http-state Working Group	A. Barth	
Internet-Draft	U.C. Berkeley	
Expires: February 9, 2010	August 08, 2009	

TOC

HTTP State Management Mechanism
draft-abarth-cookie-00

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 appriciated.

Table of Contents

---

1. Introduction

TOC

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

TOC

2. Terminology

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. Outline

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.

4.1. Syntax: General

TOC

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.
[TODO: Test this grammar. I think there are many, many issue with this grammer. For example, this grammar seems to permit whitespace around the "=", but I don't think that actually works.]

```
av-pairs        =       av-pair *(";" av-pair)
av-pair         =       attr ["=" value]        ; optional value
attr            =       token
value           =       word
word            =       token | quoted-string
```

Attributes (names) (attr) are case-insensitive. White space is permitted between tokens. Note that while the above syntax description shows value as optional, most attrs require them.
NOTE: The syntax above allows whitespace between the attribute and the = sign. [TODO: This is probably wrong, however.]

4.2. Origin Server Role

TOC

4.2.1. 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 could fold multiple such headers into a single header. [TODO: Investigate how UAs cope with such folded headers.]

4.2.2. Set-Cookie Syntax

TOC

The syntax for the Set-Cookie response header is

[TODO: Valdiate this syntax.]

```
      set-cookie      =       "Set-Cookie:" cookies
      cookies         =       1#cookie
      cookie          =       NAME "=" VALUE *(";" cookie-av)
      NAME            =       attr
      VALUE           =       value
      cookie-av       =       "Comment" "=" value
                      |       "Domain" "=" value
                      |       "Max-Age" "=" value
                              [TODO: Expires is clearly missing.]
                      |       "Path" "=" value
                      |       "Secure"
                              [TODO: HTTPOnly is also missing.]
                      |       "Version" "=" 1*DIGIT
                              [TODO: Version is likely a fantasy.]
```

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 specific attributes and the semantics of their values follows. The NAME=VALUE attribute-value pair must come first in each cookie. The others, if present, can occur in any order. If an attribute appears more than once in a cookie, the behavior is undefined. [TODO: Test what happens when attributes are multiply defined.]

> NAME=VALUE
>
> > Required. The name of the state information ("cookie") is NAME, and its value is VALUE. NAMEs that begin with $ are reserved for other uses and must not be used by applications. [TODO: I suspect the $ rule is a fantasy.] The 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 that examines the Set-Cookie header.
>
> Comment=comment
>
> > Optional. Because cookies can contain private information about a user, the Cookie attribute allows an origin server to document its intended use of a cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie. [TODO: Does this actually exist?]
>
> Domain=domain

> Optional. The Domain attribute specifies the domain for which the cookie is valid. An explicitly specified domain must always start with a dot. [TODO: Test what happens without a dot.]

Max-Age=delta-seconds

> Optional. The Max-Age attribute defines the lifetime of the cookie, in seconds. The delta-seconds value is a decimal non-negative integer. [TODO: Test negative integers.] After delta-seconds seconds elapse, the client should discard the cookie. A value of zero means the cookie should be discarded immediately.

Path=path

> Optional. The Path attribute specifies the subset of URLs to which this cookie applies.

Secure

> Optional. The Secure attribute (with no value) directs the user agent to use only (unspecified) secure means to contact the origin server whenever it sends back this cookie. [TODO: We should give better implementation advice than this.]
>
> The user agent (possibly under the user's control) may determine what level of security it considers appropriate for "secure" cookies. The Secure attribute should be considered security advice from the server to the user agent, indicating that it is in the session's interest to protect the confidentiality of the cookie's value.

Version=version

> Required [TODO: Unlikely]. The Version attribute, a decimal integer, identifies to which version of the state management specification the cookie conforms. For this specification, Version=1 applies. [TODO: Remove this attribute.]

4.2.3. Controlling Caching

TOC

[TODO: Should we go into this much detail here? This seems redudant 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.

