None (yet)	A. Barth	
Internet-Draft	U.C. Berkeley	
Expires: May 29, 2010	November 25, 2009	

TOC

HTTP State Management Mechanism
draft-abarth-cookie-05

Abstract

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

> NOTE: Much of the text herein is completely wrong. If you have suggestions for improving the draft, please send email to http-state@ietf.org. Suggestions with test cases are especially appreciated.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

TOC

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

2. Terminology

TOC

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

TOC

We outline here a way for an origin server to send state information to the user agent, and for the user agent to return the state information to the origin server.
The origin server initiates a session, if it so desires, by including a Set-Cookie header in an HTTP response. (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).
A user agent returns a Cookie request header (see below) to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It

may send the client a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all. Servers may return a Set-Cookie response headers with any response. User agents should send Cookie request headers, subject to other rules detailed below, with every request.
An origin server may include multiple Set-Cookie headers in a response. Note that an intervening gateway MUST NOT fold multiple Set-Cookie headers into a single header.
[TODO: Overview the Set-Cookie and Cookie headers.]

3.1. Examples

TOC

[TODO: Put some examples here.

---

4. Protocol Description

TOC

The cookie protocol consists of two HTTP headers: the Set-Cookie header and the Cookie header. The server sends the Set-Cookie header is to the user agent in an HTTP response, causing the user agent to modify the Cookie header it returns to the server.
This section describes the syntax and semantics of the protocol. Detailed conformance requirements for user agents are given in Section [TODO].

4.1. Set-Cookie

TOC

4.1.1. Syntax

TOC

Informally, the Set-Cookie response header comprises the token Set-Cookie:, followed by a cookie. Each cookie begins with a name-value-pair, followed by zero or more semi-colon-separated attribute-value pairs.
[TODO: Consider replacing this grammar with the one from 2009-11-07-Yui-Naruse.txt.]

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

The valid character for the value production vary depending on the attribute name.
[TODO: Investigate what token actually means.]
Attributes names are case-insensitive. White space is permitted between tokens. Servers MUST NOT include two attributes with the same name.
Note that although the above syntax description shows value as optional, some attributes require values.
The cookie-value is opaque to the user agent and MAY be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. "Opaque" implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone who examines the Set-Cookie header.
NOTE: The syntax above allows whitespace between the attribute and the U+3D ("=") character. Servers wishing to interoperate with some legacy user agents might wish to elide this extra white space to maximize compatibility.

4.1.2. Semantics

TOC

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

4.1.2.1. Max-Age

TOC

[TODO: Consider removing Max-Age from the server conformance section because it's not supported by IE.]

> **Syntax** A sequence of ASCII numerals.
>
> **Semantics** The value of the Max-Age attribute represents the maximum lifetime of the cookie, measured in seconds from the moment the

> user agent receives the cookie. If the server does not supply an Expires or a Max-Age attribute, the lifetime of the cookie is limited to the current session (as defined by the user agent).

4.1.2.2. Expires

TOC

> **Syntax** An RFC 1123 date [cite]. (Note that user agents use very forgiving date parers; see Section [TODO]).
>
> **Semantics** The value of the Expires attribute represents the maximum lifetime of the cookie, represented as the point in time at which the cookie expires. If the server does not supply an Expires or a Max-Age attribute, the lifetime of the cookie is limited to the current session (as defined by the user agent).

4.1.2.3. Domain

TOC

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

4.1.2.4. Path

TOC

> **Syntax** A sequence of characters beginning with a "/" character.
>
> **Semantics** The Path attribute specifies the scope of the cookie within a given FQDN. The user agent will include a cookie in an HTTP request only if the Request-URI's path matches, or is a subdirectory of, the cookie's Path attribute (where the "/" character is interpreted as a directory separator). The default value for the Path attribute is the directory of the Request-URI when the cookie was received.

4.1.2.5. Secure

TOC

Syntax
Servers MUST NOT include a value.

