Network Working Group
Internet-Draft
Obsoletes: 2965 (if approved)
Intended status: Standards Track
Expires: September 15, 2011

Y. Pettersen
Opera Software ASA

March 14, 2011

# HTTP State Management Mechanism v2
## draft-pettersen-cookie-v2-06

## Abstract

This document specifies a way to create a stateful session with Hypertext Transfer Protocol (HTTP) requests and responses. It describes three HTTP headers, Cookie, Cookie2, and Set-Cookie2, which carry state information between participating origin servers and user agents. The method described here differs from both Netscape's Cookie proposal [Netscape], and [RFC2965], but it can, provided some requirements are met, interoperate with HTTP/1.1 user agents that use Netscape's method. (See the HISTORICAL section.)

This document defines new rules for how cookies can be shared between servers within a domain. These new rules are intended to address security and privacy concerns that are difficult to counter for clients implementing Netscape's proposed rules or the rules specified by RFC 2965.

This document reflects implementation experience with RFC 2965 and obsoletes it.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## Status of this Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 15, 2011.

Table of Contents

## 1. Introduction

HTTP cookies are widely used to maintain state across multiple HTTP requests in a wide variety of HTTP based applications, such as shopping carts for web shops, login credentials, preferences, identity information, etc. While alternatives exists, they are more cumbersome than HTTP cookies. The flexibility and ease of use of cookies may therefore have assisted the rapid spread of the World Wide Web.

Unfortunately, some of the flexibility of cookies, specifically how cookies are shared among multiple hosts, is causing security and privacy concerns.

[RFC2965] specifies that a cookie may be shared with any server within the Reach of the host, that is, the parent domain of the host, and the [Netscape] proposal allowed, within certain restrictions, even wider sharing to servers in the entire second- or third-level domain in which the request-host is part.

In some domain hierarchies, such as the generic Top Level Domain (TLD) dotCOM domain this will work well, but in many TLDs such as the Country-Code TLD (ccTLD) dotUK, this kind of sharing can cause problems, unless severely restricted, because it makes assumptions about what control and authorization has actually been granted the request-host. The dotUK TLD and many other ccTLDs have numerous subdomains that are treated as actual TLDs or registry like domains, similar to dotCOM, dotNet and dotORG, such as the co.uk, org.uk and ac.uk domains that are used to group otherwise unrelated domains into

categories based on their intended usage (e.g. commercial, non-commercial, governmental, academic).

Permitting cookies to be shared across such registry-like domains may result in undesirable datasharing, denial of service problems, even security related problems.

The original rules in Netscape's proposal, one internal dot in domain in the generic TLDs and two in domain name for non-generic TLDs, turned out not to be good enough, nor were they properly implemented in any client as they severely limited legitimate use of cookies; RFC2965's one level up rule restricted the problem somewhat, but not enough, as it was still possible to bypass the restrictions. Clients have implemented or proposed various heuristics to limit the impact of the problem, some by using a blacklist of second level domains that the client is not permitted to accept cookies for, others use DNS IP address lookups of the Set-Cookie header field's Domain attribute as a heuristic method to determine the appropriateness of permitting a cookie to be set, and a large database of domains that





should not be able to accept cookies have also been proposed.

Similarly, but less serious, the ability to set cookies to a parent path can, under some circumstances, cause interference between different applications in a given environment. In single application environments such sharing is not dangerous, but could be problematic when multiple independent administrators share the same service, such as in shared hosting environments where all users are hosted in their own path on the same server. In such environments a malicious user can set a cookie that is shared by many users, and since most version 0 [Netscape] implementations do not enforce a prefix path restriction it is also possible to limit the cookies to a path not controlled by the user, but not visible to all the other users on the host. Such cookies can interfere with the function of other applications on the host or within the domain.

This document presents an alternative method for reducing these problems by

- Removing the Domain attribute that permitted cookies to be specified for the parent domain, and instead introduces the SubDomain attribute that will permit servers to share cookies, but

only with servers whose name domain-matches the name of the request-host that set the cookie, and not parent domains.

