies with the persistent field set to false.

6.4. The Cookie Header

When the user agent generates an HTTP request for a particular URI, the user agent SHOULD attach exactly one HTTP named Cookie if the cookie-string (defined below) for that URI is non-empty. A user agent MAY elide the Cookie header in its entirety if the user agent is configured to block sending cookie for a particular request. For example, the user agent might wish to block sending cookies during "third-party" requests. When generating a cookie-string from a URI with a "secure" scheme, the user agent MUST set the SECURE flag to true. Otherwise, the user agent MUST set the SECURE flag to false.

> NOTE: The notion of an "secure" scheme is not defined by this document. Typically, user agents consider a scheme secure if the scheme refers to a protocol that makes use of transport-layer security, such as TLS. For example, most user agents consider "https" to be a secure scheme.

When generating a cookie-string for use in an HTTP request, the user agent MUST set the HTTP flag to true. Otherwise, the user agent MUST set the HTTP flag to false. The user agent MUST use the following algorithm to compute the cookie-string from a cookie store and from a URI:

1. Let cookie-list be the set of cookies from the cookie store that meet the following requirements:

   *The cookie's domain field must domain-match the URI's host. [TODO: Spec me]

* The cookie's path field must path-match the URI's path. [TODO: Spec me]

* If the cookie's host-only flag is set, the cookie's domain field must denote exactly the same FQDN as the URI's host. [TODO: Internet Explorer does not implement this requirement but most other major implementations do.]

* If the cookie's secure-only field is true, then the SECURE flag must be true.

* If the cookie's http-only field is true, then the HTTP flag must be true.

NOTE: The Cookie header will not contain any expired cookies because cookies past their expiry date are removed from the cookie store immediately.

2. Sort the cookie-list in the following order:

   * Cookies with longer path fields are listed before cookies with shorter path field.

   * Among cookies that have equal length path fields, cookies with earlier creation dates are listed before cookies with later creation dates.

3. Update the last-access field of each cookie in the cookie-list to the current date.

4. Serialize the cookie-list into a cookie-string by processing each cookie in the cookie-list in order:

   1. Output the cookie's name field.

   2. Output the character U+3D ("=")

   3. Output the cookie's value field.

   4. If there is an unprocessed cookie in the cookie-list, output the characters U+3B and U+20 ("; ")

---

7. Caching Proxy Conformance

One reason for separating state information from both a URL and document content is to facilitate the scaling that caching permits. To support cookies, a caching proxy must obey these rules already in the

HTTP specification [TODO: If they're already in the HTTP specification, aren't they redundant here?]:

- *Honor requests from the cache, if possible, based on cache validity rules.

- *Pass along a Cookie request header in any request that the proxy must make of another server.

- *Return the response to the client. Include any Set-Cookie response header.

- *Cache the received response subject to the control of the usual headers, such as Expires, Cache-Control: no-cache, and Cache-Control: private.

- *Cache the Set-Cookie subject to the control of the usual header, Cache-Control: no-cache="set-cookie". (The Set-Cookie header should usually not be cached.)

Proxies must not introduce Set-Cookie (Cookie) headers of their own in proxy responses (requests).

---

8. Examples

TOC

[TODO: Write sensible examples.]

---

9. Implementation Considerations

TOC

Here we speculate on likely or desirable details for an origin server that implements state management.

9.1. Set-Cookie Content

TOC

An origin server's content should probably be divided into disjoint application areas, some of which require the use of state information. The application areas can be distinguished by their request URLs. The Set-Cookie header can incorporate information about the application areas by setting the Path attribute for each one.
The session information can obviously be clear or encoded text that describes state. However, if it grows too large, it can become unwieldy. Therefore, an implementor might choose for the session

information to be a key to a server-side resource. [TODO: Describe briefly how to generate a decent session key.]
[TODO: We could recommend that servers encrypt and mac their cookie data.]
[TODO: Mention issues that arise from having multiple concurrent sessions.]

9.2. Implementation Limits

TOC

Practical user agent implementations have limits on the number and size of cookies that they can store. In general, user agents' cookie support should have no fixed limits. [TODO: Why not?] They should strive to store as many frequently-used cookies as possible. Furthermore, general-use user agents should provide each of the following minimum capabilities individually, although not necessarily simultaneously: [TODO: Where do these numbers come from?]