Semantics The user agent SHOULD protect the confidentiality of cookies with the Secure attribute by not transmitting Secure cookies over an "insecure" channel (where "insecure" is defined by the user agent).

4.1.2.6. HttpOnly

TOC

Syntax Servers MUST NOT include a value.

Semantics The user agent SHOULD protect confidentiality of cookies with the HttpOnly attribute by not revealing their contents via "non-HTTP" APIs. (Note that this document does not define which APIs are "non-HTTP".)

4.2. Cookie

TOC

4.2.1. Syntax

TOC

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

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

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

4.2.2. Semantics

TOC

Each name-value-pair represents a cookie stored by the user agent. The cookie name is returned in as the name and the cookie value is returned as the value.
The meaning of the cookies in the Cookie header is not defined by this document. Servers are expected to imbue these cookies with server-specific semantics.

4.3. Controlling Caching

TOC

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

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

and one of the following:

- To suppress caching of a private document in shared caches: Cache-Control: private.
- To allow caching of a document and require that it be validated before returning it to the client: Cache-Control: must-revalidate.
- To allow caching of a document, but to require that proxy caches (not user agent caches) validate it before returning it to the client: Cache-Control: proxy-revalidate.

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

HTTP/1.1 servers must send Expires: old-date (where old-date is a date long in the past) on responses containing Set-Cookie response headers unless they know for certain (by out of band means) that there are no downstream 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.

---

5. User Agent Conformance

TOC

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

5.1. Parsing the Set-Cookie Header

TOC

Let an LWS character be either a U+20 (SPACE) or a U+09 (TAB) character.
When a user agent receives an Set-Cookie header in an HTTP response, the user agent *receives a set-cookie-string* consisting of the value of the header.
A user agent MUST use the following algorithm to parse set-cookie-strings:

1. [TODO: Deal with "," characters. My current thinking is that we don't actually have to do anything special for them.]

2. If the header contains a U+3B (";") character:

   > the name-value-pair string is characters up to, but not including, the first U+3B (";"), and the unparsed-cookie-attributes are the remainder of the header (including the U+3B (";") in question).

   Otherwise:

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

3. If the name-value-pair string contains a U+3D ("=") character:

   > the (possibly empty) name string is the characters up to, but not including, the first U+3D ("=") character, and the (possibly empty) value string is the characters after the first U+3D ("=") character.

   Otherwise:

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

4. Remove any leading or trailing space from the name string and the value string.

5. The cookie-name is the name string, and the cookie-value is the value string.

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

1. If the unparsed-attributes string is empty, skip the rest of these steps.

2. Consume the first character of the unparsed-attributes (which will be a U+3B (";") character).

3. If the remaining unparsed-attributes contains a U+3B (";") character:

   > Consume the characters of the unparsed-attributes up to, but not including, the first U+3B (";") character.

   Otherwise:

   > Consume the remainder of the unparsed-attributes.

   The characters consumed in this step comprise the attribute-value-pair string.

4. If the attribute-value-pair string contains a U+3D ("=") character:

   > the (possibly empty) name string is the characters up to, but not including, the first U+3D ("=") character, and the

> (possibly empty) value string is the characters after the first U+3D ("=") character .

Otherwise:

> the name string is the entire attribute-value-pair string, and the value string is empty. (Note that this step differs from the analogous step when parsing the name-value-pair string.)

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

6. If the name is a ASCII case-insensitive match for an entry in the following table, process the value string as instructed.

Attribute	Instruction
Max-Age	See Section [TODO]
Expires	See Section [TODO]
Domain	See Section [TODO]
Path	See Section [TODO]
Secure	See Section [TODO]
HttpOnly	See Section [TODO]

7. Return to Step 1.

[TODO: Can parsing a cookie ever fail? Doesn't look like it! Well, unless you count "Set-Cookie: " as a fail...]
When the user agent finishes parsing the set-cookie-string header, the user agent *receives a cookie* from the origin server with name cookie-name, value cookie-value, and attributes cookie-attribute-list.

