HTTP Working Group
INTERNET DRAFT

David M. Kristol
Bell Laboratories
Lou Montulli
Netscape Communications

<draft-kristol-http-state-mgmt-01.ps>
April 15, 1996

Expires October 15, 1996

Proposed HTTP State Management Mechanism

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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.''

To learn the current status of any Internet-Draft, please check the ''1id-abstracts.txt'' listing contained in the Internet- Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

This is authors' draft 2.12. Change bars reflect changes since the previous Internet Draft, authors' draft 2.10.

1. ABSTRACT

HTTP, the protocol that underpins the World-Wide Web (WWW), is stateless. That is, each request stands on its own; origin servers don't need to remember what happened with previous requests to service a new one. Statelessness is a mixed blessing, because there are potential WWW applications, like ''shopping baskets'' and library browsing, for which the history of a user's actions is useful or essential.

This proposal outlines a way to introduce state into HTTP. New request and response headers, `Cookie` and `Set-Cookie`, carry the state back and forth, thus relieving the origin server from needing to keep an extensive per-user or per-connection database. The changes required to user agents, origin servers, and proxy servers to support state management are modest.

This proposal, while very similar to Netscape's original state management (''Cookie'') proposal, differs slightly and incompatibly. It assumes that origin servers, proxies, and user agents use the HTTP/1.1 protocol. However, it can interoperate with HTTP/1.0 agents. (See the **HISTORICAL** section.)

2. TERMINOLOGY

The terms *user agent*, *client*, *server*, *proxy*, and *origin server* have the same meaning as in the HTTP/1.0 specification.

Fully-qualified host name (FQHN) means either the numeric IP address of a host, or its full Internet domain name, resolved to a top-level domain such as `.com` or `.uk`.

The terms *request-host* and *request-URI* refer to the values the client would send to the server in, respectively, the `Host` header and the Request-URI portion of the HTTP request line. Note that request-host is a FQHN.

Hosts can be specified either as an IP address or a FQHN string. Sometimes we compare one host with another. Host A *domain-matches* host B if

- both hosts are IP addresses and their host strings match exactly; or
- both hosts are FQHN strings and their host strings match exactly; or
- both hosts are FQHN strings and host A's string has the form N.B, where B is host B's FQHN string, and N is a name string that may have embedded dots, but not leading or trailing dots.

Note that domain-match is not a commutative operation: `a.b.c.com` domain-matches `c.com`, but not the reverse.

Because it was used in Netscape's original implementation of state management, we will use the term *cookie* to refer to the state information that passes between an origin server and user agent, and that gets stored by the user agent.

3. STATE AND SESSIONS

This proposal outlines how to introduce state into HTTP, the protocol that underpins the World-Wide Web (WWW). At present, HTTP is stateless: a WWW origin server obtains everything it needs to know about a request from the request itself. After it processes the request, the origin server can ''forget'' the transaction.

What do we mean by ''state?'' ''State'' implies some relation between one request to an origin server and previous ones made by the same user agent to the same origin server. If the sequence of these requests is considered as a whole, they can be thought of as a ''session.''

Koen Holtman identified these dimensions for the ''solution space'' of stateful dialogs:

- simplicity of implementation
- simplicity of use
- time of general availability when standardized
- downward compatibility
- reliability
- amount of privacy protection
- maximum complexity of stateful dialogs supported
- amount of cache control possible
- risks when used with non-conforming caches

The paradigm we have in mind obtains the same effect as if a user agent connected to an origin server, carried out many transactions at the user's direction, then disconnected. Two example applications we have in mind are a ''shopping cart,'' where the state information comprises what the user has bought, and a magazine browsing system, where the state information comprises the set of journals and articles the user has looked at already. Note some of the key points in the session paradigm:

1. The session has a beginning and an end.
2. The session is relatively short-lived.
3. Either the user agent or the origin server may terminate a session. There is no defined mechanism to inform the origin server that the user agent has terminated a session.
4. State is a property of the session. The user agent can receive, store, and send cookies that reflect this state.

4. PROPOSAL OUTLINE

The proposal we outline here defines 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 of the proposal 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-Cookie` and `Cookie`, have common syntactic properties involving attribute-value pairs. The following uses the notation and tokens `ALPHA` (lower and upper case letters), `DIGIT` (decimal digits), and `word` (informally, a `"`-quoted string or sequence of non-special, non-white space characters) from the HTTP/1.1 specification to describe their syntax.

```
av-pairs    =    av-pair *(";" av-pair)
av-pair     =    attr 0,1*("=" value)        ; optional value
attr        =    ALPHA *(ALPHA | DIGIT)
value       =    word
```

Attributes (names) (`attr`) are case-insensitive. White space is permitted between tokens. Note that while the above syntax description shows `value` as optional, most `attr`s require them.