- Removing the Path attribute, replacing it by the SubPath attribute that may be used to specify which resources under the request-path will be allowed to receive the cookie, instead of specifying which parent path is allowed to send the cookie.

This specification will not be able to accept cookies for hosts that are using domain specifications for parent domains as defined by the previous cookie specifications, but implementations using the older specification will be able to accept cookies from hosts following this specification.

This document also introduces new requirements for the contents of the Cookie header field, specifically that the $Domain and $Path attributes must always be sent, even when no domain or path has been specified, as this will allow request-hosts to verify the domain of the cookies even for cookies received from hosts using the older specifications.

## 2. Terminology

The terms user agent, client, server, proxy, origin server, and header field, and the tokens DIGIT, token, http_URL, and quoted-





string have the same meaning as in the HTTP/1.1 specification [RFC2616]. The terms abs_path and absoluteURI have the same meaning as in the URI Syntax specification [RFC3986].

The grammar uses the notation from the HTTP/1.1 specification [RFC2616] to describe the syntax of the HTTP header fields.

Host name (HN) means either the host domain name (HDN) or the numeric Internet Protocol (IP) address or IP-literal of a host, as defined by [RFC3986]. The fully qualified domain name is preferred; use of numeric IP addresses or IP-literals 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 effective host name is related to host name.  If a host name is a host domain name and contains no dots, the effective host name is that name with the string .local appended to it.  Otherwise the effective host name is the same as the host name.  Note that all effective host names contain at least one dot.

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, for HTTP URIs, and the HTTPS default, 443, for HTTPS URIs [RFC2818].  The request-port of a cookie is the request-port of the request in which a Set-Cookie2 response header field was returned to the user agent.

Host names can be specified either as an IP address, IP-literal or an HDN string.  Sometimes we compare one host name with another.  (Such comparisons SHALL be case-insensitive.)  Host A's name domain-matches host B's if

o  their host name strings string-compare equal; or

o  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:

o  If





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

   *  H has the form A.B; and

   *  A has no embedded (that is, interior) dots; and

   *  B has at least one embedded dot, or B is the string "local". then the reach of H is .B.

o  Otherwise, the reach of H is H.

For two strings that represent paths, P1 and P2, P1 path-matches P2 if P2 is a prefix of P1 (including the case where P1 and P2 string-compare equal). Thus, the string /tec/waldo path-matches /tec.

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. Description

We describe 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.