* at least 4096 bytes per cookie (as measured by the size of the characters that comprise the cookie non-terminal in the syntax description of the Set-Cookie header)

User agents created for specific purposes or for limited-capacity devices should provide at least 50 cookies of 4096 bytes, to ensure that the user can interact with a session-based origin server.
The information in a Set-Cookie response header must be retained in its entirety. If for some reason there is inadequate space to store the cookie, it must be discarded, not truncated.
Applications should use as few and as small cookies as possible, and they should cope gracefully with the loss of a cookie. [TODO: Could mention latency issues that arise from having tons of cookies.]

9.2.1. Denial of Service Attacks

TOC

User agents may choose to set an upper bound on the number of cookies to be stored from a given host or domain name or on the size of the cookie information. Otherwise, a malicious server could attempt to flood a user agent with many cookies, or large cookies, on successive responses, which would force out cookies the user agent had received from other servers. However, the minima specified above should still be supported. [TODO: These minima still let an attacker exhaust the entire cookie store. There's not much we can do about it though.]

10. Privacy

TOC

10.1. User Agent Control

TOC

An origin server could create a Set-Cookie header to track the path of a user through the server. Users may object to this behavior as an intrusive accumulation of information, even if their identity is not evident. (Identity might become evident if a user subsequently fills out a form that contains identifying information.) This state management specification therefore requires that a user agent give the user control over such a possible intrusion, although the interface through which the user is given this control is left unspecified. However, the control mechanisms provided shall at least allow the user

- to completely disable the sending and saving of cookies,
- to determine whether a stateful session is in progress, and
- to control the saving of a cookie on the basis of the cookie's Domain attribute.

Such control could be provided by, for example, mechanisms

- to notify the user when the user agent is about to send a cookie to the origin server, offering the option not to begin a session,
- to display a visual indication that a stateful session is in progress,
- to let the user decide which cookies, if any, should be saved when the user concludes a window or user agent session, or
- to let the user examine the contents of a cookie at any time.

A user agent usually begins execution with no remembered state information. It should be possible to configure a user agent never to send Cookie headers, in which case it can never sustain state with an origin server. (The user agent would then behave like one that is unaware of how to handle Set-Cookie response headers.)
When the user agent terminates execution, it should let the user discard all state information. Alternatively, the user agent may ask the user whether state information should be retained. If the user chooses to retain state information, it would be restored the next time the user agent runs.

10.2. Protocol Design

TOC

The restrictions on the value of the Domain attribute are meant to reduce the ways that cookies can "leak" to the "wrong" site. The intent is to restrict cookies to one, or a closely related set of hosts. Therefore a request-host is limited as to what values it can set for Domain.

11. Security Considerations

TOC

11.1. Clear Text

TOC

The information in the Set-Cookie and Cookie headers is transmitted in the clear. Three consequences are:

1. Any sensitive information that is conveyed in in the headers is exposed to an easedropper.
2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.
3. A malicious client could alter the Cookie header before transmission, with unpredictable results.

These facts imply that information of a personal and/or financial nature should be sent over a secure channel. For less sensitive information, or when the content of the header is a database key, an origin server should be vigilant to prevent a bad Cookie value from causing failures.

11.2. Cookie Spoofing

TOC

[TODO: Mention integrity issue where a sibling domain can inject cookies.]
[TODO: Mention integrity issue where a HTTP can inject cookies into HTTPS.]

11.3. Unexpected Cookie Sharing

TOC

A user agent should make every attempt to prevent the sharing of session information between hosts that are in different domains. Embedded or inlined objects may cause particularly severe privacy problems if they can be used to share cookies between disparate hosts. For example, a malicious server could embed cookie information for host a.com in a URI for host b.com. User agent implementors are strongly encouraged to prevent this sort of exchange whenever possible. [TODO: How are they supposed to do this? This section makes little sense.]

---

12. Other, Similar, Proposals

TOC

[TODO: Describe relation to the Netscape Cookie Spec, RFC 2109, RFC 2629, and cookie-v2.]

Appendix A. Acknowledgements

TOC

This document borrows heavily from RFC 2109. [TODO: Figure out the proper way to credit the authors of RFC 2109.]

Appendix B. Tabled Items

TOC

Tabled items:

*Public suffix.

Author's Address

TOC

	Adam Barth
	University of California, Berkeley
Email:	abarth@eecs.berkeley.edu
URI:	http://www.adambarth.com/