NOTE: The syntax above allows space 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 is a logical connection, not a physical one. Don’t confuse these logical sessions with various ‘‘keepalive’’ proposals for physical sessions.) To initiate a session, the origin server returns an extra response header to the client, `Set-Cookie.` (The details follow later.)

A user agent returns a `Cookie` request header (see below) to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send back to the client a `Set-Cookie` response header with the same or different information, or it may send no `Set-Cookie` header at all. The origin server effectively ends a session by sending back a `Set-Cookie` header that has an immediate expiration time.

An origin server may include multiple `Set-Cookie` headers in a response. Note that an intervening gateway could fold multiple such headers into a single header.

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

```
set-cookie    =    "Set-Cookie:" cookies
cookies       =    cookie *("," cookie)
cookie        =    NAME "=" VALUE *(";" cookie-av)
NAME          =    attr
VALUE         =    value
cookie-av     =    "Domain" "=" value
              |    "Max-Age" "=" value
              |    "Path" "=" value
              |    "Secure"
              |    "Version" "=" value
```

Informally, the `Set-Cookie` response header comprises the token `Set-Cookie:`, followed by a comma-separated list of one or more cookies. Each cookie begins with a *NAME=VALUE* pair, followed by zero or more semi-colon-separated attribute-value pairs. The 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 behavior is undefined.

NAME=VALUE

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

The *VALUE* is opaque and may be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. ‘‘Opaque’’ implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone that examines the `Set-Cookie` header.

`Domain=`*domain*
Optional. The `Domain` attribute specifies the host and domain name for which the cookie is valid.

`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 to which this cookie applies.

`Secure`
Optional. The `Secure` attribute (with no value) directs the user agent only to use (unspecified) secure means to contact the origin server whenever it sends back this 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.

`Version=`*version*
Required. The `Version` attribute 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-Cookie` header. Caching ‘‘public’’ documents is desirable. For example, if the origin server wants to use a public document such as a ‘‘front door’’ page as a sentinel to indicate the beginning of a session for which a `Set-Cookie` response header must be generated, it needs to 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.

If the cookie is intended for use by a single user, the `Set-cookie` header should not be cached. A cookie header that is intended to be shared by multiple users may be cached.

The origin server should send the following additional response headers, depending on circumstances.

- `Cache-control: no-cache="set-cookie"`, to suppress caching the `Set-Cookie` header

and one of the following:

- `Cache-control: Max-Age=0` or `Cache-control: Expires=`*<yesterday>*, to ‘‘pre-expire’’ a cache entry. The resource may be cached, but the cache must validate it before returning it to the client.
- `Cache-control: no-cache`, to suppress caching of a private document.

4.3 User Agent Role

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

`Version` Defaults to ‘‘old cookie’’ behavior as originally specified by Netscape. See the **HISTORICAL** section.

`Domain` Defaults to the 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-Cookie` response, up to, but not including, the right-most `/`.

`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:

- The value for the `Path` attribute is not a prefix of the request-URI.
- The value for the `Domain` attribute contains no dots, or begins with a dot.
- The value for the `Domain` attribute ends with a dot and contains no other dots.
- The value for the request-host does not domain-match the `Domain` attribute.
- The request-host is a name (not IP address), D is the value of the `Domain` attribute, and request-host has the form H.D, where H is a string, and H contains one or more dots.

  (Examples: a `Set-Cookie` from request-host `y.x.foo.com` for `Domain=foo.com` would be rejected, because H is `y.x` and contains a dot, but a `Set-Cookie` from request-host `x.host.com` would be accepted.)

4.3.3 Cookie Management If a user agent receives a `Set-Cookie` response header whose *NAME=VALUE*, `Domain`, and `Path` attribute values exactly match those of a pre-existing cookie, the new one supersedes the old. However, if the `Set-Cookie` has a value for `Max-Age` of zero, the (old and new) cookie is discarded. Otherwise cookies accumulate until they expire (resources permitting), at which time they are discarded.

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.

User agents should allow the user control over cookie destruction. An infrequently-used cookie may function as a ‘‘preferences file’’ for network applications, and a user may wish to keep it even if it is the least-recently-used cookie. One possible implementation would be an interface that allows the permanent storage of a cookie through a checkbox (or, conversely, its immediate destruction).