4.3. User Agent Role

TOC

TOC

4.3.1. Interpreting Set-Cookie

The user agent keeps separate track of state information that arrives via Set-Cookie response headers from each origin server (as distinguished by name or IP address and port). The user agent applies these defaults for optional attributes that are missing:

> **Version** Defaults to "old cookie" behavior as originally specified by Netscape. See the HISTORICAL section. [TODO: Unlikely.]
>
> **Domain** Defaults to the request-host. (Note that there is no dot at the beginning of request-host.) [TODO: This is important to test!]
>
> **Max-Age** The default behavior is to discard the cookie when the user agent exits. [TODO: Interaction with Expires.]
>
> **Expires** The default behavior is to discard the cookie when the user agent exits. [TODO: Interaction with Max-Age.]
>
> **Path** Defaults to 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]
>
> **Secure** If absent, the user agent may send the cookie over an insecure channel.

4.3.2. Rejecting Cookies

TOC

To prevent possible security or privacy violations, a user agent must reject a cookie (shall not store its information) if any of the following is true:

*The value of the Path attribute is not a prefix of the request-URI. [TODO: This is a lie.]

*The value for the Domain attribute contains no embedded dots or does not start with a dot.

*The value for the request-host does not domain-match the Domain attribute. [TODO: Test whether you can set a cookie for a subdomain of yourself.]

*The request-host is a FQDN (not IP address) and has the form HD, where D is the value of the Domain attribute, and H is a string

that contains one or more dots. [TODO: I don't think this is right. foo.bar.baz.com can set a cookie for .baz.com]

*[TODO: Need to interact with public suffix list!]

Examples:

*A Set-Cookie from request-host y.x.foo.com for Domain=.foo.com would be rejected, because H is y.x and contains a dot. [TODO: I don't think this is right.]

*A Set-Cookie from request-host x.foo.com for Domain=.foo.com would be accepted.

*A Set-Cookie with Domain=.com or Domain=.com., will be rejected, because there is no embedded dot.

*A Set-Cookie with Domain=foo.com will be rejected because the value for Domain does not begin with a dot. [TODO: This seems unlikely, but test!]

*A Set-Cookie with Domain=.co.uk will be rejected because .co.uk is a public suffix.

4.3.3. Cookie Management

TOC

If a user agent receives a Set-Cookie response header whose NAME is the same as a pre-existing cookie, and whose Domain and Path attribute values exactly (string) match those of a pre-existing cookie, the new cookie supersedes the old. However, if the Set-Cookie has a value for Max-Age of zero, the (old and new) cookie is discarded. Otherwise cookies accumulate until they expire (resources permitting), at which time they are discarded. [TODO: Do cookies really accumulate like this? Also, need to talk about Expires]

Because user agents have finite space in which to store cookies, they may also discard older cookies to make space for newer ones, using, for example, a least-recently-used algorithm, along with constraints on the maximum number of cookies that each origin server may set. [TODO: Consider recommending a cookie eviction strategy that works in practice.]

If a Set-Cookie response header includes a Comment attribute, the user agent should store that information in a human-readable form with the cookie and should display the comment text as part of a cookie inspection user interface. [TODO: I think the Comment attribute is a fantasy.]

User agents should allow the user to control cookie destruction. An infrequently-used cookie may function as a "preferences file" for network applications, and a user may wish to keep it even if it is the least-recently-used cookie. One possible implementation would be an interface that allows the permanent storage of a cookie through a checkbox (or, conversely, its immediate destruction). [TODO: Remove?] Privacy considerations dictate that the user have considerable control over cookie management. The PRIVACY section contains more information.

4.3.4. Sending Cookies to the Origin Server

TOC

When it sends a request to an origin server, the user agent sends a Cookie request header to the origin server if it has cookies that are applicable to the request, based on

- *the request-host,
- *the request-URI, and
- *the cookie's age.

The syntax for the header is:

```
cookie          =       "Cookie:" cookie-version
                        1*((";" | ",") cookie-value)
cookie-value    =       NAME "=" VALUE [";" path] [";" domain]
cookie-version  =       "$Version" "=" value
NAME            =       attr
VALUE           =       value
path            =       "$Path" "=" value
domain          =       "$Domain" "=" value
```

[TODO: This syntax is entirely wrong.]
The following rules apply to choosing applicable cookie-values from among all the cookies the user agent has.

Domain Selection

> The origin server's fully-qualified host name must domain-match the Domain attribute of the cookie.

Path Selection

> The Path attribute of the cookie must match a prefix of the request-URI. [TODO: Need a more complex algorithm here involving the / character.]

Max-Age Selection

> Cookies that have expired should have been discarded and thus are not forwarded to an origin server.