### 3.1. Syntax: General

The two state management header fields, Set-Cookie2 and Cookie, have common syntactic properties involving attribute-value pairs.

```
av-pairs    =     av-pair *(";" av-pair)
av-pair     =     attr ["=" value] ; optional value
attr        =     token
value       =     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 also that, unless prohibited, all values in the grammar below can be represented as quoted string, even if the grammar does not directly indicate it.

NOTE: The syntax above allows whitespace between the attribute name and the = sign.





### 3.2. Origin Server Role

### 3.2.1. General

The origin server initiates a session, if it so desires. To do so, it returns an extra HTTP response header to the client, Set-Cookie2, described below.

A user agent returns a Cookie request header field (see below) to the origin server if it chooses to continue the 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 field 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 field with Max-Age=0.

Servers MAY return Set-Cookie2 response header fields with any response. User agents SHOULD send Cookie request header fields, subject to other rules detailed below, with every request.

An origin server MAY include multiple Set-Cookie2 header fields in a response. Note that an intervening gateway could fold such multiple headers into a single header, as described by [RFC2616].

### 3.2.2. Set-Cookie2 Syntax

The syntax for the Set-Cookie2 response header field 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"
                |       "SubDomain"
                |       "Max-Age" "=" value
                |       "SubPath" "=" value
                |       "Port" [ "=" <"> portlist <"> ]
                |       "Unsecure"
                |       "Version" "=" 1*DIGIT
                |       "HttpOnly"
portlist        =       1#portnum
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, the client SHALL use only the value associated with the first appearance of the attribute; a client MUST ignore values after the first.

The NAME of a cookie MAY be the same as one of the attributes in this specification. However, because the cookie's NAME must come first in a Set-Cookie2 response header field, the NAME and its VALUE cannot be confused with an attribute-value pair.

NAME=VALUE REQUIRED. The name of the state information ("cookie") is NAME, and its value is VALUE. NAMEs that begin with $ are reserved 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 field.

Comment=value OPTIONAL. Because cookies can be used to derive or store private information about a user, the value of 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. Characters in value MUST be in UTF-8 encoding. [RFC3629]

CommentURL="http_URL" OPTIONAL. Because cookies can be used to derive or store private information about a user, the CommentURL attribute allows 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.

SubDomain OPTIONAL. The SubDomain attribute specifies that the user agent should share the cookie with any hosts that domain-matches the name of the host sending the cookie

Max-Age=value OPTIONAL. The value of the Max-Age attribute is delta-seconds, the lifetime of the cookie in seconds, a decimal non-negative integer. To handle cached cookies correctly, a client SHOULD calculate the age of the cookie according to the age calculation rules in the HTTP/1.1 specification [RFC2616]. When the age is greater than delta-seconds seconds, the client SHOULD discard the cookie. A value of zero means the cookie SHOULD be discarded immediately.

SubPath=value OPTIONAL. The value of the SubPath attribute specifies the subset of URLs within the default path on the origin server to which this cookie applies.

Port["portlist"] OPTIONAL. The Port attribute restricts the port to which a cookie may be returned in a Cookie request header field. Note that the syntax REQUIREs quotes around the OPTIONAL portlist even if there is only one portnum in portlist.

Unsecure OPTIONAL. The Unsecure attribute (with no value) is only used for cookies sent over secure connections and directs the user agent that the associated cookie can also be sent over an unsecure connection, not just to over (unspecified) secure connections, to the origin server whenever it sends back this cookie. The default for cookies sent over a secure connection is to protect the confidentiality and authenticity of the information in the cookie, so the client MUST NOT send a cookie over an unsecure connection if that cookie was received over a secure connection, but only send it over a connection at least as secure as the one it was received over unless the Unsecure flag was set for that cookie.