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;
- 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]
cookie-version  =       "Cookie-Version" "=" <version from Set-Cookie, else "0">
NAME            =       attr
VALUE           =       value
path            =       "Cookie-Path" "=" <path from Set-Cookie>
domain          =       "Cookie-Domain" "=" <domain from Set-Cookie>
```

The user agent should send path or domain only if the corresponding value was provided explicitly in the `Set-Cookie` response header.

These rules apply to choosing applicable cookie-values from among all the cookies the user agent has.

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

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 are thus 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 A transaction is *verifiable* if the user 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. User agents typically use unverifiable transactions when they automatically get inlined or embedded entities or when they resolve redirection (3xx) responses. Typically the *origin transaction*, the transaction that the user initiates, is verifiable, and that transaction directly or indirectly causes the user agent to make unverifiable transactions.

The following rule eases the implementation of user agent mechanisms that give the user control over sessions (see **PRIVACY** section) by restricting the number of cases where control is needed. The rule prevents malicious service authors 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.

When it makes an unverifiable transaction, a user agent must only enable the cookie functionality, that is the sending of `Cookie` request headers and the processing of any `Set-Cookie` response headers, if a cookie with a domain attribute D was sent or received in its origin transaction, such that the host name in the Request-URI of the unverifiable transaction domain-matches D.

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.''

NB: Many current user agents already provide an acceptable review option that would render many links verifiable.

- When the mouse is over a link, the user agents display the link that would be followed.
- By letting a user view source, or save and examine source, the user agents let a user examine the link that would be followed when a form's submit button is selected.
- When automatic image loading is disabled, users can review the links that correspond to images by using one of the above techniques. However, if automatic image loading is enabled, those links would be unverifiable.

4.4 How an Origin Server Interprets `Cookie` |

A user agent returns much of the information in the `Set-Cookie` 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 `Cookie-` specially, as an attribute for the adjacent cookie. The value for such a *NAME* is to be interpreted as applying to the lexically (left-to-right) most recent cookie whose name does not have the `Cookie-` prefix. If there is no previous cookie, the value applies to the cookie mechanism as a whole. For example, consider the cookie |

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

`Cookie-Version` applies to the cookie mechanism as a whole (and gives the version number for the cookie mechanism). `Cookie-Path` is an attribute whose value (`/acme`) defines the `Path` attribute that was used when the `Customer` cookie was defined in a `Set-Cookie` response header.

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-Cookie` response header.
- Cache the received response subject to the control of the usual headers, such as `Expires`, `Cache-control: no-cache`, and `Cache-control: private`,
- Cache the `Set-Cookie` subject to the control of the usual header, `Cache-control: no-cache="set-cookie"`. (Usually the `Set-Cookie` header should not be cached.)

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/1.0
   [form data]
   ```

   User identifies self.

2. Server → User Agent

   ```
   HTTP/1.0 200 OK
   Set-Cookie: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
   ```

   Cookie reflects user's identity.

3. User Agent → Server

   ```
   POST /acme/pickitem HTTP/1.0
   Cookie: Cookie-Version="1"; Customer="WILE_E_COYOTE";
               Cookie-Path="/acme"
   [form data]
   ```

   User selects an item for ''shopping basket.''

4. Server → User Agent

   ```
   HTTP/1.0 200 OK
   Set-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1";
               Path="/acme"
   ```

   Shopping basket contains an item.

5. User Agent → Server

```
POST /acme/shipping HTTP/1.0
Cookie: Cookie-Version="1";                                              |
            Customer="WILE_E_COYOTE"; Cookie-Path="/acme";               |
            Part_Number="Rocket_Launcher_0001"; Cookie-Path="/acme"       |
```

User selects shipping method from form.

6. Server → User Agent

```
HTTP/1.0 200 OK
Set-Cookie: Shipping="FedEx"; Version="1"; Path="/acme"                  |
```

New cookie reflects shipping method.

7. User Agent → Server

```
POST /acme/process HTTP/1.0
Cookie: Cookie-Version="1";                                              |
            Customer="WILE_E_COYOTE"; Cookie-Path="/acme";               |
            Part_Number="Rocket_Launcher_0001"; Cookie-Path="/acme";      |
            Shipping="FedEx"; Cookie-Path="/acme"                        |
[form data]
```

User chooses to process order.

8. Server → User Agent

```
HTTP/1.0 200 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-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1"; Path="/acme"|
```