5.1.1. The Max-Age Attribute

TOC

When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "Max-Age", the user agent MUST process the value string as follows.
If the first character of the value string is not a DIGIT or a "-" character, the user agent MUST ignore the attribute.
If the remainder of value string contains a non-DIGIT character, the user agent MUST ignore the attribute.
Let delta-seconds be the contents of the value string converted to an integer.

If delta-seconds is less than or equal to 0, then append an attribute named Expires (note the name conversion) to the cookie-attribute-list with a value equal to the current date and time.
If delta-seconds is strictly greater than 0, then append an attribute named Expires (note the name conversion) to the cookie-attribute-list with a value equal to the current date and time plus delta-seconds seconds.

5.1.2. The Expires Attribute

TOC

Unfortunately, cookie dates are quite complex for historical reasons. When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "Expires", the user agent MUST process the value string as follows.
If the attribute lacks a value or the value is the empty string, abort these steps.
Using the grammar below, divide the value of the attribute into date-tokens.

```
cookie-date     = date-token-list
date-token-list = date-token [ delimiter date-token-list ]
delimiter       = U+09 / U+20 / U+21 / U+22 / U+23 / U+24 /
                  U+25 / U+26 / U+27 / U+28 / U+29 / U+2A /
                  U+2B / U+2C / U+2D / U+2E / U+2F / U+3B /
                  U+3C / U+3D / U+3E / U+3F / U+40 / U+5B /
                  U+5C / U+5D / U+5E / U+5F / U+60 / U+7B /
                  U+7C / U+7D / U+7E
date-token      = day-of-month / month / year / time / mystery
day-of-month    = 2DIGIT / DIGIT
month           = "jan" [ mystery ] / "feb" [ mystery ] /
                  "mar" [ mystery ] / "apr" [ mystery ] /
                  "may" [ mystery ] / "jun" [ mystery ] /
                  "jul" [ mystery ] / "aug" [ mystery ] /
                  "sep" [ mystery ] / "oct" [ mystery ] /
                  "nov" [ mystery ] / "dec" [ mystery ]
year            = 5DIGIT / 4DIGIT / 3DIGIT / 2DIGIT / DIGIT
time            = 2DIGIT ":" 2DIGIT ":" 2DIGIT
mystery         = (anything except a delimiter)
```

Process each data-token sequentially in the order the date-tokens appear in the attribute value:

1. If the found-day-of-month flag is not set and the token matches the day-of-month production, set the found-day-of-month flag and set the day-of-month-value to the number denoted by the

token. Skip the remaining sub-steps and continue to the next token.

2. If the found-month flag is not set and the token matches the month production, set the found-month flag and set the month-value to the month denoted by the token. Skip the remaining sub-steps and continue to the next token.

3. If the found-year flag is not set and the token matches the year production, set the found-year flag and set the year-value to the number denoted by the token. Skip the remaining sub-steps and continue to the next token.

4. If the found-time flag is not set and the token matches the time production, set the found-time flag and set the hour-value, minute-value, and second-value to the numbers denoted by the digits in the token, respectively. Skip the remaining sub-steps and continue to the next token.

Abort these steps if

*at least one of the found-day-of-month, found-month, found-year, or found-time flags is not set,

*the day-of-month-value is less than 1 or greater than 31,

*the year-value is less than 1601 or greater than 30827,

*the hour-value is greater than 23,

*the minute-value is greater than 59, or

*the second-value is greater than 59.

