HTTP Working Group | David M. Kristol
INTERNET DRAFT | Bell Laboratories, Lucent Technologies
Obsoletes: RFC 2109 | Lou Montulli
Netscape Communications

<draft-ietf-http-state-man-mec-03.ps>
July 30, 1997 | Expires January 30, 1998

HTTP State Management Mechanism (Rev1)

Status of this Memo

This is authors’ draft 2.68.

1. ABSTRACT

This document specifies a way to create a stateful session with HTTP requests and responses. It describes two new headers, `Cookie` and `Set-Cookie2`, which carry state information between participating origin servers and user agents. The method described here differs from Netscape’s Cookie proposal, but it can interoperate with HTTP/1.0 user agents that use Netscape’s method. (See the **HISTORICAL** section.)

This document reflects implementation experience with RFC 2109 and obsoletes it.

2. TERMINOLOGY

The terms *user agent*, *client*, *server*, *proxy*, and *origin server* have the same meaning as in the HTTP/1.1 specification [RFC 2068].

Host name (HN) means either the host domain name (HDN) 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.

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 is a HN.

The term *request-port* refers to the `port` portion of the `absoluteURI` (`http_URL`) of the HTTP request line. If the `absoluteURI` has no explicit `port`, the request-port is the HTTP default, 80. The *request-port of a cookie* is the request-port of the request in which a `Set-Cookie2` response header was returned to the user agent.

Host names can be specified either as an IP address or a HDN string. Sometimes we compare one host name with another. Host A’s name *domain-matches* host B’s if

- their host name strings match exactly; or

- A is a HDN string and has the form NB, where N is a non-empty name string, B has the form .B’, and B’ is a HDN 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.

The *reach* R of a host name H is defined as follows:

- If

  — H is the host domain name of a host; and,

  — H has the form A.B; and

  — A has no embedded dots; and

  — B has at least one embedded dot
  then the reach of H is .B.

- Otherwise, the reach of H is H.

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

This document describes a way to create stateful sessions with HTTP requests and responses. Currently, 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

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 goal is to have a minimal impact on HTTP and user agents. Only origin servers that need to maintain sessions would suffer any significant impact, and that impact can largely be confined to Common Gateway Interface (CGI) programs, unless the server provides more sophisticated state management support. (See **Implementation Considerations**, below.)

4.1 Syntax: General

The two state management headers, `Set-Cookie2` and `Cookie`, have common syntactic properties involving attribute-value pairs. The following grammar uses the notation, and tokens `DIGIT` (decimal digits), `token` (informally, a sequence of non-special, non-white space characters), and `http_URL` from the HTTP/1.1 specification [RFC 2068] to describe their syntax.

```
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.

4.2 Origin Server Role

4.2.1 General 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-Cookie2.` (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 back to the client a `Set-Cookie2` response header with the same or different information, or it may send no `Set-Cookie2` header at all. The origin server effectively ends a session by sending the client a `Set-Cookie2` header with `Max-Age=0`.

Servers may return `Set-Cookie2` 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-Cookie2` headers in a response. Note that an intervening gateway could fold multiple such headers into a single header.

4.2.2 `Set-Cookie2` *Syntax* The syntax for the `Set-Cookie2` response header is