If multiple cookies satisfy the criteria above, they are ordered in the Cookie header such that those with more specific Path attributes precede those with less specific. Ordering with respect to other attributes (e.g., Domain) is unspecified. [TODO: Figure out the correct ordering.]
Note: For backward compatibility, the separator in the Cookie header is semi-colon (;) everywhere. A server should also accept comma (,) as the separator between cookie-values for future compatibility. [TODO: Test whether servers actually do this.]

4.3.5. Sending Cookies in Unverifiable Transactions

TOC

[TODO: This entire section seems like a fantasy.]
[TODO: Consider explaining how third-party cookie blocking works.]

4.4. How an Origin Server Interprets the Cookie Header

TOC

[TODO: This section appears to be nonsense.]

4.5. Caching Proxy Role

TOC

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).

---

5. Examples

TOC

5.1. Example 1

TOC

Most detail of request and response headers has been omitted. Assume the user agent has no stored cookies.

1. User Agent -> Server

```
POST /acme/login HTTP/1.1
[form data]
```

   User identifies self via a form.

2. Server -> User Agent

```
HTTP/1.1 200 OK
Set-Cookie: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
```

   Cookie reflects user's identity. [TODO: This is insecure.]

3. User Agent -> Server

```
POST /acme/pickitem HTTP/1.1
Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme"
[form data]
```

User selects an item for "shopping basket."

4. Server -> User Agent

```
HTTP/1.1 200 OK
Set-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1"; Path="/acme"
```

Shopping basket contains an item.

5. User Agent -> Server

```
POST /acme/shipping HTTP/1.1
Cookie: $Version="1";
        Customer="WILE_E_COYOTE"; $Path="/acme";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme"
[form data]
```

User selects shipping method from form.

6. Server -> User Agent

```
HTTP/1.1 200 OK
Set-Cookie: Shipping="FedEx"; Version="1"; Path="/acme"
```

New cookie reflects shipping method.

7. User Agent -> Server

```
POST /acme/process HTTP/1.1
Cookie: $Version="1";
        Customer="WILE_E_COYOTE"; $Path="/acme";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme";
        Shipping="FedEx"; $Path="/acme"
[form data]
```

User chooses to process order.

8. Server -> User Agent

```
HTTP/1.1 200 OK
```

Transaction is complete.

[TODO: This example is really silly. We shouldn't be recommending this at all.]

The user agent makes a series of requests on the origin server, after each of which it receives a new cookie. All the cookies have the same Path attribute and (default) domain. Because the request URLs all have /acme as a prefix, and that matches the Path attribute, each request contains all the cookies received so far.

5.2. Example 2

TOC

This example illustrates the effect of the Path attribute. All detail of request and response headers has been omitted. Assume the user agent has no stored cookies.
Imagine the user agent has received, in response to earlier requests, the response headers

```
Set-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1";
        Path="/acme"
```

and

```
Set-Cookie: Part_Number="Riding_Rocket_0023"; Version="1";
        Path="/acme/ammo"
```

A subsequent request by the user agent to the (same) server for URLs of the form /acme/ammo/... would include the following request header:

```
Cookie: $Version="1";
        Part_Number="Riding_Rocket_0023"; $Path="/acme/ammo";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme"
```

Note that the NAME=VALUE pair for the cookie with the more specific Path attribute, /acme/ammo, comes before the one with the less specific Path attribute, /acme. Further note that the same cookie name appears more than once.
A subsequent request by the user agent to the (same) server for a URL of the form /acme/parts/ would include the following request header:

```
Cookie: $Version="1"; Part_Number="Rocket_Launcher_0001"; $Path="/acme"
```

Here, the second cookie's Path attribute /acme/ammo is not a prefix of the request URL, /acme/parts/, so the cookie does not get forwarded to the server.

TOC

6. Implementation Considerations

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

6.1. 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.]

6.2. 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 300 cookies
- 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)
- at least 20 cookies per unique host or domain name

User agents created for specific purposes or for limited-capacity devices should provide at least 20 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.]

6.2.1. 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.]

---

7. Privacy

TOC

7.1. 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.

7.2. 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.

---

8. Security Considerations

TOC

8.1. 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.

8.2. 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.]

8.3. 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.]

---

9. 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.]

Author's Address

TOC

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