If the year-value is greater than 68 and less than 100, increment the year-value by 1900.
If the year-value is greater than or equal to 0 and less than 69, increment the year-value by 2000.
Let the expiry-time be the date whose day-of-month, month, year, hour, minute, and second (in GMT) are the day-of-month-value, the month-value, the year-value, the hour-value, the minute-value, and the second-value, respectively.
If the expiry-time is later than the last date the user agent can represent, the user agent MAY replace the expiry-time with the last representable date.
If the expiry-time is earlier than the first date the user agent can represent, the user agent MAY replace the expiry-time with the first representable date.
Append an attribute named Expires to the cookie-attribute-list with a value equal to expiry-time.

5.1.3. The Domain Attribute

TOC

When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "Domain", the user agent MUST process the value string as follows:

*If the value string is empty, then ignore the attribute. [TODO: Add a test for this with multiple Domain attributes.]

*If the first character of the value string is ".", then append an attribute named Domain to the cookie-attribute-list with a value equal to value string excluding the leading "." character.

*If the first character of the value string is not ".", then append an attribute named Domain to the cookie-attribute-list with a value equal to value string and mark the attribute as host-only.

*[TODO: Deal with domains that have an insufficient number of fields.]

*Otherwise, ignore the attribute.

5.1.4. The Path Attribute

TOC

The user agent MUST use the following algorithm to compute the default-path of a cookie:

1. Let uri-path be the path portion of the URI from which the user agent received the cookie. [TODO: Define this more precisely.]

2. If the first character of the uri-path is not a "/" character, output "/" and skip the remaining steps.

3. If the uri-path contains only a single "/" character, output "/" and skip the remaining steps.

4. Output the characters of the uri-path from the first character up to, and but not including, the right-most "/".

A request-path path-matches a cookie-path if the cookie-path is a prefix of the request-path and at least one of the following conditions hold:

*The last character of the cookie-path is "/".

*The first character of the request-path that is not included in the cookie-path is a "/" character.

When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "Path", the user agent MUST process the value string as follows:

*If the value string is empty, then append an attribute named Path to the cookie-attribute-list with a value equal to default-path of the cookie. [TODO: Is this right if there are more than one path attribute?]

*If the value string is non-empty and the first character is "/", then append an attribute named Path to the cookie-attribute-list with a value equal to value string.

*Otherwise, ignore the attribute.

[TODO: Test \ ? ; # $ % etc]

5.1.5. The Secure Attribute

TOC

When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "Secure", the user agent MUST append an attribute named Secure to the cookie-attribute-list with an empty value regardless of the value string.

5.1.6. The HttpOnly Attribute

TOC

When the user agent receives a cookie attribute with a name string that case-insensitively matches the string "HttpOnly", the user agent MUST append an attribute named HttpOnly to the cookie-attribute-list with an empty value regardless of the value string.

TOC

5.2. Storage Model

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

- *name (a sequence of bytes)
- *value (a sequence of bytes)
- *expiry (a date)
- *domain (a cookie-domain)
- *path (a sequence of bytes)
- *creation (a date)
- *last-access (a date)
- *persistent (a Boolean)
- *host-only (a Boolean)
- *secure-only (a Boolean)
- *http-only (a Boolean)

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

1. Create a new cookie based on the parsed Set-Cookie header:
    1. Create a new cookie with the following default field values:
        - *name = the cookie-name
        - *value = the cookie-value
        - *expiry = the latest representable date
        - *domain = the request-host
        - *path = the cookie's default-path
        - *last-access = the date and time the cookie was received

*persistent = false

*host-only = true

*secure-only = false

*http-only = false

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

   expiry If the cookie-attributes contains at least one valid Expires attribute, store the expiry-value of the last such attribute in the expiry field. Store the value true in the persistent field. [TODO: Test that this really works when mixing Max-Age and Expires.]

   domain If the cookie-attributes contains at least one Domain attribute, store the value of the last such attribute in the domain field. Store the value false in the host-only field. [TODO: Reject cookies for unrelated domains.] [TODO: If the URL's host is an IP address, let Domain to be an IP address if it matches the URL's host exactly, but set the host-only flag.]

   path If the cookie-attributes contains at least one Path attribute, store the value of the last such attribute in the path field.

   secure-only If the cookie-attributes contains at least one Secure attribute, store the value true in the secure-only field.

   http-only If the cookie-attributes contains at least one HttpOnly attribute, store the value true in the http-only field.