Version=value REQUIRED. The value of the Version attribute, a decimal integer, identifies the version of the state management specification to which the cookie conforms. For this specification, Version=2 applies.

HttpOnly Cookies with this flag set MUST NOT be provided to non-HTTP requestors, such as scripts. Additionally, non-HTTP updates that would overwrite such cookies, or that includes this flag, MUST be refused.

### 3.2.3. Controlling Caching

An origin server must be cognizant of the effect of possible caching of both the returned resource and the Set-Cookie2 header field. 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 field 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 field SHOULD NOT be cached. A Set-Cookie2 header field that is intended to be shared by multiple users MAY be cached.

The origin server SHOULD send the following additional HTTP/1.1 response header fields, depending on circumstances:

- To suppress caching of the Set-Cookie2 header field:

  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 header fields unless they know for certain (by out of band means) that there are no HTTP/1.0 proxies in the response chain. 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.





## 3.3. User Agent Role

### 3.3.1. Interpreting Set-Cookie2

The user agent keeps separate track of state information that arrives via Set-Cookie2 response header fields 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 or that contain invalid data, and if necessary ignore the entire header field. 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 effective request-host. (Note that because there is no dot at the beginning of effective request-host, the default Domain can only domain-match itself.)

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 and including the right-most /.

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

Unsecure  Only for cookies received over a secure connection: If absent, the user agent MUST NOT send the cookie over an unsecure channel.  (Cookies received over an unsecure connection can be sent to secure connections)

HttpOnly  The default behavior is that a cookie can be included in all listings of cookies for a given URL, also those requested by non-HTTP requestors, e.g scripts.

The user agent MUST ignore the SubDomain attribute if the effective request-host is an IP-address or IP-literal.

If the SubDomain attribute is present the state attribute Domain becomes .H where H is the effective request-host.

If the SubPath attribute is present the state attribute Path becomes Px where P is the default path, up to and including the right-most / and x is the value of the attribute.





### 3.3.2. Rejecting Cookies

To prevent possible security or privacy violations, a user agent rejects a cookie according to rules below.  The goal of the rules is to try to limit the set of servers for which a cookie is valid, based on the values of the Path, Domain, and Port attributes and the request-URI, request-host and request-port.

A user agent rejects (SHALL NOT store its information) if the Version attribute is missing, or contains a value of 1 or higher.  Moreover, a user agent rejects (SHALL NOT store its information) if any of the following is true of the attributes explicitly present in the Set-Cookie2 response header field:

- The value for the SubPath attribute appended to the default path is not a prefix of the request-URI.

- The Port attribute has a "port-list", and the request-port was not

in the list.

- The source of the cookie is non-HTTP, e.g. a script, that either include the HttpOnly attribute, or would overwrite a cookie with the HttpOnly attribute.

Examples:

- 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.

- A Set-Cookie2 from a path /example1/example1 for SubPath=exam will be accepted for the path /example1/exam

- A Set-Cookie2 from a path /example1/example1 for SubPath=exor will be rejected because exor is not a prefix of example1.

### 3.3.3. Cookie Management

If a user agent receives a Set-Cookie2 response header field whose NAME is the same as that of a cookie it has previously stored, the new cookie supersedes the old when: the old and new Domain attribute values compare equal, using a case-insensitive string-compare; and, the old and new Path attribute values string-compare equal (case-sensitive). 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 field 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 field 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 field, 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.

### 3.3.4. Sending Cookies to the Origin Server

When it sends a request to an origin server, the user agent includes a Cookie request header field if it has stored 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 field is:

```
cookie          =  "Cookie:" cookie-version 1*((";" | ",") cookie-value)
cookie-value    =  NAME "=" VALUE ";" cookie-path ";" cookie-domain
                   [";" cookie-port] [";" cookie-origin]

cookie-version  =  "$Version" "=" value
NAME            =  attr
VALUE           =  value
cookie-path     =  "$Path" "=" value
cookie-domain   =  "$Domain" "=" value
cookie-port     =  "$Port" [ "=" <"> port_list <"> ]
cookie-origin   =  "$Origin" "=" [ <"> http_URL <"> ]
```

cookie-version  The value of the cookie-version attribute MUST be the value from the Version attribute of the corresponding Set-Cookie2 response header field. Otherwise the value for cookie-version is 0.

name  This is the verbatim name from the original Set-Cookie or Set-Cookie2 header field setting the cookie item.

cookie-value  This is a verbatim copy of the value from the original Set-Cookie or Set-Cookie2 header field setting the cookie item. Quotes MUST be included if they were originally used, and MUST NOT be used if the original value did not contain them.

cookie-path  The value for the cookie-path attribute MUST be the value from the cookie's Path state attribute, as determined when the corresponding Set-Cookie2 response header field was parsed. If the cookie was set using a previous specification this value

MUST be the value of the Path attribute of the corresponding response header field or, if the Path attribute was absent, the default path of the URI used to set the cookie.

cookie-domain The value for the cookie-domain attribute MUST be the value from the cookie's Domain state attribute, as determined when the corresponding Set-Cookie2 response header field was parsed. If the response header field used the SubDomain attribute the domain value MUST be prefixed by a ".", if the Domain attribute is a default domain the domain value MUST NOT be prefixed by a ".". If the cookie was set by host supporting a previous version this value MUST be the Domain attribute from the correponding header field, including a preceding "." if the Domain attribute was present; if it was not present the domain value must be the name of the host setting the cookie, without being prefixed with a ".".

cookie-port The cookie-port attribute of the Cookie request header field MUST be exactly the value of the Port attribute, if one was present, in the corresponding Set-Cookie2 response header field. That is, the port attribute MUST be present if the Port attribute was present in the Set-Cookie2 header field, and it MUST have the same value (the exact character sequence), if any. Otherwise, if the Port attribute was absent from the Set-Cookie2 header field, the attribute likewise MUST be omitted from the Cookie request header field.

cookie-origin The cookie-origin attribute of the Cookie header field MUST be sent for all cookies that were set according to [Netscape] and [RFC2965], and the value MUST be at least the scheme, authority and default path portion (as defined for the cookie path attribute) of the URI that originally set the cookie. If the URI is not known then the URI part of this attribute MUST be empty. This attribute can be used by the server receiving the cookie to determine if it is willing to trust the cookie, based on which server (and path) originally set the cookie. The basic information of this attribute, except the scheme, is implied by the cookie-domain and cookie-path attributes when the cookie is set according to this specification, and the client SHOULD NOT send this attribute with cookies set according to this specification.

Note that there is neither a Comment nor a CommentURL attribute in the Cookie request header field corresponding to the ones in the Set-Cookie2 response header field. 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 a Cookie request header field from among all





the cookies it has received:

Domain Selection The origin server's effective host name MUST domain-match the Domain state attribute of the cookie.

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

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 request-URI MUST path-match the Path state attribute of the cookie.

Max-Age Selection Cookies that have expired should have been discarded and thus are not forwarded to an origin server.

HttpOnly Cookies with the HttpOnly attribute MUST NOT be returned to non-HTTP requestors, e.g. Javascript.

If multiple cookies satisfy the criteria above, they are ordered in the Cookie header field 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 field is semi-colon (;) everywhere. A server SHOULD also accept comma (",") as the separator between cookie-values for future compatibility.

A client MAY split a Cookie header field into multiple Cookie header fields, but SHOULD NOT do this unless comma is used as the separator,

and the receiving server is expected to handle a multi-header Cookie value.

### 3.3.5. Identifying What Version is Understood: Cookie2

The Cookie2 request header field facilitates interoperation between clients and servers that understand different versions of the cookie specification. When the client sends one or more cookies to an





origin server, if at least one of those cookies contains a $Version attribute whose value is different from the version that the client understands, then the client MUST also send a Cookie2 request header field, the syntax for which is

```
cookie2 =       "Cookie2:" cookie-version
```

Here the value for cookie-version is the highest version of cookie specification (currently 2) that the client understands. The client MUST only send at most one such request header field per request.

### 3.3.6. 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".

(NOTE: 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.

## 3.4. How an Origin Server Interprets the Cookie Header

A user agent returns much of the information in the Set-Cookie2 header field to the origin server when the request-URI path-matches the Path attribute of the cookie. When it receives a Cookie header field, the origin server SHOULD treat cookies with NAMEs whose prefix

is $ specially, as an attribute for the cookie.

## 3.5. 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 field in any request that the proxy must make of another server.
- Return the response to the client. Include any Set-Cookie2 response header field.
- Cache the received response subject to the control of the usual header fields, such as Expires,

  Cache-control: no-cache





  and

  Cache-control: private

- Cache the Set-Cookie2 subject to the control of the usual header field,

  Cache-control: no-cache="set-cookie2"
  (The Set-Cookie2 header field should usually not be cached.)

Proxies MUST NOT introduce Set-Cookie2 (Cookie) header fields of their own in proxy responses (requests).

# 4. Examples

## 4.1. Example 1

Most detail of request and response header fields has been omitted.

Assume the user agent has no stored cookies, and that the hostname is www.example.com

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="2";
```

Cookie reflects user's identity.

3. User Agent -> Server

```
POST /acme/pickitem HTTP/1.1
Cookie: $Version="2"; Customer="WILE_E_COYOTE";
        $Domain="www.example.com"; $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="2"
```

Shopping basket contains an item.