and

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

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

```
Cookie: Cookie-Version="1";                                              |
            Part_Number="Riding_Rocket_0023"; Cookie-Path="/acme/ammo";   |
            Part_Number="Rocket_Launcher_0001"; Cookie-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 URLs of the form `/acme/...` (assuming ‘‘`...`’’ did not have the prefix `ammo`) would include the following request header:

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

Here, the second cookie's `Path` attribute `/acme/ammo` is not a prefix of the request URL, 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-Cookie` 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-Cookie` header can incorporate information about the application areas by setting the `Path` attribute for each one.

The session information can obviously be clear or encoded text that describes state. However, if it grows too large, it can become unwieldy. Therefore, an implementor might choose for the session information to be a key into a server-side database. Of course, using a database creates some problems that the state management proposal 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 displayed a user's current basket contents on each page, those pages could not be cached, because each user's basket's contents would be different. On the other hand, if each page contained just a link that allowed the user to ''Look at My Shopping Basket,'' the page could 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. Furthermore, they should provide the following **minimum** capabilities:

- at least 300 cookies
- at least 4096 bytes per cookie (as measured by the size of the characters that comprise the `cookie` non-terminal in the syntax description of the `Set-Cookie` header)
- at least 20 cookies per unique host or domain name

The information in a `Set-Cookie` response header must be retained in its entirety. If for some reason there is inadequate space to store the cookie, it must be discarded, not truncated.

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 must still be supported.

7. PRIVACY

7.1 User Agent Control

An origin server could create a `Set-Cookie` header to track the path of a user through the server. Users may object to this behavior as an intrusive accumulation of information, even if their identity is not evident. (Identity might become evident if a user subsequently fills out a form that contains identifying information.)

The state management proposal 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 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-Cookie` response headers.)

When the user agent terminates execution, it should let the user discard all state information. Alternatively, the user agent may ask the user whether state information should be retained; the default should be ''no.'' Retained state information would then 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 messed up.

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. So 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-Cookie` 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

As defined here, cookies are vulnerable to spoofing attacks from related domains. Consider:

1. User agent makes request to `victim.hacker.edu`, gets back cookie `session_id=1234` and sets the default domain `victim.hacker.edu`.

2. User agent makes request to `spoof.hacker.edu`, gets back cookie `session-id=1111`, with `Domain=hacker.edu`.
3. User agent makes request to `victim.hacker.edu` again, and passes `Cookie: session_id=1234; session_id=1111`. The server at `victim.hacker.edu` may have trouble detecting that the second cookie was not one it originated.

This threat is eliminated by including `Cookie-Domain=` and `Cookie-Path=` attributes to accompany each cookie in the `Cookie` request header, so that the receiving origin server can tell if some cookies have been set in a domain containing servers it cannot trust. *

8.3 Unexpected Cookie Sharing

A user agent should make every attempt to prevent the sharing of session information between hosts that do not domain-match. 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

Three other proposals have been made to accomplish similar goals. This proposal is 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 ''clickstreams.'' It would not carry any origin-server-defined state, however. Phillip Hallam-Baker has proposed another client-defined session ID mechanism for similar purposes.

10. HISTORICAL

10.1 Compatibility With Netscape's Implementation

HTTP/1.0 clients and servers may use `Set-Cookie` and `Cookie` headers that reflect Netscape's original cookie proposal. These notes cover inter-operation between ''old'' and ''new'' cookies.

10.1.1 Extended `Cookie` *Header* This proposal adds attribute-value pairs to the `Cookie` request header in a compatible way. An ''old'' client that receives a ''new'' cookie will ignore attributes it does not understand; it returns what it does understand to the origin server. A ''new'' client always sends cookies in the new form. An ''old'' server that receives a ''new'' cookie will see what it thinks are many cookies with names that begin `Cookie-`, and it will ignore them. (The ''old'' server expects these cookies to be separated by semi-colon, not comma.)

10.1.2 Expires and Max-Age Netscape's original proposal defined an `Expires` header that took a date value in a fixed-length variant format in place of `Max-Age`.

10.1.3 Punctuation In Netscape's original proposal, the values in attribute-value pairs did not accept `"`-quoted strings. Origin servers should be cautious about sending values that require quotes unless they know the receiving user agent understand them (*i.e.*, ''new'' cookies). A (''new'') user agent should only use quotes around values in `Cookie` headers when the cookie's version(s) is (are) all compliant with this specification or later.

In Netscape's original proposal, no whitespace was permitted around the `=` that separates attribute-value pairs. Therefore such whitespace should be used with caution in new implementations.

10.1.4 Rejecting Cookies ''Old'' cookies were permitted to have `Domain` attributes that begin with a dot; ''new'' ones are not. To permit interoperability, user agents should implement the ''old'' rules for `Domain` matching for ''old'' cookies.

10.2 Caching and HTTP/1.0

Some caches, such as those conforming to HTTP/1.0, will inevitably cache the `Set-Cookie` header, 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-Cookie` headers will usually either be uncachable, or will be ‘‘pre-expired.’’ As long as caches obey instructions not to cache documents (following `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-Cookie` header is 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, Marc Hedlund, Koen Holtman, Shel Kaphan, Rohit Khare.

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) 582-5809
Email: `dmk@research.att.com`

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

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

Expires October 15, 1996