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

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

The user agent MUST evict a cookie from the cookie store if A cookie exists in the cookie store with an expiry date in the past.

The user agent MAY evict a cookie from the cookie store if the number of cookies sharing a domain field exceeds some predetermined upper bound (such as 50 cookies). [TODO: Explain where 50 comes from.]
The user agent MAY evict cookies from the cookie store if the cookie store exceeds some maximum storage bound (such as 3000 cookies). [TODO: Explain where 3000 comes from.]
When the user agent evicts cookies from the cookie store, the user agent MUST evict cookies in the following priority order:

1. Cookies with an expiry date in the past.

2. Cookies that share a domain field more than a predetermined number of other cookies.

3. All other cookies.

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

> NOTE: This document does not define when "the current session is over." Many user agents remove non-persistent cookies when they exit. However, other user agent expire non-persistent cookies using other heuristics.

5.3. The Cookie Header

TOC

When the user agent generates an HTTP request for a particular URI, the user agent SHOULD attach exactly one HTTP header named Cookie if the cookie-string (defined below) for that URI is non-empty.
A user agent MAY elide the Cookie header in its entirety if the user agent is configured to block sending cookie for a particular request. For example, the user agent might wish to block sending cookies during "third-party" requests.
The user agent MUST use the following algorithm to compute the cookie-string from a cookie store and from a URI:

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

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

   *The cookie's path field must path-match the URI's path.

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

*If the cookie's secure-only field is true, then the URI's scheme must denote a "secure" protocol.

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

*If the cookie's http-only field is true, then include the cookie unless the cookie-string is begin generated for a "non-HTTP" API. (Note that this document does not define which APIs are "non-HTTP".)

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

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

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

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

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

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

    1. If the cookie's name field is non-empty, output the cookie's name field followed by the character U+3D ("=").

    2. Output the cookie's value field.

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

6. Implementation Considerations

TOC

6.1. Set-Cookie Content

TOC

An origin server's content should probably be divided into disjoint application areas, some of which require the use of state information. The application areas can be distinguished by their request URLs. The Set-Cookie header can incorporate information about the application areas by setting the Path attribute for each one.
The session information can obviously be clear or encoded text that describes state. However, if it grows too large, it can become unwieldy. Therefore, an implementor might choose for the session information to be a key to a server-side resource. [TODO: Describe briefly how to generate a decent session key.]
[TODO: We could recommend that servers encrypt and mac their cookie data.]
[TODO: Mention issues that arise from having multiple concurrent sessions.]

6.2. Implementation Limits

TOC

Practical user agent implementations have limits on the number and size of cookies that they can store. General-use user agents SHOULD provide each of the following minimum capabilities:

- 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). [TODO: Validate]

- At least 50 cookies per domain. [TODO: History lesson]

- At least 3000 cookies total.

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, the cookie must be discarded, not truncated.
Applications should use as few and as small cookies as possible, and they should cope gracefully with the loss of a cookie. [TODO: Could mention latency issues that arise from having tons of cookies.]

7. Security Considerations

TOC

7.1. Clear Text

TOC

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

1. Any sensitive information that is conveyed in in the headers is exposed to an eavesdropper.

2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.

3. A malicious client could alter the Cookie header before transmission, with unpredictable results.

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

7.2. Weak Isolation

TOC

[TODO: Weak isolation by port.]
[TODO: Weak isolation by scheme (e.g., ftp, gopher, etc).]

7.3. Cookie Spoofing

TOC

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

TOC

8. Other, Similar, Proposals

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

Appendix A. Acknowledgements

TOC

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

Appendix B. Tabled Items

TOC

Tabled items:

*Public suffix.

Author's Address

TOC

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