5. User Agent -> Server

```
POST /acme/shipping HTTP/1.1
Cookie: $Version="2";
        Customer="WILE_E_COYOTE";
                 $Domain="www.example.com"; $Path="/acme/";
        Part_Number="Rocket_Launcher_0001";
                 $Domain="www.example.com"; $Path="/acme/"
```

```
    [form data]
```

User selects shipping method from form.

6. Server -> User Agent

```
    HTTP/1.1 200 OK
    Set-Cookie2: Shipping="FedEx"; Version="2"
```

New cookie reflects shipping method.

7. User Agent -> Server

```
    POST /acme/process HTTP/1.1
    Cookie: $Version="2";
            Customer="WILE_E_COYOTE";
                     $Domain="www.example.com"; $Path="/acme/";
            Part_Number="Rocket_Launcher_0001";
                     $Domain="www.example.com"; $Path="/acme/";
            Shipping="FedEx";
                     $Domain="www.example.com"; $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-URIs all path-match /acme/, the Path attribute of each cookie, each request contains all the cookies received so far.





## 4.2. Example 2

This example illustrates the effect of the Path attribute. All detail of request and response header fields has been omitted. Assume the user agent has no stored cookies.

Imagine the user agent has received, in response to earlier requests, the response header fields

```
Set-Cookie2: Part_Number="Rocket_Launcher_0001"; Version="2"
```

and

```
Set-Cookie2: Part_Number="Riding_Rocket_0023"; Version="2";
        SubPath="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 field:

```
Cookie: $Version="2";
        Part_Number="Riding_Rocket_0023";
                    $Domain="www.example.com"; $Path="/acme/ammo";
        Part_Number="Rocket_Launcher_0001";
                    $Domain="www.example.com"; $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 field:

```
Cookie: $Version="2"; Part_Number="Rocket_Launcher_0001";
                      $Domain="www.example.com"; $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.

## 5. Implementation Considerations

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

## 5.1. 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 field 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.

## 5.2. 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.

## 5.3. 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 characters that comprise the cookie non-terminal in the syntax description of the Set-Cookie2 header field, and as received in the Set-Cookie2 header field)

- 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 field 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.

### 5.3.1. 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.

## 5.4. Backwards Compatibility

Servers that send cookies according to this specification and that wish to send cookies with the same properties to a client following the RFC2965 specification MAY send Domain and Path attributes in the same header field as the version 2 arguments. Clients following this specification MUST ignore these attributes.

# 6. Privacy

Informed consent should guide the design of systems that use cookies. A user should be able to find out how a web site plans to use information in a cookie and should be able to choose whether or not those policies are acceptable. Both the user agent and the origin server must assist informed consent.

## 6.1. User Agent Control

An origin server could create a Set-Cookie2 header field 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, for example, 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, for example, by mechanisms

- to notify the user when the user agent is about to send a cookie to the origin server, to offer 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 and delete 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 a Cookie header field to an origin server, 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 header fields.)

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.

## 6.2. Origin Server Role

An origin server SHOULD promote informed consent by adding CommentURL or Comment information to the cookies it sends. CommentURL is





preferred because of the opportunity to provide richer information in a multiplicity of languages.

## 6.3. Clear Text

The information transmitted in the Set-Cookie2 and Cookie header fields is unprotected. As a consequence:

1. Any sensitive information that is conveyed in them is exposed to intruders.

2. A malicious intermediary could alter the header fields 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 field is a database key, an origin server should be vigilant to prevent a bad Cookie value from causing failures.

A user agent in a shared user environment poses a further risk. Using a cookie inspection interface, User B could examine the contents of cookies that were saved when User A used the machine.