```
set-cookie     =    "Set-Cookie2:" cookies
cookies        =    1#cookie
cookie         =    NAME "=" VALUE *(";" set-cookie-av)
NAME           =    attr
VALUE          =    value
set-cookie-av  =    "Comment" "=" value
               |    "CommentURL" "=" <"> http_URL <">
               |    "Discard"
               |    "Domain" "=" value
               |    "Max-Age" "=" value
               |    "Path" "=" value
               |    "Port" [ "=" <"> 1#portnum <"> ]
               |    "Secure"
               |    "Version" "=" 1*DIGIT
portnum        =    1*DIGIT
```

Informally, the `Set-Cookie2` response header comprises the token `Set-Cookie2:`, 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 syntax for attribute-value pairs was shown earlier. 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, only the value associated with the first appearance of the attribute shall be used; subsequent values after the first must be ignored.

NAME=*VALUE*

Required. The name of the state information (‘‘cookie’’) is *NAME*, and its value is *VALUE*. *NAME*s that begin with `$` are reserved for other uses and must not be used by applications.

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-Cookie2` header.

`Comment=`*comment*
: Optional. Because cookies can be used to derive or store private information about a user, the `Comment` attribute allows an origin server to document how it intends to use the cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie.

`CommentURL="`*http_URL*`"`
: Optional. Because cookies can be used to derive or store private information about a user, the `CommentURL` attribute alows an origin server to document how it intends to use the cookie. The user can inspect the information identified by the URL to decide whether to initiate or continue a session with this cookie.

`Discard`
: Optional. The `Discard` attribute instructs the user agent to discard the cookie unconditionally when the user agent terminates.

`Domain=`*domain*
: Optional. The `Domain` attribute specifies the domain for which the cookie is valid. If an explicitly specified *domain* does not start with a dot, the user agent supplies a leading 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. 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 on the origin server to which this cookie applies.

`Port[="`*port-list*`"]`
: Optional. The `Port` attribute restricts the port to which a cookie may be returned in a `Cookie` request header. Note that the syntax **requires** quotes around the optional *port-list* even if there is only one `portnum` in *port-list*.

`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, to protect the confidentially and authenticity of the information in the cookie.

: 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 cookie contents. When it sends a "secure" cookie back to a server, the user agent should use no less than the same level of security as was used when it received the cookie from the server.

`Version=`*version*
: **Required**. The `Version` attribute, a decimal integer, identifies to which version of the state management specification the cookie conforms. For this specification, `Version=1` applies.

4.2.3 Controlling Caching An origin server must be cognizant of the effect of possible caching of both the returned resource and the `Set-Cookie2` 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-Cookie2` 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-Cookie2` header should not be cached. A `Set-Cookie2` 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:

- To suppress caching of the `Set-Cookie2` header: `Cache-control: no-cache="set-cookie2"`.

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, max-age=0`.
- 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, max-age=0`.
- 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-Cookie2` response headers unless they know for certain (by out of band means) that there are no upstream 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

4.3.1 Interpreting `Set-Cookie2` The user agent keeps separate track of state information that arrives via `Set-Cookie2` response headers from each origin server (as distinguished by name or IP address and port). The user agent must ignore attribute-value pairs whose attribute it does not recognize. The user agent applies these defaults for optional attributes that are missing:

`Discard` The default behavior is dictated by the presence or absence of a `Max-Age` attribute.

`Domain` Defaults to the request-host. (Note that there is no dot at the beginning of request-host.)

`Max-Age` The default behavior is to discard the cookie when the user agent exits.

`Path` Defaults to the path of the request URL that generated the `Set-Cookie2` response, up to, but not including, the right-most `/`.

`Port` The default behavior is that a cookie may be returned to any request-port.

`Secure` If absent, the user agent may send the cookie over an insecure channel.

4.3.2 Rejecting Cookies To prevent possible security or privacy violations, a user agent rejects a cookie (shall not store its information) if any of the following is true of the attributes explicitly present in the `Set-Cookie2` response header:

- The value for the `Path` attribute is not a prefix of the request-URI.
- The value for the `Domain` attribute contains no embedded dots.
- The value for the request-host does not domain-match the `Domain` attribute.
- The request-host is a HDN (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.
- The `Port` attribute has a `"port-list"`, and the request-port was not in the list.

Examples:

- A `Set-Cookie2` from request-host `y.x.foo.com` for `Domain=.foo.com` would be rejected, because H is `y.x` and contains a dot.
- A `Set-Cookie2` from request-host `x.foo.com` for `Domain=.foo.com` would be accepted.
- A `Set-Cookie2` with `Domain=.com` or `Domain=.com.`, will always be rejected, because there is no embedded dot.
- A `Set-Cookie2` with `Domain=ajax.com` will be accepted, and the value for `Domain` will be taken to be `.ajax.com`, because a dot gets prepended to the value.
- A `Set-Cookie2` with `Port="80,8000"` will be accepted if the request was made to port 80 or 8000 and will be rejected otherwise.

4.3.3 Cookie Management If a user agent receives a `Set-Cookie2` 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-Cookie2` has a value for `Max-Age` of zero, the (old and new) cookie is discarded. Otherwise a cookie persists (resources permitting) until whichever happens first, then gets discarded: its `Max-Age` lifetime is exceeded; or, if the `Discard` attribute is set, the user agent terminates the session,

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.

If a `Set-Cookie2` 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.

If a `Set-Cookie2` response header includes a `CommentURL` attribute, the user agent should store that information in a human-readable form with the cookie, or, preferably, should allow the user to follow the `http_URL` link as part of a cookie inspection user interface.

The cookie inspection user interface may include a facility whereby a user can decide, at the time the user agent receives the `Set-Cookie2` response header, whether or not to accept the cookie. A potentially confusing situation could arise if the following sequence occurs:

- the user agent receives a cookie that contains a CommentURL attribute;
- the user agent's cookie inspection interface is configured so that it presents a dialog to the user before the user agent accepts the cookie;
- the dialog allows the user to follow the CommentURL link when the user agent receives the cookie; and,
- when the user follows the CommentURL link, the origin server (or another server, via other links in the returned content) returns another cookie.

The user agent should not send any cookies in this context. The user agent may discard any cookie it receives in this context that the user has not, through some user agent mechanism, deemed acceptable.

User agents should allow the user to control cookie destruction, but they must not extend the cookie's lifetime beyond that controlled by the `Discard` and `Max-Age` attributes. 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).

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 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 and request-port;
- the request-URI;
- the cookie's age.

The syntax for the header is:

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

The value of the cookie-version attribute must be the value from the Version attribute of the corresponding Set-Cookie2 response header. Otherwise the value for cookie-version is 0. The value for the path attribute must be the value from the Path attribute, if one was present, of the corresponding Set-Cookie2 response header. Otherwise the attribute should be omitted from the Cookie request header. The value for the domain attribute must be the value from the Domain attribute, if one was present, of the corresponding Set-Cookie2 response header. Otherwise the attribute should be omitted from the Cookie request header.

The port attribute of the Cookie request header must mirror the Port attribute, if one was present, in the corresponding Set-Cookie2 response header. That is, the port attribute must be present if the Port attribute was present in the Set-Cookie2 header, and it must have the same value, if any. Otherwise, if the Port attribute was absent from the Set-Cookie2 header, the attribute likewise must be omitted from the Cookie request header.

Note that there is neither a Comment nor a CommentURL attribute in the Cookie request header corresponding to the ones in the Set-Cookie2 response header. The user agent does not return the comment information to the origin server.

The user agent applies the following rules to choose applicable cookie-values to send in Cookie request headers from among all the cookies it has received.

Domain Selection
: The origin server's host name must domain-match the Domain attribute of the cookie.

Port Selection
: There are three possible behaviors, depending on the Port attribute in the Set-Cookie2 response header:

1. By default (no Port attribute), the cookie may be sent to any port.
2. If the attribute is present but has no value (e.g., Port), the cookie must only be sent to the request-port it was received from.
3. If the attribute has a port-list, the cookie must only be returned if the new request-port is one of those listed in port-list.

Path Selection
: The Path attribute of the cookie must match a prefix of the request-URI.

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.

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.

4.3.5 Sending Cookies in Unverifiable Transactions Users must have control over sessions in order to ensure privacy. (See **PRIVACY** section below.) To simplify implementation and to prevent an additional layer of complexity where adequate safeguards exist, however, this document distinguishes between transactions that are verifiable and those that are unverifiable. A transaction is *verifiable* if the user, or a user-designated agent, has the option to review the request-URI prior to its use in the transaction. A transaction is *unverifiable* if the user does not have that option. Unverifiable transactions typically arise when a user agent automatically requests inlined or embedded entities or when it resolves redirection (3xx) responses from an origin server. Typically the *origin transaction*, the transaction that the user initiates, is verifiable, and that transaction may directly or indirectly induce the user agent to make unverifiable transactions.

An unverifiable transaction is to a *third-party host* if its request-host U does not domain-match the reach R of the request-host O in the origin transaction. When it makes an unverifiable transaction, a user agent must disable all cookie processing (*i.e.*, must not send cookies, and must not accept any received cookies) if the transaction is to a third-party host.

This restriction prevents a malicious service author from using unverifiable transactions to induce a user agent to start or continue a session with a server in a different domain. The starting or continuation of such sessions could be contrary to the privacy expectations of the user, and could also be a security problem.

User agents may offer configurable options that allow the user agent, or any autonomous programs that the user agent executes, to ignore the above rule, so long as these override options default to ‘‘off.’’

(**N.B.** Mechanisms may be proposed that will automate overriding the third-party restrictions under controlled conditions.)

Many current user agents already provide a review option that would render many links verifiable. For instance, some user agents display the URL that would be referenced for a particular link when the mouse pointer is placed over that link. The user can therefore determine whether to visit that site before causing the browser to do so. (Though not implemented on current user agents, a similar technique could be used for a button used to submit a form -- the user agent could display the action to be taken if the user were to select that button.) However, even this would not make all links verifiable; for example, links to automatically loaded images would not normally be subject to ‘‘mouse pointer’’ verification.

Many user agents also provide the option for a user to view the HTML source of a document, or to save the source to an external file where it can be viewed by another application. While such an option does provide a crude review mechanism, some users might not consider it acceptable for this purpose.

4.4 How an Origin Server Interprets the `Cookie` Header

A user agent returns much of the information in the `Set-Cookie2` header to the origin server when the `Path` attribute matches that of a new request. When it receives a `Cookie` header, the origin server should treat cookies with *NAME*s whose prefix is `$` specially, as an attribute for the cookie.

4.5 Caching Proxy Role

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:

- 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-Cookie2` 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-Cookie2` subject to the control of the usual header, `Cache-control: no-cache="set-cookie2"`. (The `Set-Cookie2` header should usually not be cached.)

Proxies must not introduce `Set-Cookie2` (`Cookie`) headers of their own in proxy responses (requests).

5. EXAMPLES

5.1 Example 1

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-Cookie2: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
   ```