# 7. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

# 8. Security Considerations

## 8.1. Protocol Design

The restrictions on the value of the Domain state attribute by using the SubDomain 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 host, or a closely related set of hosts. We consider it acceptable for hosts host1.foo.com and host2.foo.com to share cookies, but not a.com and b.com. Because of the many hierarchies used to organize domain names, it is not possible to define a small set of rules that can tell the client if a domain name is in fact similar to .com, or not. For that reason, this specification introduces the "Subdomain" attribute as a replacement of the "Domain" attribute defined by





[RFC2965], so that only hosts in the domain below the server top.example.com can receive the cookie.

Similarly, a server can only share cookies with resource in subfolders of the default path derived from the request-URI.

## 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 cracker.edu, gets back cookie session-id="1111", with a SubDomain attribute.

3. User agent makes request to victim.cracker.edu again, and passes

```
Cookie: $Version="1"; session_id="1234";
                $Domain="victim.cracker.edu"; $Path="/example/" ,
```

```
        $Version="1"; session_id="1111";
                $Domain=".cracker.edu"; $Path="/"
```

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.

### 8.4. Cookies for Account Information

While it is common practice to use them this way, cookies are not designed or intended to be used to hold authentication information, such as account names and passwords. Unless such cookies are exchanged over an encrypted path, the account information they contain is highly vulnerable to perusal and theft.





## 9. Historical

### 9.1. Compatibility with Existing Implementations

Existing cookie implementations, based on the Netscape specification, use the Set-Cookie (not Set-Cookie2) header field. User agents that receive in the same response both a Set-Cookie and a Set-Cookie2 response header field 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 field, that the sending server complies with this document and will understand Cookie request header fields 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 field, and the NAME and the Domain and Path state 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 field and will receive and send cookies according to the older specification.

A user agent that supports both this specification and Netscape-style cookies SHOULD still send a Cookie request header field that follows the format specified in this document, as the benefit of adding domain and path information to each cookie and thus providing even older server with the ability to detect incorrectly set cookies outweigh the potential problems unknown cookienames may cause.

The client should also send this header field in requests to servers that receive cookies that are not of the version specified by this document

```
   Cookie2: $Version="2"
```

The Cookie2 header field 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 field, rather than Set-Cookie. A server that does not understand new-style cookies will simply ignore the Cookie2 request header field.





## 9.2. Caching and HTTP/1.0

Some caches, such as those conforming to HTTP/1.0, will inevitably cache the Set-Cookie2 and Set-Cookie header fields, 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 header fields

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 header fields are stored in the cache, and, although the document is stale (expired), the cache returns the document in response to later requests, including cached header fields.

## 10. Acknowledgements

This document is based on [RFC2965] by David Kristol and Lou Montulli and the collective efforts of the HTTP Working Group of the IETF and, particularly, the following people, in addition to the authors of RFC 2965: Roy Fielding, Yaron Goland, Marc Hedlund, Ted Hardie, Koen Holtman, Shel Kaphan, Rohit Khare, Foteos Macrides, David W. Morris.

This document include some changes suggested by RFC 2965 errata drafts posted by Arne Thomassen [ERRATA1] and David Kristol [ERRATA2].

## 11. References

### 11.1. Normative References

[I-D.ietf-httpstate-cookie]
Barth, A., "HTTP State Management Mechanism", draft-ietf-httpstate-cookie-23 (work in progress), March 2011.

[Netscape]
"Persistent Client State -- HTTP Cookies", <http://www.netscape.com/newsref/std/cookie_spec.html>.

available at
<http://www.netscape.com/newsref/std/cookie_spec.html>

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

### 11.2. Non-normative References

[ERRATA1] "RFC Errata: HTTP State Management Mechanism (draft)", October 2000, <http://retawq.sourceforge.net/cookies/cookie-errata.html>.

[ERRATA2] "[Draft of] Errata to RFC 2965", May 2003, <http://kristol.org/cookie/errata.html>.

## Appendix A. Open issues

- Should cookies with cookie-values with unquoted whitespace be rejected?

Author's Address

Yngve N Pettersen
Opera Software ASA
Waldemar Thranes gate 98
N-0175 OSLO,
Norway

Email: yngve@opera.com