   Cookie reflects user's identity.

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-Cookie2: 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-Cookie2: 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.

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

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-Cookie2: Part_Number="Rocket_Launcher_0001"; Version="1";
            Path="/acme"
```

and

```
Set-Cookie2: 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.

6. IMPLEMENTATION CONSIDERATIONS

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

6.1 `Set-Cookie2` Content

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-Cookie2` 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. Of course, using a database creates some problems that this state management specification was meant to avoid, namely:

1. keeping real state on the server side;
2. how and when to garbage-collect the database entry, in case the user agent terminates the session by, for example, exiting.

6.2 Stateless Pages

Caching benefits the scalability of WWW. Therefore it is important to reduce the number of documents that have state embedded in them inherently. For example, if a shopping-basket-style application always displays a user's current basket contents on each page, those pages cannot be cached, because each user's basket's contents would be different. On the other hand, if each page contains just a link that allows the user to ''Look at My Shopping Basket,'' the page can be cached.

6.3 Implementation Limits

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. 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:

- 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-Cookie2` 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-Cookie2` 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.

6.3.1 Denial of Service Attacks 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.

7. PRIVACY

7.1 User Agent Control

An origin server could create a `Set-Cookie2` 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.
- 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.
- 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-Cookie2` 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; the default should be ‘‘no.’’ If the user chooses to retain state information, it would be restored the next time the user agent runs.

NOTE: User agents should probably be cautious about using files to store cookies long-term. If a user runs more than one instance of the user agent, the cookies could be commingled or otherwise corrupted.

7.2 Protocol Design

The restrictions on the value of the `Domain` attribute, and the rules concerning unverifiable transactions, 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`. We consider it acceptable for hosts `host1.foo.com` and `host2.foo.com` to share cookies, but not `a.com` and `b.com`.

Similarly, a server can only set a `Path` for cookies that are related to the request-URI.

8. SECURITY CONSIDERATIONS

8.1 Clear Text

The information in the `Set-Cookie2` and `Cookie` headers is unprotected. Two consequences are:

1. Any sensitive information that is conveyed in them is exposed to intruders.
2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.

These facts imply that information of a personal and/or financial nature should only 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

Proper application design can avoid spoofing attacks from related domains. Consider:

1. User agent makes request to `victim.cracker.edu`, gets back cookie `session_id="1234"` and sets the default domain `victim.cracker.edu`.
2. User agent makes request to `spoof.cracker.edu`, gets back cookie `session-id="1111"`, with `Domain=".cracker.edu"`.
3. User agent makes request to `victim.cracker.edu` again, and passes

   ```
   Cookie:     $Version="1"; session_id="1234",
               $Version="1"; session_id="1111"; $Domain=".cracker.edu"
   ```

   The server at `victim.cracker.edu` should detect that the second cookie was not one it originated by noticing that the `Domain` attribute is not for itself and ignore it.

8.3 Unexpected Cookie Sharing

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 a CGI on host `b.com`. User agent implementors are strongly encouraged to prevent this sort of exchange whenever possible.

9. OTHER, SIMILAR, PROPOSALS

Apart from RFC 2109, three other proposals have been made to accomplish similar goals. This specification began as an amalgam of Kristol's State-Info proposal and Netscape's Cookie proposal.

Brian Behlendorf proposed a `Session-ID` header that would be user-agent-initiated and could be used by an origin server to track ‘‘clicktrails.’’ It would not carry any origin-server-defined state, however. Phillip Hallam-Baker has proposed another client-defined session ID mechanism for similar purposes.

While both session IDs and cookies can provide a way to sustain stateful sessions, their intended purpose is different, and, consequently, the privacy requirements for them are different. A user initiates session IDs to allow servers to track progress through them, or to distinguish multiple users on a shared machine. Cookies are server-initiated, so the cookie mechanism described here gives users control over something that would otherwise take place without the users' awareness. Furthermore, cookies convey rich, server-selected information, whereas session IDs comprise user-selected, simple information.

10. HISTORICAL

10.1 Compatibility with Existing Implementations

Existing cookie implementations, based on the Netscape specification, use the `Set-Cookie` (not `Set-Cookie2`) header. User agents that receive in the same response both a `Set-Cookie` and `Set-Cookie2` response header for the same cookie must discard the `Set-Cookie` information and use only the `Set-Cookie2` information. Furthermore, a user agent must assume, if it received a `Set-Cookie2` response header, that the sending server complies with this document and will understand `Cookie` request headers that also follow this specification.

New cookies must replace both equivalent old- and new-style cookies. That is, if a user agent that follows both this specification and Netscape's original specification receives a `Set-Cookie2` response header, and the *NAME* and the `Domain` and `Path` attributes match (per the **Cookie Management** section) a Netscape-style cookie, the Netscape-style cookie must be discarded, and the user agent must retain only the cookie adhering to this specification.

Older user agents that do not understand this specification, but that do understand Netscape's original specification, will not recognize the `Set-Cookie2` response header and will receive and send cookies according to the older specification.

A user agent that supports both this specification and Netscape-style cookies should send a `Cookie` request header that follows the older Netscape specification if it received the cookie in a `Set-Cookie` response header and not in a `Set-Cookie2` response header. However, it should send the following request header as well:

```
Cookie2: $Version="1"
```

The `Cookie2` header advises the server that the user agent understands new-style cookies. If the server understands new-style cookies, as well, it should continue the stateful session by sending a `Set-Cookie2` response header, rather than `Set-Cookie`. A server that does not understand new-style cookies will simply ignore the `Cookie2` request header.

10.2 Caching and HTTP/1.0

Some caches, such as those conforming to HTTP/1.0, will inevitably cache the `Set-Cookie2` and `Set-Cookie` headers, because there was no mechanism to suppress caching of headers prior to HTTP/1.1. This caching can lead to security problems. Documents transmitted by an origin server along with `Set-Cookie2` and `Set-Cookie` headers usually either will be uncachable, or will be ‘‘pre-expired.’’ As long as caches obey instructions not to cache documents (following `Expires:` <*a date in the past*> or `Pragma: no-cache` (HTTP/1.0), or `Cache-control: no-cache` (HTTP/1.1)) uncachable documents present no problem. However, pre-expired documents may be stored in caches. They require validation (a conditional GET) on each new request, but some cache operators loosen the rules for their caches, and sometimes serve expired documents without first validating them. This combination of factors can lead to cookies meant for one user later being sent to another user. The `Set-Cookie2` and `Set-Cookie` headers are stored in the cache, and, although the document is stale (expired), the cache returns the document in response to later requests, including cached headers.

11. ACKNOWLEDGEMENTS

This document really represents the collective efforts of the following people, in addition to the authors: Roy Fielding, Yaron Goland, Marc Hedlund, Ted Hardie, Koen Holtman, Shel Kaphan, Rohit Khare, Foteos Macrides, David W. Morris.

12. AUTHORS’ ADDRESSES

David M. Kristol
Bell Laboratories, Lucent Technologies
600 Mountain Ave. Room 2A-227
Murray Hill, NJ 07974

Phone: (908) 582-2250
FAX: (908) 582-1239
Email: `dmk@bell-labs.com`

Lou Montulli
Netscape Communications Corp.
501 E. Middlefield Rd.
Mountain View, CA 94043

Phone: (415) 528-2600
Email: `montulli@netscape.com`

Expires January 30, 1998